Csernáth Géza

Bevezetés a számítógép-architektúrákba

Scientia Kiadó

Csernáth Géza

Bevezetés a számítógép-architektúrákba

Csernáth Géza

Bevezetés a számítógép-architektúrákba

Scientia Kiadó
Kolozsvár ▪ 2024

sapientia tankönyvek

műszaki tudományok

Megvalósult a Magyar Kormány támogatásával

Miniszterelnökség Nemzetpolitikai Államtitkárság

Bethlen Gábor Alap

Kiadja a
Scientia Kiadó
400112 Kolozsvár, Mátyás király (Matei Corvin) u. 4.
Tel./fax: +40-364-401454, e-mail: scientia@kpi.sapientia.ro
www.scientiakiado.ro

Felelős kiadó:
Sorbán Angella

Lektor:
Dr. Domokos József

Borítóterv:
Tipotéka Kft.

Kiadói koordinátor:
Szabó Beáta

A szakmai felelősséget teljes mértékben a szerző vállalja.

Descrierea CIP a Bibliotecii Naţionale a României
CSERNÁTH, GÉZA
Bevezetés a számítógép-architektúrákba / Csernáth Géza. - Cluj-Napoca : Scientia, 2024
Conţine bibliografie
ISBN 978-606-975-092-6

004

Tartalomjegyzék

Cuprins

Contents

KÖSZÖNETNYILVÁNÍTÁS

Köszönöm mindazon nagyszerű embereknek, akik a könyv megírásában segítettek és támogattak: Ferencz Katalin doktorandusznak, a Sapientia EMTE Marosvásárhelyi Kar oktatójának az ábrák és táblázatok átszerkesztésében, a számrendszereket bemutató fejezet összeállításában, valamint a Logisim szimulációs környezet használati utasításának megírásában nyújtott segítségét; feleségemnek, Csiszér Zsuzsa pszichológusnak és gyerekeimnek, Eszternek és Áronnak, valamint tágabb családomnak a támogatását és türelmét, amellyel a könyv megírásának másfél éve alatt a családunk életében jelentkező hiányomat elviselték; dr. Túrós László kollégámnak, a Sapientia EMTE Marosvásárhelyi Kar adjunktusának a könyv formátumának és szerkesztésének lehetőségeit és módszereit taglaló magyarázatait; dr. Domokos Józsefnek, a Sapientia EMTE Marosvásárhelyi Kar dékánjának és dr. Szabó Lászlónak, a Sapientia EMTE Marosvásárhelyi Kar Villamos Tanszéke vezetőjének motiváló támogatását, Csenteri Barna szoftvermérnök kollégámnak az assembler fejlesztőkörnyezet programozásában nyújtott segítségét és dr. Haller Piroskának, volt egyetemi tanárnőmnek, hogy tanulmányi éveim alatt feltárta előttem a számítástechnika részleteit, összefüggéseit, valamint az egyetemi éveket meghaladóan nyújtott lelkesítő támogatását.

Marosvásárhely, 2023. október 7. Csernáth Géza

BEVEZETŐ

Könyvünk fő célkitűzése megismertetni olvasóinkat a számítógép-architektúrák alapfogalmaival. Fejezeteinek tanulmányozásához elegendőek a gimnáziumi évek alatt elsajátított fizikai és matematikai alapfogalmak, ezekre építjük magyarázatainkat, és ezekkel kapcsolatban hangsúlyozzuk a bonyolultabb rendszerek működésének értelmezését elősegítő elvonatkoztatási (absztraktizációs) képesség kialakításának és fejlesztésének szükségességét.

Kezdjük a számítástechnika kialakulásának történetével, folytatjuk a bináris számábrázolás, majd logikai és aritmetikai műveletek tárgyalásával. Bemutatjuk a félvezető technológia és gyártás legfontosabb elveit és lépéseit, rávilágítunk a technológiai összefüggésekre az áramköri megvalósítások és elvi kialakítás között, majd továbblépünk az egyszerű logikai áramkörök működésének alapelveihez.

Az alapok megteremtése után rátérünk a számítástechnikai rendszer áramköreinek ismertetésére. Itt derül ki, milyen áramkörökből épül fel és hogyan működik egy aritmetikai-logikai egység vagy egy memória, valamint megtudható, hogyan valósul meg az utasítás- és adattárolás, majd a végrehajtás ütemezése. Ezután a számítástechnikai rendszer áramköreit bonyolultabb struktúrákba rendezzük, feltárva az így kialakuló rendszer szinkronban történő működését a programvégrehajtás és adatáramlás folyamatán keresztül.

Rövid kitérőt teszünk a különböző számítógép-architektúrák irányába, kiemelve a rendszerek fejlődése során bekövetkező változásokat. Végül a számítástechnikai rendszeren belüli adatáramlás gyorsításának lehetőségeit vizsgáljuk a közvetlen memóriahozzáférés-vezérlő, valamint megszakításvezérlő áramkörök felhasználásával.

A tárgyalt témákhoz kapcsolódó gyakorlatainkban az adatáramlást megvalósító eszközök működésének alapjait szimulációs környezetben vizsgáljuk. A szimulált számítástechnikai rendszerre lehetőség van saját programot fejleszteni, azt lefordítani és futtatni, miközben akár az elemi logikai kapuk szintjéig lekövethető egy-egy utasítás végrehajtásának folyamata.

Reméljük, olvasóink haszonnal forgatják majd könyvünk lapjait, találnak benne érdeklődésüket felkeltő részleteket, amelyek hasznukra lesznek számítástechnikai ismereteik megalapozásában és bővítésében.

1. Alapfogalmak

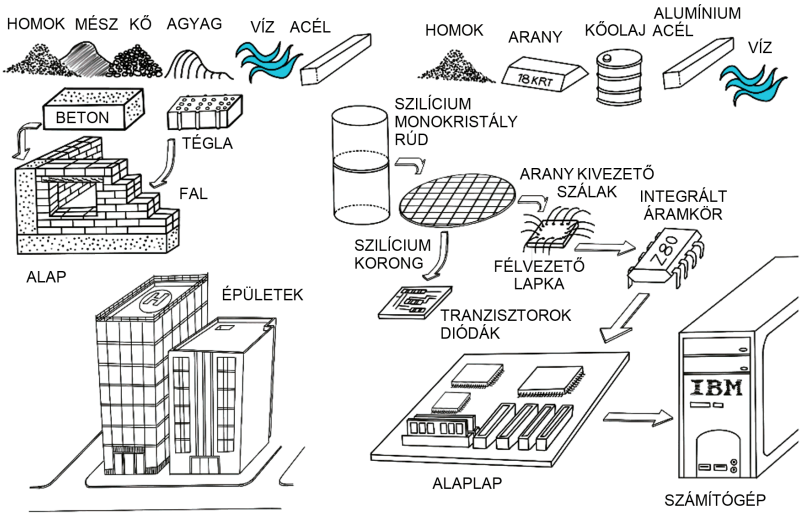
1.1. Számítógép-architektúrák – szóeredet

A számítástechnikai rendszerek felépítését leíró és meghatározó elméleti és gyakorlati szabályok, módszerek, műszaki megoldások elbeszélési, megjelenítési módozatainak összességét illetjük ma a számítógép-*architektúrák* megnevezéssel.

Az architektúra szó görög eredetű szavak összeolvadásából keletkezett, majd vélhetően latin közvetítéssel került át a magyar nyelvbe. A görög szavak, jelentésük *archi* – mester, vezető, valamint *tekton* – építő, ács stb., összeolvadásából keletkezik az *architekton* – építőmester szó. Latinul *architectus* formában használatos, ekkor már az épületeket tervező személyek megnevezésére is használják.

1.2. Anyagok, hasonlóságok, különbségek

Érdekes módon a modern számítástechnikai rendszereket alkotó integrált áramkörök alapanyagai és az építkezésben használt anyagok között több hasonlóságot is felfedezhetünk. Az alábbi, 1.1. ábra mutatja az összefüggéseket:



1.1. ábra. *Hasonlóságok és különbségek az építészet és a számítástechnikai rendszerek között*

Az építkezéseknél kiemelten használt a homok, a mész, a kavics, az agyag, a víz és az acél. A beton és a tégla az alapanyagok vízzel kevert, majd a száradás során kristályosodó anyagszerkezetű változatai.

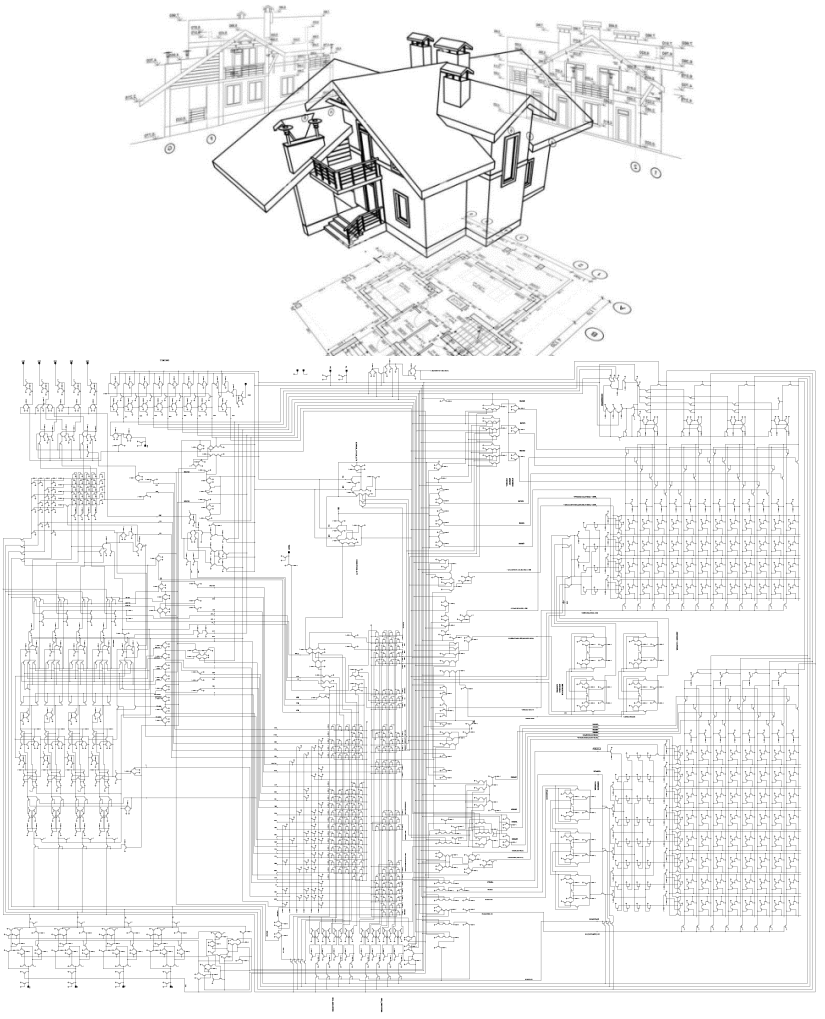
Az integrált áramkörök alapanyagát a kvarchomok képezi, de a kialakításukhoz használnak még aranyat, kőolajszármazékokat, rezet és acélt is. A gyártási folyamatokat vizes fürdők, lemosások, anyagfeloldások tarkítják. A részben hasonló alapanyagok miatt is érdekes ma párhuzamot vonni az épületek formai és szerkezeti sajátosságait, felépítésük módozatait meghatározó épületarchitektúra és a számítástechnikai rendszerek felépítését meghatározó számítógép-architektúra megnevezések között. De a hasonlóság még további részletekben is felfedezhető, ha arra gondolunk, hogy az épületeknek alapra van szükségük mechanikai stabilitásuk biztosításához, a számítógépeknek meg alaplapra az őket alkotó áramköri elemek egybekapcsolásához. Ahogy egy falat téglákból vagy a betont alkotó homoknak és kavicsoknak mészkristályokkal összekötött struktúráiból építenek fel, úgy épülnek mikroszkopikus szilíciummezőket összekapcsoló rézvezetékek egymás fölé létrehozott rétegeiből az integrált áramkörök.

Az integrált áramkör (IC – Integrated Circuit) megnevezés olyan, az elektronikában használatos alkatrészre utal, amely több passzív (ellenállások, kondenzátorok, tekercsek) és aktív (diódák, tranzisztorok) miniatürizált alkatrészhálózatából áll és valamilyen speciális feladatot valósít meg. A számítástechnikai rendszerek esetében az integrált áramköröket többségükben úgynevezett logikai áramkörök jelentik, ezek kombinációiból épül fel egy számítógép mikroprocesszora, memóriája, cím- és adatsínjei, valamint ki- és bemeneti eszközeit csatoló egységei.

1.3. Ábrázolásmódok, hasonlóságok, különbségek

Az építészet és a számítástechnika, azaz a kétféle architektúra közötti hasonlóság ellenpontozásaként hat az ábrázolási és tervezési módszerek közötti különbség. Míg az építészetben az ábrázolás és leírás az épület térbeli és anyagbeli sajátosságait közvetlen módon tükrözi és a felhasznált anyagok egyezményes ábrázolási módon jelennek meg, ezért a tervrajz viszonylag könnyen áttekinthető, addig a számítástechnikai rendszer ábrázolása közvetett és sokrétegű, és a részletek absztrakt ábrázolása nélkül szinte áttekinthetetlen. Például egy épület valamilyen műszaki megoldását a tervező az alkotóelemek és a közöttük lévő térbeli és méretbeli összefüggések, tulajdonságok ábrázolásával közvetíti. Ha viszont arra gondolunk, hogy egy ma kapható mikroprocesszor több százmillió áramköri elemből épül fel, képtelenség lenne ennek részletes áramköri rajzát áttekintve abból megérteni az egész áramkör működését és megvalósításának módszerét. Ezért nagyon fontos, hogy az áramkörök által megvalósított funkciókat, feladatköröket absztrakt, sematikus (vázlatos, csak a lényeget kiemelő) módon, akár egyezményes jelölési módozatokat felhasználva ábrázoljuk, majd ettől elkülönítve határozzuk

meg, ábrázoljuk és írjuk le a megvalósítási, gyártási módszereket és folyamatokat. Lentebb egy lakóház térbeli ábrázolása és műszaki rajzai, illetve a legendás Intel 4004-es processzor (1971) elektromos kapcsolási rajza látható (1.2. ábra).



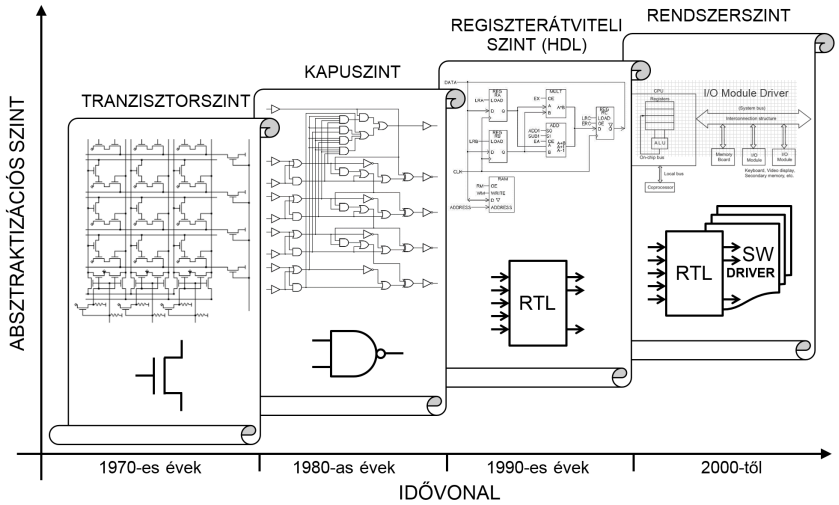
1.2. ábra. *Épület műszaki rajza versus processzor kapcsolási rajza*

A kapcsolási rajzból nehezen tudunk a mikroprocesszor felépítésére és gyártástechnológiájára következtetni, és hasonlóan nehezen látjuk át az áramkörök egyes tömbjeinek funkcionalitását, míg az épületet ábrázoló rajzból könnyebben

megérthető ennek térbeli formája, méretei, a felhasznált anyagok és az építési módszerek egy része.

1.4. Absztraktizáció az ábrázolás szintjén

A mikroprocesszor és általában a komplex logikai áramkörök ábrázolásakor többszintű absztraktizációt (elvonatkoztatási réteget) alkalmaznak. Az 1.3. ábra időben követi az egymásra épülő elvonatkoztatási szintek megjelenését. Ahogy nő a számítástechnikai rendszerek áramköri komplexitása, egyre újabb absztraktizációs szintet kell bevezetni a teljes rendszer átláthatósága érdekében. Míg kezdetben az alkatrészek kevés száma miatt a teljes integrált áramkör kapcsolási rajza ábrázolható az egyes alkatrészek feltüntetésével, ezt fokozatosan felváltja a logikai kapu szintű ábrázolás, majd a funkcionális elemeket kiemelő RTL (Register Transfer Level) és HDL (Hardware Description Language) szimbolikus elemekkel való jelölés.

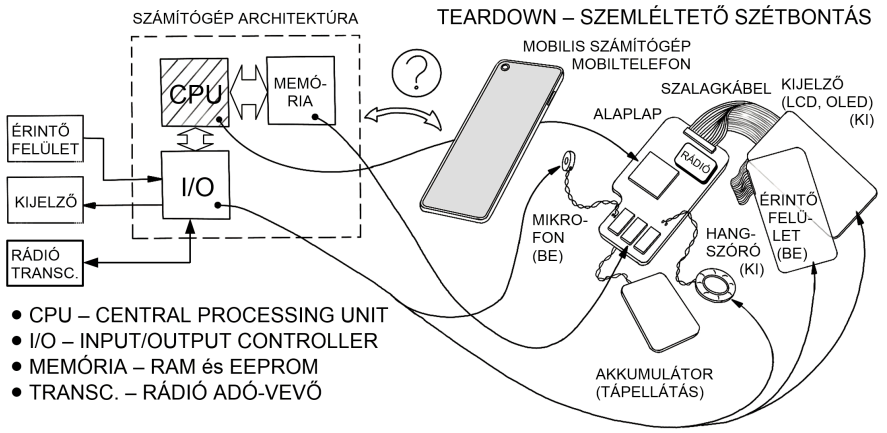


1.3. ábra. *Elvonatkozási szintek megjelenésének időrendi ábrázolása*

Napjainkra a rendszerszintű elemek ábrázolásának szintjén kezdjük el egy integrált áramkör megjelenítését, fokozatosan haladva az egyre részletesebb szintek felé, az ábrák létrehozását és ellenőrzését számítógépes programok segítségével végezzük el.

A következő, 1.4. ábra, egy mobilis számítógépet vagy okostelefont ábrázol. Az ábra bal oldala sematikus, absztrakt jelöléseket (feliratos négyzetek: CPU,

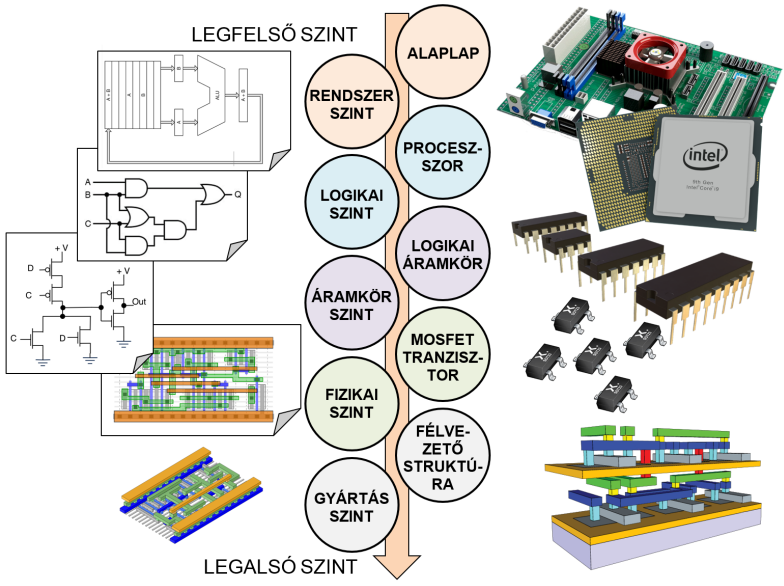
MEMÓRIA, I/O) mutat, a jobb oldala fizikai alkatrészeket jelenít meg (alaplap, processzor, memória-áramkörök, kijelző, érintőképernyő stb.). Az eszköz működésének megértéséhez hiába vizsgálnánk a fizikai alkatrészeket vagy akár az egyes vezetékeken mérhető jelformákat, a részegységek közötti összefüggéseket csak nagyon nehezen tudnánk felfedezni és megérteni. Szükségünk van tehát egy olyan elvont ábrázolási formára, amely feltárja a részegységek közötti összefüggéseket, az adatutak irányát, a részegységek mellé- vagy alárendelt viszonyát.



1.4. ábra. *Számítástechnikai rendszer ábrázolása sematikusan és fizikai alkatrészeit megjelenítve*

1.5. Integráltáramkör-tervezési hierarchia

Az integrált áramkörök tervezéséhez általánosságban a fentről lefele haladó, részletező módszert használják. Először az áramkör rendszerszintű funkcionalitását határozzák meg és írják le. Ezen a szinten sajátos tömbvázlatokat és egyezményes jeleket használnak, a tömbök közötti adatutakat nyilazott vonalak ábrázolják. Ez alatt következik a megvalósított logikai függvények ábrázolása az úgynevezett logikai kapuk egyezményes szimbólumainak felhasználásával. Itt logikai jelek terjedési útjait a kapuk bemeneteit és kimeneteit összekötő vonalak hálózata ábrázolja. Az egyes logikai kapukat alkotó aktív és passzív áramköri elemek ábrázolását a következő, még alacsonyabb szinten, az elektromos kapcsolási rajz segítségével valósítják meg. Itt az elektromos jelek terjedési útját az egyes alkatrészeket összekötő vonalak hálózata ábrázolja. A még alacsonyabb, úgynevezett fizikai szinten az aktív és passzív alkatrészek (ideértve a vezetékek hálózatát is) térbeli kialakítását és elhelyezkedését határozzák meg. A sort végül a gyártástechnológiai módszereket és eljárásokat rögzítő dokumentációs anyag zárja (1.5. ábra).



1.5. ábra. *Az integrált áramkör tervezésének hierarchikus rétegei*

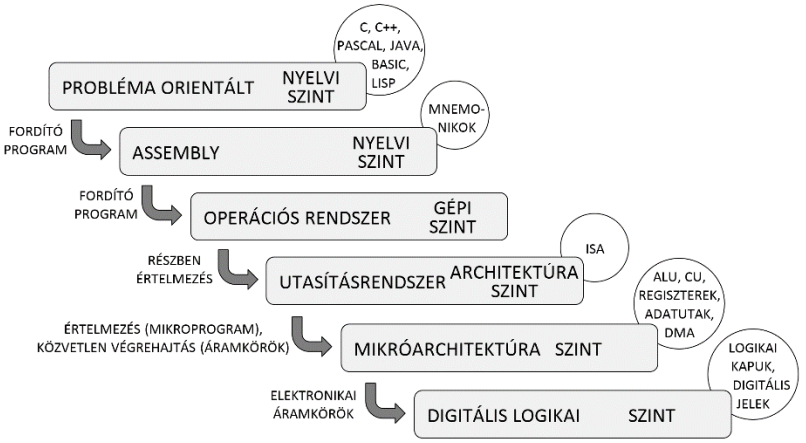
1.6. A többszintes gép fogalma

Az előbbiek alapján a számítástechnikai rendszer, itt röviden gép (számítógép) működésének bemutatása és közvetítése is egy többszintes modell segítségével történik (1.6. ábra). Az alacsonyabb szintek valamennyire kapcsolódnak az ábrázolásmódok esetében vázolt rétegekhez, a magasabb szintek a gép és program közötti összefüggéseket írják le. Minden szintnek megvan a maga megjelenítési sajátossága.

A digitális logikai szint (1. szint) jeleníti meg a számítástechnikai rendszer komplex áramköreit alkotó logikai kapukat és azok kombinációit, illetve a közöttük áramló elektromos jelek sajátosságait. Rögzíti a logikai kapuk működési feltételeit, jelátviteli karakterisztikáját, a jelszintek feszültségértékeit.

A mikroarchitektúra szintjén (2. szint) az előző szint elemeiből összekombinált egységeket találjuk, regisztereket, memóriamodulokat és egyéb komplexebb feladatokat megoldó részeket. Az ALU be- és kimeneteihez regiszterek kapcsolódnak, a közöttük áramló adatok adatutakon vagy síneken terjednek. A számítógép

kiépítésétől függően a mikroarchitektúra szintjén az adatút működését vagy közvetlenül a hardver vezéreli (huzalozott számítógépek), vagy egy mikroprogram (mikroprogramozott számítógépek) utasításaiba kódolt digitális jelzések.



1.6. ábra. *A többszintes gép rétegei és a közöttük lévő összefüggések*

Az utasításrendszer-architektúra (3. szint, Instruction Set Architecture, ISA-szint) szintje a gépi utasításrendszert jelöli, amelyet a mikroprogramozott vagy a hardveres (huzalozott) vezérlő és végrehajtó áramkör értelmez. Ezen a szinten az utasítások végrehajtásának folyamata különféle stratégiák szerint valósulhat meg, a minél gyorsabb, minél kevesebb végrehajtási időt igénylő megoldások irányába optimalizálva.

Az operációs rendszer gép szintje (4. szint) részben a 3-as szint utasításkészletét is felhasználja. A 4. szintű utasítások egy részét az operációs rendszer, más részét közvetlenül a mikroprogram értelmezi. Ezen a szinten találjuk a számítógépet rendszerként működtető eljárásokat, amelyek a számítógépbe beépített részegységek működését egybehangolják, és a felsőbb szintek számára az alsóbb szinteket átlátszóvá teszik (absztraktizálják).

Az 5. szint az első nyelvi szint, itt az utasítások a felhasználó számára jelentéssel bíró rövidítésekből állnak, az alsóbb szintekhez tartozó, binárisan vagy hexadecimálisan ábrázolt kódok itt szimbolikus formában jelennek meg. Alapfokon ezen a szinten lehet a 2–4. szintekre programot írni a bináris vagy hexadecimális kódok nehezen követhető rendszerét felváltó kényelmesebb módon.

A 6. szint nyelveit a komplex alkalmazási feladatokat megoldó programozóknak hozták létre. Komplex logikai összefüggéseket, algoritmustípusok megfogalmazására alkalmas utasításszavakat tartalmaznak legtöbbször angol nyelven.

Használatukhoz ismerni kell az adott nyelv szavait és nyelvtanát (szintaxis). Ilyen a C, C++, Pascal, Java, LISP, Prolog és számos másik nyelv. A magas szintű nyelvekben egy utasításszó fordítása több tíz vagy száznál is több assembler szintű szót jelent.

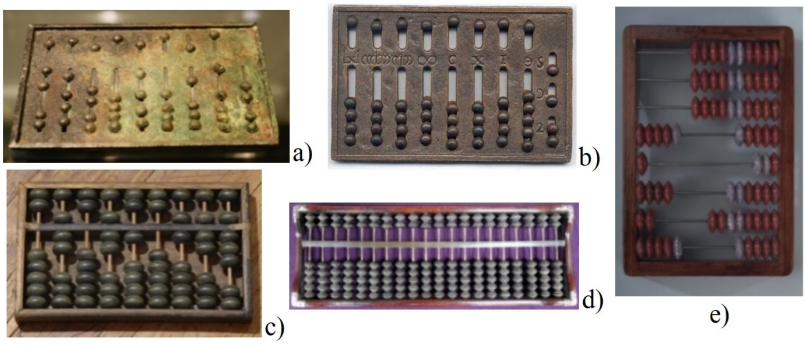
- computer architecture in general terms
- multilevel computer architecture

2. Abakuszok, számológépek, kódfejtők, számítógépek, processzorok

2.1. Ókori szerkezetek, abakuszok, trigramok

Mielőtt a ma ismeretes fogalmaink szerinti számítógépekről lenne szó, meg kell vizsgálnunk az eszköz eredetét is. A kifejezés is azt sugallja, hogy olyan eszközökről, szerkezetekről van szó, amelyek számításokat végeznek. Kezdetben egyszerűbb vagy bonyolultabb matematikai műveleteket végeznek el, később azonban szimbolikus vagy teljesen elvont feladatokat is képesek megoldani, eljutva a bonyolult folyamatok vezérléséig.

Az emberi történelemben jegyzett legrégebbi nagy kultúrák idejére visszatekintve (sumér, babiloni) már időszámításunk előtt 2700-tól utalás történik egy sumér szerkezetre, amelyet számítások elvégzésére használnak (feltehetően 60-as számrendszerben). Történészek szerint az óbabiloni vésett, írásos kőtáblákon fennmaradt egyik piktogram is ennek az emlékét őrzi. Pontos keletkezési ideje nem ismert. A szerkezet ma is használt neve görög (abax – tábla), majd római (abacus) közvetítéssel érkezik nyelvünkbe – *abakusz*. Az ókori Egyiptomban is felbukkan a nyoma, majd jóval későbbről datálódik az ókori Görögország területén megtalált, időszámításunk előtti 5. századból származó Szalamisz szigeti tábla. Ázsiai területen először a 12. században, Kínában bukkan fel *suanpan* néven, majd innen terjed tovább Japánba, ahol a 14. századtól datálhatóan *soroban* néven ismerik és használják. Sokkal később, az újkorban, Oroszországban is megjelenik *stchety* néven. Bár kivitelükben különbséget mutatnak (míg a római, kínai, japán, verziók



2.1. ábra. *Különböző abakuszok megjelenésük időrendi sorrendjében: a) görög, b) római, c) kínai suanpan, d) japán soroban, e) orosz stchety*

függőleges pálcákra fűzött golyókat használnak, addig az orosz változat vízszintes pálcákra fűzött golyókat alkalmaz) működési elvükben megegyeznek (2.1.ábra). A számokat jobbról balra haladva, növekvő helyértékek szerint jelenítik meg: egyesek, tízesek, százasok stb. oszlopaként. A matematikai alapműveleteket a helyérték szerinti golyók átcsoportosításával végzik. Nagy előnye, hogy a számítás elvégzése láthatóvá, követhetővé és megismételhetővé válik, ily módon erősítve a bizalmat a szerkezetet használó kereskedők között. Rendkívüli népszerűségét jelzi, hogy Ázsia nagy részében a 20. század második felében is használták a kereskedelemben. Hátránya viszont, hogy a számítás elvégzéséhez a mozgatható golyók pozíciójának és az alapműveletek végrehajtási módszerének alapos ismerete és közvetlen, lépésenkénti beavatkozás szükséges.

Későbbi keletkezésű szerkezetről i. e. 82-ből maradt feljegyzés, amelyet feltehetően Rodosz szigetén készíthettek és csillagászati számítások elvégzésére használták. Az antikytherai mechanizmusnak (2.2. ábra, tinyurl.com/46zthpb4) nevezett szerkezet fogaskerekekből és tengelyekből áll, nem az abakusz továbbfejlesztése. A fogaskerekek bizonyos pozíciókban való együttállásai utalnak az eredményre, például a csillagok állására és az abból levezethető helyi időre.

2.2. ábra. *Az antikytherai mechanizmus*

A 11. század elején a kínai Shou Yong filozófus és gondolkodó, a kínai mitologikus és filozófiai hagyományokban már i. e. 300-tól felbukkanó (de ismeretlen eredetű), úgynevezett hexagramokból levezetett egy újabb jelölési rendszert: a

trigramokat vagy báguát. Amennyire ma megállapítható, írásában nem matematikai megközelítésben használja. A hexagramokat alkotó folytonos és két szakaszból álló vonalak a yin (két szakasz) és yang (folytonos) princípiumokat és az ezek keveredéséből létrejövő kifejezéseket rögzítik grafikusan kódolt formában. King Wen (Wen király) szimbólumaiként vagy kifejezéseiként is emlegetik ezt a jelölésformát.

A hexagramokat 1-től 64-ig terjedő, egyenként hat folytonos vagy szakaszos vonalból álló szimbólum alkotja. A trigramok 1-től 8-ig terjedő csoportját három folytonos vagy szakaszos vonal alkotja, és nagyon sokféle jelentés fűződik hozzájuk (2.3. ábra, tinyurl.com/3muu9rhk). Érdekes áttekinteni, hogy a duális gyökerű (yin és yang) létmeghatározásból hogyan bomlik ki egy kifejezéshálózat, amely kifejezések a vonalakkal súlyozottan jelölten inkább yin vagy inkább yang jelleggel bírnak. Ez az egyik legrégebbi, két jelet használó kódolási forma.

Trigrammák	Név (kínai)	Név	Tulajdonság	Kép	Családi kapcsolat
☰	Qian	a kreatív	erős	menny	apa
☷	Kun	a fogadó	odaadó	föld	anya
☳	Zhen	az ébresztő	mozgás	mennydörgés, fa	legidősebb fia
☵	Kan	a mély	veszély	víz, felhők	középső fia
☶	Gen	az álló	mozdulatlanság	hegy	legfiatalabb fia
☴	Xun	a lágy	behatolás	szél, fa	legidősebb lánya
☲	Li	a ragaszkodó	fényadó	villám, tűz	középső lánya
☱	Dui	az örömteli	öröm	tó	legfiatalabb lánya

2.3. ábra. *A kínai trigramok kódolási rendszere és jelentéshálója*

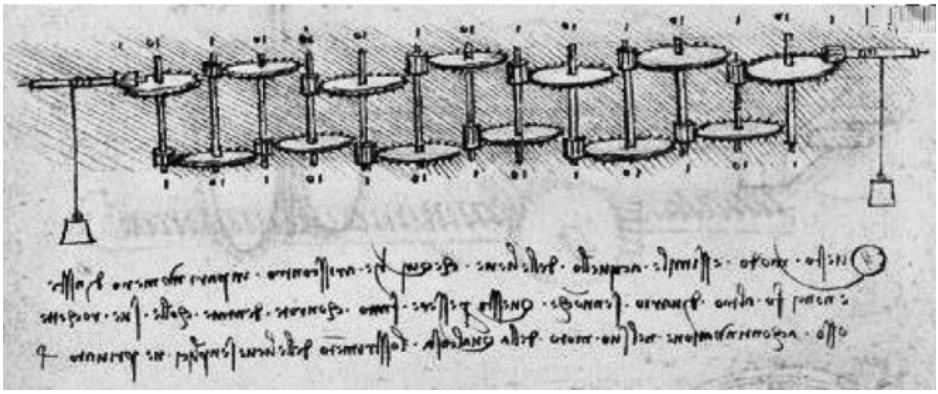
- ancient computing devices
- hexagram trigram Leibniz

2.2. Mechanikus számoló szerkezetek

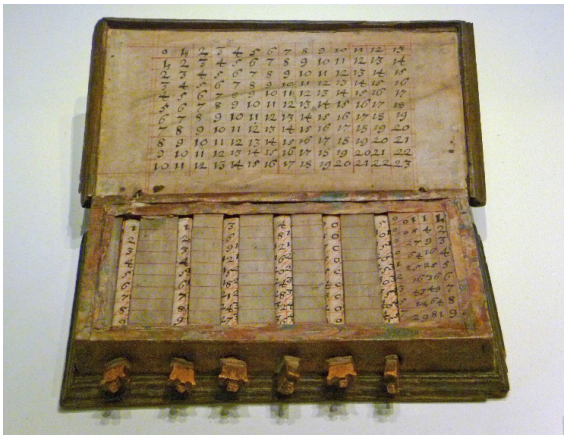
Folyamatos tehát az igény egy olyan berendezés elkészítésére, amely a számítások elvégzésének terhét átveszi, használata csak az operandusok beállítását és a művelet kiválasztását igényli.

A két ősgéptípus megemlítése nem véletlen, ugyanis Leonardo da Vinci kéziratai között 1502-re datálható feljegyzésre bukkantak, amely egy fogaskerekek összekapcsolásával megvalósítható számológép ötletét vezeti fel (2.4. ábra, tinyurl.com/bdftxapz). John Napier skót matematikus és politikus 1617-ben, Wilhelm

Schickard tübingeni tudós pap, csillagász 1623-ban a négy alapművelet elvégzésére képes eszközt készített. Mindkét eszköz továbblépést jelent az abakuszok családjához képest, az eredmény az operandusok beállítása után bizonyos logika szerint közvetlenül leolvasható.



2.4. ábra. *Leonardo da Vinci vázlata fogaskerekes számoló berendezésről*



2.5. ábra. *John Napier számológépe*

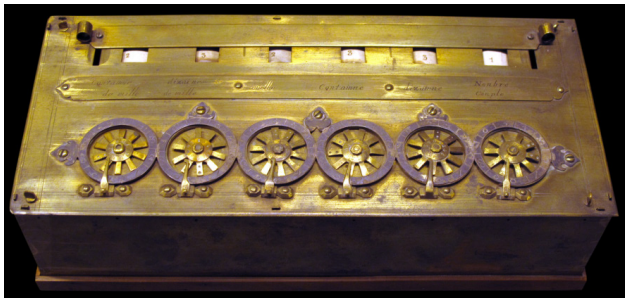
Napier gépe fadobozban elhelyezett, forgatható fahengerekből állt. A hengerekre ragasztott papírlapokon, csoportokban felírt számokkal teli táblázatok találhatók (2.5. ábra, tinyurl.com/5n72ybur). A számítás eredményét a megfelelően beállított hengerekről kellett leolvasni. Napier szerkezete Kínába is eljutott, de elterjedése nem múlta felül a suanpanét.

2.6. ábra. *Wilhelm Schickard számológépe*

Schickard összeadógépe (2.6. ábra, tinyurl.com/3m42w8p7) fogaskerekekkel működött, a Napier-gép utódjának tekinthető, ugyanis a számokkal teli papírlapokkal borított fahengerek itt is megvannak. Forgatásuk viszont fogaskerekekkel összehangolt, az operandusok és az adott alapművelet kis tárcsákon való beállítása után az eredmény közvetlenül leolvasható, ha tudjuk, hogy a réselt, fazsalukkal takart számhengerek melyikét kell figyelnünk. Schickard szerkezetét a feljegyzések szerint barátjának, a csillagász Keplernek készítette bonyolult csillagászati számításainak megkönnyítésére.

Blaise Pascal francia tudós 1642-ben – könyvelő édesapja munkáját segítendő – elkészítette az összeadógépet, amely első változatában hatjegyű, majd későbbi modelljeiben nyolcjegyű számok összeadására, kivonására volt képes (2.7. ábra, tinyurl.com/4xrrk45x). Pascal készülékének nagy újítása elődjeihez képest a helyi

értékek átvitelében nyilvánul meg, amit addig egyetlen szerkezet sem valósított meg. Pascal több összeadógépet is készített, egy pár eredeti példány – többek között francia múzeumokban – ma is megtekinthető.



2.7. ábra. *Blaise Pascal összeadó-kivonó gépe*

Harminc évvel később (1673) Gottfried Wilhelm Leibniz német matematikus Pascal találmányát a szorzás műveletének hozzáadásával szerette volna továbbfejleszteni, ezt azonban a Pascal-féle szerkezet csak egy irányba forgó fogaskerekes rendszere nem tette lehetővé. Leibniz két évtizedes munkával, átvitelszelektáló fogasrúd feltalálásával és alkalmazásával két prototípust is készített, amelyek az összeadás és kivonás mellett szorzás és osztás műveleteket is elvégeztek a sorozatos összeadás, illetve kivonás műveletre bontás módszerével (2.8. ábra, tinyurl.com/24hv5c4f). Mivel a szerkezet a mechanikai pontatlanságok okán nehézkesen, olykor hibásan működött, gyakorlati alkalmazására nem került sor. Viszont több mechanikai újítása is a későbbi mechanikus számológép-generációk alapját képezte.

2.8. ábra. *Gottfried Wilhelm Leibniz szelektáló fogasrudas számológépe*

Leibniz szerepe a számítástechnika történetében a számológépe fejlesztésén túl is meghatározó, a kínai trigramok (lásd a 2.3. ábrát) rendszeréről is inspirálódó,

bináris számrendszerről írt művével (1703) megnyitja az utat a digitális áramkörökre épülő számítástechnikai berendezések kifejlődésének irányába.

Az abakuszokat felváltó mechanikus szerkezetek a 18. század végére képessé váltak számítások önműködő elvégzésére. Az operandusokat és az elvégzendő művelet típusát közvetlen módon állították be rajtuk, majd általában hajtókarok segítségével forgatták a szerkezet fogaskerekeit, amíg a kívánt eredményt meg nem kapták a kijelzésre használt számfeliratos tárcsákon. Operandusok, eredmények eltárolására, későbbi felhasználására nem volt lehetőség. Ehhez szükség volt még egy találmányra.

- Napier bones
- Blaise Pascal calculator

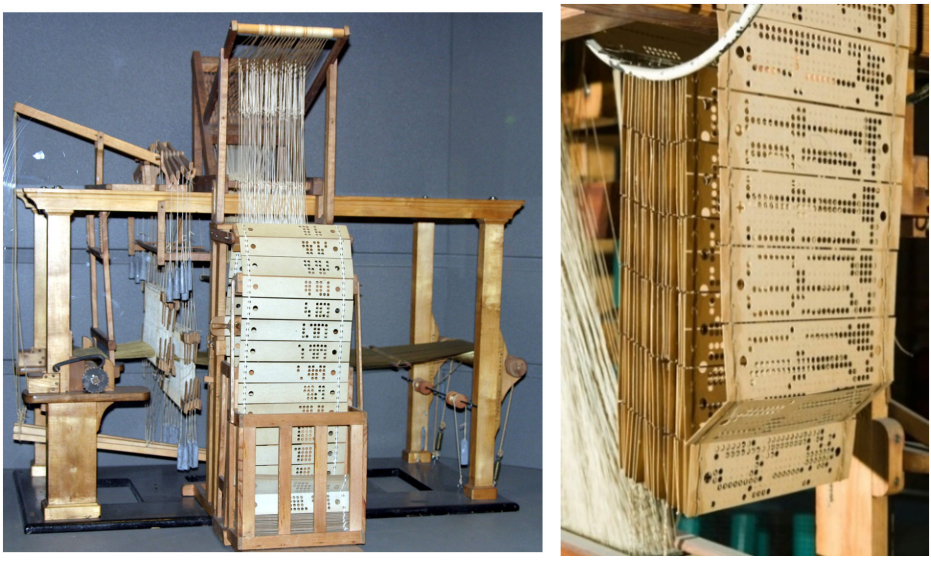
2.3. Lyukkártya-vezérlésű szövőszék, analitikus gép, tabulátor

1728-ban Jean Baptiste Falcon, majd módszerét tökéletesítve 1810-ben Joseph-Marie Jacquard olyan szövőszéket készített, amelyeket vastag papírra, majd később vékony falapocskákra kialakított, úgynevezett lyukkártyák vezéreltek (2.9. ábra, tinyurl.com/3rrn36mu). Bár találmányaikkal a textilipar termelékenységét növelték, mégis nagyban hozzájárultak a későbbi számológép- és számítógép-fejlesztésekhez.

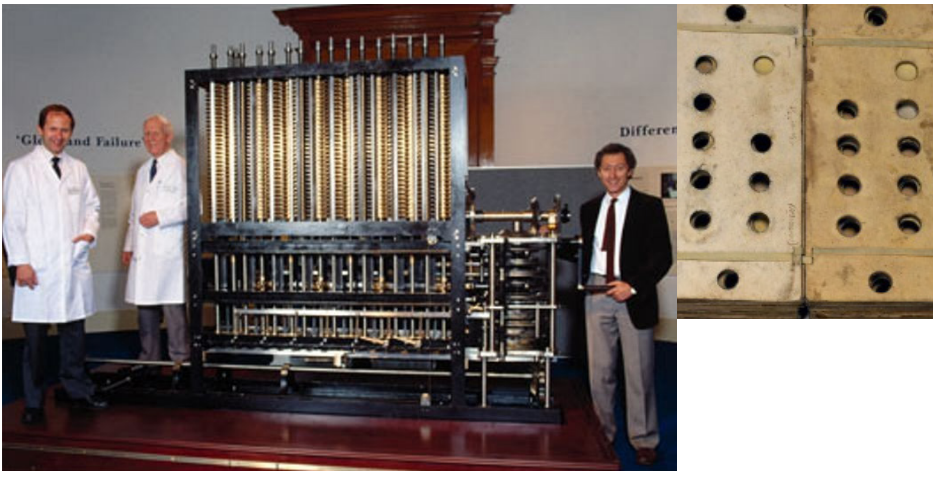
A lyukkártyákon a mintakészítők által megalkotott minta pontjainak soronkénti bontásában éppen a lyukak hiánya felelt meg, a szövőgépek azokat a láncfonalakat emelték fel a nyüstpálcák segítségével, ahol a letapogatórudak a kártya lyukain áthatoltak.

Minden ciklusban a lyukkártyákon tárolt minta alapján szétválogatott (lent maradt, illetve felemelkedett) láncfonalak közé (szádanyílás) kerül be a vetülékfonal. Ezzel megvalósul az első gyakorlatban is alkalmazott, szabadon programozható gépvezérlés. Bár a feltalálók ezt nem látták előre, valójában a bináris adattárolás alapjait fektették le szövőszékeik vezérlési rendszerében.

1833-ban, a Jacquard-szövőgépek elterjedésével egy időben, Charles Babbage angol matematikus elkészítette egy mechanikus számológép tervét, amely az első programvezérelt gép volt, amely elméletileg általános célú matematikai műveletek elvégzését tette lehetővé. Az analitikus gép (Analytical Engine) alkalmas volt a négy alapművelet elvégzésére. Babbage tanulmányozta Jacquard szövőgépének lyukkártyás vezérlését (sok pénzért Franciaországból beszerzett és hazaszállíttatott egy szövőgépet), és a megoldást felhasználta az analitikus gépe programjának rögzítésére és bevitelére. Az analitikus gép felépítése előrevetíti a mai számítógépek főbb elemeit, volt lyukkártyás tárolóegysége, vezérlőegysége, számolóegysége, adatbeviteli és adatkibocsátó egysége.



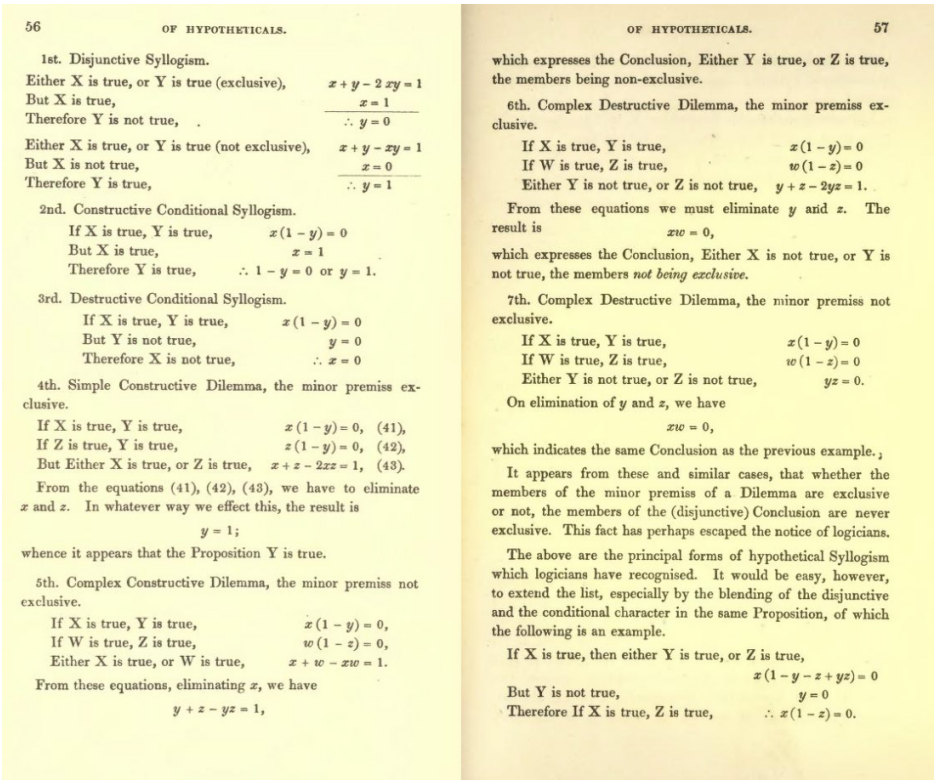
2.9. ábra. *Joseph-Marie Jacquard lyukkártya-vezérlésű szövőszéke*



2.10. ábra. *Charles Babbage 2002-ben megépített analitikus gépe és a programozásához használt eredeti, Babbage-féle lyukkártya*

Babbage munkásságához nagyban hozzájárult Augusta Ada Byron, Lord Byron leányának tudományos tevékenysége, aki elsőként ír tudományos cikket az analitikus gép alkalmazásával kapcsolatban, és elsőként ismeri fel a számítástechnika szinte korlátlan lehetőségeit elméleti programokat fogalmazva a géphez. Azonban

Babbage terve abban a korban technikailag és anyagilag kivitelezhetetlen volt. A gép egészében ezért csak a 20. század végén készülhetett el egy elhivatott kutatókból és mérnökökből álló csoport munkájaként (átadása 2002-ben, 2.10. ábra, tinyurl.com/2y5342pb), bizonyítva, hogy Babbage egy működő gépet álmodott és tervezett meg. Ez a szerkezet jelenti a számítástechnika történetében azt a pontot, ahonnan a programvezérelt számítógépeket származtatjuk. A számoló- és számítógépek története itt elágazik.



56 OF HYPOTHETICALS.

1st. Disjunctive Syllogism.

Either X is true, or Y is true (exclusive), $x + y - 2xy = 1$
But X is true, $x = 1$
Therefore Y is not true, $\therefore y = 0$

Either X is true, or Y is true (not exclusive), $x + y - xy = 1$
But X is not true, $x = 0$
Therefore Y is true, $\therefore y = 1$

2nd. Constructive Conditional Syllogism.

If X is true, Y is true, $x(1 - y) = 0$
But X is true, $x = 1$
Therefore Y is true, $\therefore 1 - y = 0$ or $y = 1$.

3rd. Destructive Conditional Syllogism.

If X is true, Y is true, $x(1 - y) = 0$
But Y is not true, $y = 0$
Therefore X is not true, $\therefore x = 0$

4th. Simple Constructive Dilemma, the minor premiss exclusive.

If X is true, Y is true, $x(1 - y) = 0$, (41),
If Z is true, Y is true, $z(1 - y) = 0$, (42),
But Either X is true, or Z is true, $x + z - 2xz = 1$, (43).

From the equations (41), (42), (43), we have to eliminate x and z. In whatever way we effect this, the result is

$$y = 1;$$

whence it appears that the Proposition Y is true.

5th. Complex Constructive Dilemma, the minor premiss not exclusive.

If X is true, Y is true, $x(1 - y) = 0$,
If W is true, Z is true, $w(1 - z) = 0$,
Either X is true, or W is true, $x + w - xw = 1$.

From these equations, eliminating x, we have

$$y + z - yz = 1,$$

OF HYPOTHETICALS. 57

which expresses the Conclusion, Either Y is true, or Z is true, the members being non-exclusive.

6th. Complex Destructive Dilemma, the minor premiss exclusive.

If X is true, Y is true, $x(1 - y) = 0$
If W is true, Z is true, $w(1 - z) = 0$
Either Y is not true, or Z is not true, $y + z - 2yz = 1$.

From these equations we must eliminate y and z. The result is

$$xw = 0,$$

which expresses the Conclusion, Either X is not true, or Y is not true, the members *not being exclusive*.

7th. Complex Destructive Dilemma, the minor premiss not exclusive.

If X is true, Y is true, $x(1 - y) = 0$
If W is true, Z is true, $w(1 - z) = 0$
Either Y is not true, or Z is not true, $yz = 0$.

On elimination of y and z, we have

$$xw = 0,$$

which indicates the same Conclusion as the previous example.

It appears from these and similar cases, that whether the members of the minor premiss of a Dilemma are exclusive or not, the members of the (disjunctive) Conclusion are never exclusive. This fact has perhaps escaped the notice of logicians.

The above are the principal forms of hypothetical Syllogism which logicians have recognised. It would be easy, however, to extend the list, especially by the blending of the disjunctive and the conditional character in the same Proposition, of which the following is an example.

If X is true, then either Y is true, or Z is true, $x(1 - y - z + yz) = 0$
But Y is not true, $y = 0$
Therefore If X is true, Z is true, $\therefore x(1 - z) = 0$.

2.11. ábra. *G. Boole dolgozatának oldalai logikai függvényekkel*

Babbage még javában dolgozik gépe elméleti tökéletesítésén, amikor 1847-ben George Boole tudományos értekezést ír a később Boole-algebra néven ismertté váló logikai rendszerről (The Mathematical Analysis of Logic, 2.11. ábra, tinyurl.com/yzj3u472). Az írás (csak 86 oldal terjedelmű) a logika történetének meghatározó műve, későbbiekben létfontosságúnak bizonyul a bináris aritmetika áramkörökkel való megvalósításában. Boole közeli barátja, Augustus de Morgan szintén hozzájárul a bináris aritmetika további összefüggéseinek feltárásához a De Morgan-törvényekként ismert szabályok felállításával és bizonyításával.

A 19. század végének technikatörténeti áttekintésekor feltétlen szólnunk kell Herman Hollerith munkásságáról is, aki az Egyesült Államok népszámlálási adatainak feldolgozására egy elektromágnesekkel működtetett, lyukkártyás számológépet fejlesztett ki (2.12. ábra, tinyurl.com/33nyjde2). Az 1890-es népszámlálás adatait így mindössze 6 év alatt (63 millió fő) dolgozta fel 43 gép, mindez 1880-ban (50 millió fő) kézi módszerrel 500 ember közel 8 évi munkáját jelentette. A módszer lényege az volt, hogy a begyűjtött adatokat az operátorok egy sémának megfelelően lyukkártyára vitték át, majd az ily módon kialakított lyukkártyákat a számológépbe betáplálva automatikusan megszámlálták. Hollerith 1896-ban alapított cége (CTR) egy cégcsoport részeként 1924-től az irodai, számítástechnikai eszközeiről híres International Business Machine Corporation (IBM) nevet vette fel.

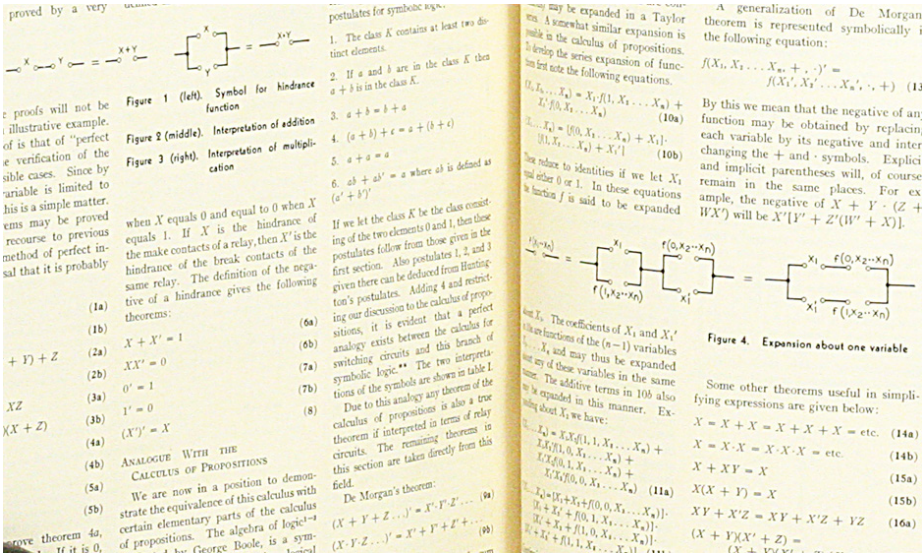
2.12. ábra. *Herman Hollerith lyukkártyás tabulátora*

- Charles Babbage's analytical engine
- The Mathematical Analysis of Logic

2.4. Kódolók és kódfejtők, a számítógép megszületése

A 20. század elejétől több kiemelkedő szerkezet is készül. Az első világháború műszaki kihívásai a különböző mechanikus számológépeknek jósolnak nagy jövőt, tüzérségi lövedékek röppályáinak, térképészeti adatok vagy éppen torpedók útvonalának kiszámításához használják őket. Közös tulajdonságuk, hogy a szerkezetek felhasználói már nem annak alkotói, nem szükséges ismerniük a szerkezet belső működését, csupán használniuk kell az adott számítások elvégzéséhez. Észrevehető,

hogy az adott feladat elvégzésére alkalmas számológépek és az általános felhasználásra szánt szerkezetek külön fejlődnek tovább. A 20. század első felében kialakuló és háborúba torkolló globális hatalmi konfliktusok sajnálatos módon folyamatosan táplálják az igényt olyan szerkezetek megtervezésére és használatára, amelyek a hadi cselekmények sikerét növelhetik. Miközben a titkosításra használt szerkezetek, majd azok megfejtésére titokban kifejlesztett és használt eszközök a tudományos újítások fókuszában állnak, a csatazajba vesznek Konrad Zuse német mérnök és feltaláló első változataiban mechanikus (Z1, Z2), majd később, 1941-ben elkészített harmadik (Z3), elektromechanikus gépei. Mivel ezek teljesen vagy részben elpusztulnak, Németországon kívül szinte egyáltalán nem ismerik őket.



2.13. ábra. *Claude Shannon dolgozata a bináris aritmetika kapcsolókkal való megvalósításáról*

Nagyon fontos mozzanat, hogy 1937-ben az Egyesült Államokban a Massachusetts Institute of Technology egyetemen Claude Shannon megírta *A Symbolic Analysis of Relay and Switching Circuits* című mesteri dolgozatát (2.13. ábra, tinyurl.com/rdespsfb). Ebben a Boole-algebra és a bináris aritmetika kapcsolókkal és relékkel való megvalósítását tárgyalja, megalapozva a digitális áramkörök elméletét.

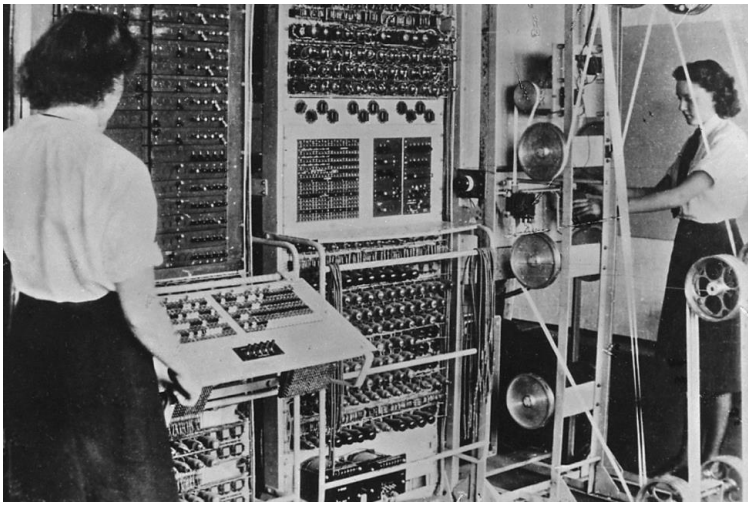
A német hadsereg eközben rendszeresíti az üzenetváltások titkosítására használt szerkezetét, amit a kereskedelmi forgalomban kapható, Enigma nevű titkosítóberendezésre épít (2.14. ábra, tinyurl.com/ydb4w8en). A szerkezet feltalálója Arthur Scherbius német mérnök és feltaláló, aki az első változatot 1918-ban szabadalmaztatja. Az elektromechanikus szerkezet lényege abban áll, hogy a bemeneti jelet (lenyomott karaktert) úgynevezett kódolótárcsákon vezeti keresztül, amelyekben

az egyes karaktereknek megfelelő kapcsolódási pontok vezetékeit rendre más karakterek kapcsolódási pontjai felé vezeti tovább. Minden karakterbevitel után a tárcsák egy pozícióval tovább fordulnak. Így abban az esetben, ha egy szóban egymás után két azonos betű következik, ezek kódolt változatai különböző betűk. Az adott napon az első üzenethez beállított kulcs-, azaz tárcsapozíciók határozzák meg a kódot. A megfejtést a hatalmas mennyiségű lehetséges kulcskombináció nehezíti, amelyet a kódolótárcsák számának növelésével, illetve a szerkezethez kapcsolt dugaszos váltótáblával lehet fokozni. A német hadsereg ezt a szerkezetet módosítja és tökéletesíti (változatait használja az olasz és japán hadsereg is).

2.14. ábra. *Az Enigma nevű üzenettitkosító berendezés*

Talán ez a mozzanat ad újabb lendületet a későbbi számítástechnikai fejlesztéseknek és kutatásoknak. Mivel a titkosított katonai üzenetek megfejtése életbevágó, ezek feltörése még a második világháború előtt elkezdődik. Lengyel kódfejtők Marian Rejewski vezetésével már 1936-ban kidolgozzák a megfejtési módszert, amelyhez francia titkosszolgálati közvetítéssel kapnak kódolási kulcsokat. A második világháború kitörésével az üzenetek megfejtése napi szintű aktualitássá válik a szövetséges hatalmak számára. Rejewski minden elérhető matematikust és kódfejtőt, közöttük a kiemelkedően tehetséges Jerzy Różyckit és Henryk Zygalskit is beveszi a csapatába. Kidolgoznak és megalkotnak egy elektromágneses kapcsolókat használó, „Bomba” nevű szerkezetet, amely a kódolótárcsák algoritmikusan szimulált forgatásával napi akár 100 000 kódkombinációt is képes kipróbálni a bevitt üzenetek megfejtése érdekében. A háború alatti napi üzenetmennyiség azonban még ezt a kapacitást is hamar meghaladja, a lengyel kódfejtő csapat ezért módszereit megosztja a szövetségesekkel, a kódfejtőket kimenekítik a megszállt Lengyelország

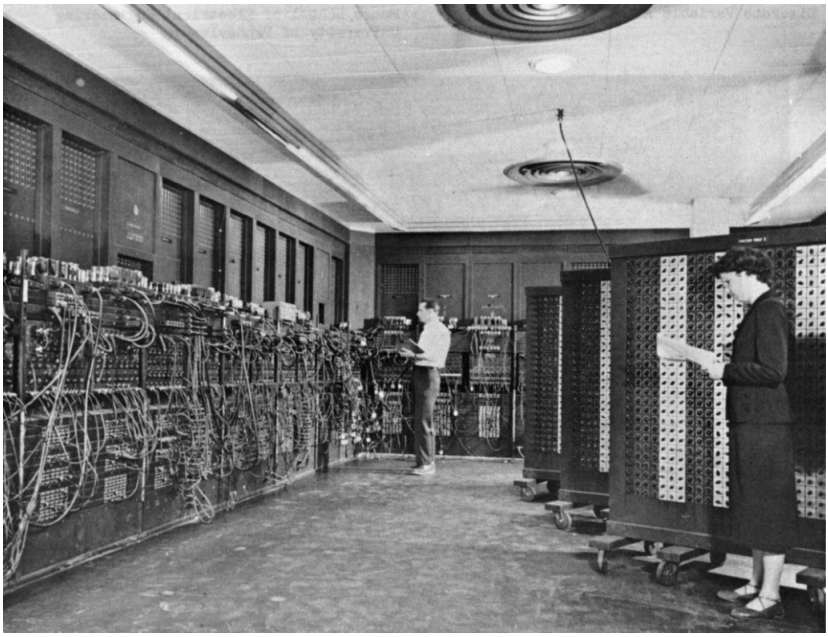
területéről. Világossá válik, hogy a feltörési algoritmust gépesíteni kell. A témával az Egyesült Államok és Anglia is aktívan foglalkozik. 1941-ben itt kapcsolódik be a történetbe Alan Turing angol tudós és matematikus, aki a híres buckinghamshire-i Bletchley Park épületében tudóstársaival (többek között Tommy Flowersszel) és a lengyel kódfejtőkkel közösen továbbfejleszti a „Bombát", több hasonló szerkezet elektromágneses kapcsolókkal kialakított adatutakkal és regiszterekkel való összekapcsolásával. A lehallgatott üzenetek megfejtésének további felgyorsítása érdekében Max Newman angol matematikus vezetésével az angol kormány elindítja a Colossus nevű számítástechnikai berendezés kifejlesztését (2.15. ábra, tinyurl.com/mp9ufk7h). A fejlesztésben aktívan részt vesz Tommy Flowers is. Turing egyes ötleteit és más kódolóberendezések technikai megoldásait felhasználva egy elektroncsövekkel szerelt dekódoló számítástechnikai eszközt építenek, amelyet 1943-ban Colossus Mark 1 néven állítanak üzembe. A gép átmenetet képez az automatizált, algoritmusalapú feladat-végrehajtó eszköz és az általánosan felhasználható számítógép között. Lyukszalagos adatbeviteli rendszerrel, a dekódolási algoritmust végrehajtó elektroncsöves logikai áramkörökkel, eredményszámláló és kijelzőtáblával rendelkezik. A gépnek több változata is megépül, a Colossus Mark 2-t 1944-ben állítják üzembe a normandiai „D" napi partraszállás előtt. Becslések szerint ezek a kódfejtési erőfeszítések százezrek vagy akár milliók életét mentette meg, lerövidítette a háborút, és katonai előnyhöz juttatta a szövetséges hatalmakat.



2.15. ábra. *A Colossus Mark 1 nevű jelfogós számítógép*

- Enigma machine
- Colossus computer

Egy másik híres elektromechanikus gép a Harvard MARK I, melynek terveit Howard Aiken egyesült államokbeli fizikus 1937-ben a Harvard Egyetemen fejleszti ki. Ezeket a terveket mutatja be az IBM-nek. A cég a találmány kifejlesztésének anyagi támogatásáról dönt, így megépítése 1939-ben elkezdődhet. 765 000 elektromechanikus kapcsolóelemet és más szerkezeti megoldást alkalmazva alakították ki az utasítás-végrehajtó rendszerét, a munkamemóriát képező regiszterláncokat és adatutakat. Az adatokat és utasításokat bináris rendszerben kódolják és tárolják két külön memóriatérben (adatmemória és programmemória). Az adatbevitel kapcsolós konzollal történik, a program utasításait lyukszalagról olvassa be a berendezés. 1944-ben a Manhattan terven (az Egyesült Államok titkos atombomba-fejlesztési programja) dolgozó, magyar származású Neumann János (John von Neumann) programot ír rá és fizikai számítások elvégzésére alkalmazza a berendezést. Közben feltehetően tanulmányozza gyengeségeit, és terveket sző a berendezés új elvek alapján való átépítésére. Idővel Aiken kapcsolata az IBM-mel szerzői jogi nézeteltérések miatt megromlott, azonban ez nem akadályozta őt abban, hogy számítógépének további három változatát (Mark II, III és IV) is megépítse.

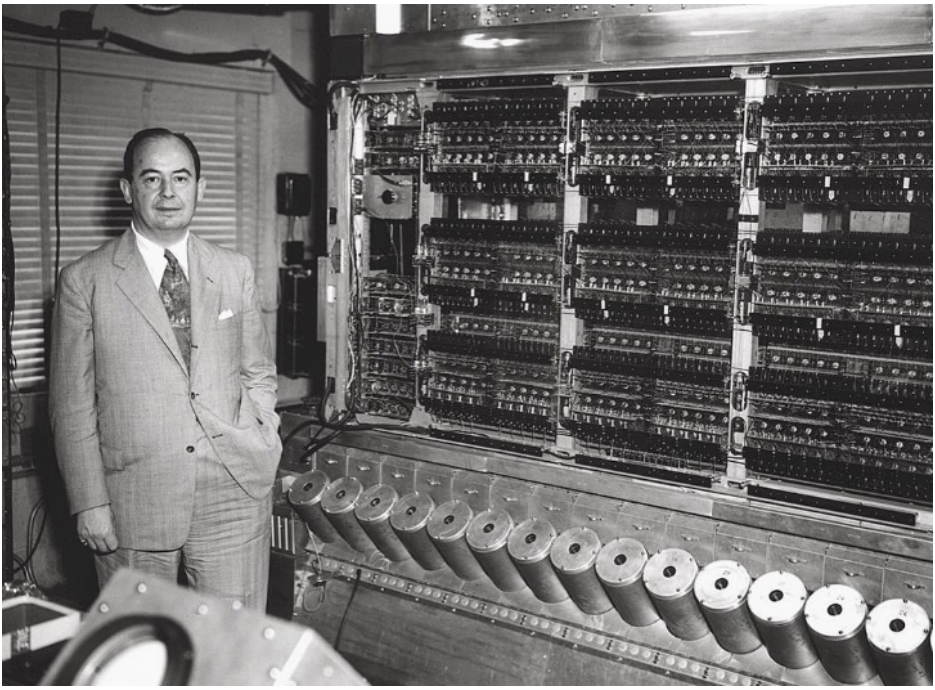


2.16. ábra. *Az ENIAC számítógép és programtároló kapcsolótáblái*

1945–47 között John Mauchly és J. Presper Eckert, a Pennsylvania Egyetem villamosmérnökei az Egyesült Államok hadseregének felkérésére megalkották az első, viszonylag megbízhatóan működő, elektronikus számítógépet, az ENIAC-ot

(2.16. ábra, tinyurl.com/y34vrmse). Elsődleges feladata a ballisztikus lövedékek pályaadatainak kiszámítása volt. Ezzel az addig közel 20 órás számítási feladat megoldását 30 másodpercre csökkentette. Logikai áramköreinek kapcsolóelemeit elektroncsövekből építik meg. Érdekessége, hogy tízes számrendszert használ az adatok ábrázolásához. Az ENIAC (Electronic Numerical Integrator And Calculator) gépet külső kapcsolótáblák segítségével programozták, az éppen végrehajtandó programrészleteket beállító kapcsolótábla-modulokat a berendezés beviteliregiszter-áramköreihez kapcsolták.

Az elektromechanikus elődeinél kb. 2000-szer volt gyorsabb, előállítási költsége, akkori áron, 10 millió dollár volt. Az ENIAC hatalmas helyet foglalt el, 30 m hosszú, 3 m magas és 1 m széles, tömege több mint 30 tonna, és körülbelül 1800 db elektroncsövet tartalmazott.



2.17. ábra. *Neumann János az EDVAC számítógép memóriaegysége előtt*

1946-ban Neumann János kidolgozza a számítógépek hatékony és általános működését leíró, úgynevezett Neumann-elveket, amely szerint a gépnek öt alapvető funkcionális egységből kell állnia: bemeneti egység, memória, aritmetikai egység, vezérlőegység, kimeneti egység. Ezek mellett nagyon fontos, hogy a gép működését a tárolt program elvére kell alapozni. Ez azt jelenti, hogy a gép a program utasításait az adatokkal együtt a központi memóriában, bináris formában tárolja,

és az utasításokat ezek sorrendjében hajtja végre. Ezt az elvet követik napjaink számítógépei is. Tekintsük át a feltételeket:

– A számítógép használja a kettes számrendszert, és legyen teljesen elektronikus. A kettes számrendszert és a rajta értelmezett aritmetikai és logikai műveleteket könnyű megvalósítani kétállapotú áramkörökkel, amelyek a két állapotnak két feszültségszintet feleltetnek meg (1 – magasabb feszültség, 0 – alacsonyabb feszültség).

– A számítógép a központi memóriában tárolt utasításokat egymás után, sorban hajtsa végre.

– A számítógépnek legyen saját regiszteralapú, köztes memóriája is. A működés felgyorsítása érdekében és a hibalehetőségek kizárása okán a felhasználó nem avatkozhat bele a számítás (végrehajtás) menetébe. A köztes memóriában tárolhatók az adatok és az egyes számítások részeredményei, így a gép bizonyos műveletsorokat automatikusan el tud végezni a részeredmények újrafelhasználásával.

– A tárolt program elve: a programot alkotó utasítások kifejezhetők számokkal, azaz adatként kezelhetők. Ezek a központi memóriában tárolhatók. Így a számítógép önállóan, folyamatos kezelői beavatkozás nélkül képes működni, az adatokat és az utasításokat egyaránt a központi memóriából veszi elő.

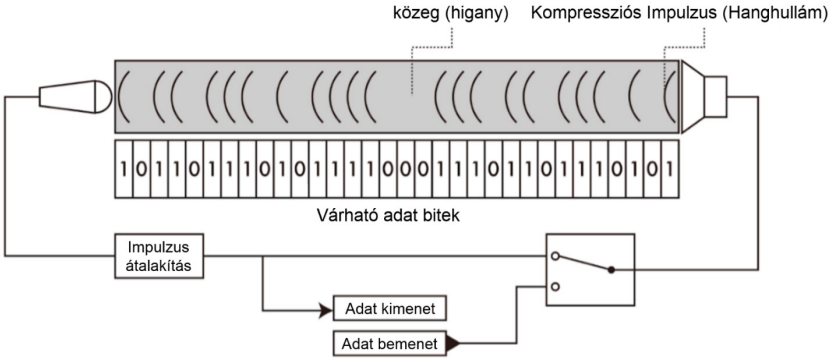
– A számítógép legyen általánosan felhasználható a legkülönbözőbb számítások vagy algoritmusok végrehajtására. Alan Turing bebizonyította, hogy az a számítógép, amely el tud végezni néhány alapvető műveletet, elvileg az alapvető műveletekből összekombinálható bármilyen számítás elvégzésére alkalmas.

Technikatörténeti érdekesség, hogy Mauchly és Eckert még 1944-ben felvetik az EDVAC (Electronic Discrete Variable Automatic Computer, 2.17. ábra, tinyurl.com/mr2e9jab) megépítésének ötletét, de tervük megvalósítását csak később, az ENIAC megépítésének befejezése előtt kezdhetik el. A pennsylvaniai Moore School of Electrical Engineering intézményi kereteiben zajló tervezési és kivitelezési folyamatban tanácsadóként aktívan részt vesz Neumann János. Érvényesíti a számítógép felépítésével kapcsolatos elveit, majd dolgozatot közöl a folyamatról (First Draft of a Report on the EDVAC), melynek kapcsán az eredeti ötletgazdákkal (Mauchly és Eckert) és több más mérnökkel is szerzői jogi vitákba keveredik.

A berendezés két változata (N-EDVAC és M-EDVAC) szintén az Egyesült Államok hadseregének finanszírozásával épül meg, és egy évtizeden át használják főként ballisztikus lövedékek pályaadatainak kiszámítására.

Az N-EDVAC gép konstrukciójában egy nagyon érdekes újítás jelenik meg, az úgynevezett higanyalapú, akusztikus memória (2.18. ábra, tinyurl.com/yc8brjh4). Ez a ma használatos számítógépek háttértárolóinak és dinamikus (állandó tartalomfrissítést igénylő) memóriáinak elődje, soros be- és kiírású, de a lyukkártyákhoz és lyukszalagokhoz képest tetszőlegesen változó adatok tárolására képes. Adatátviteli sebessége (48–384 mikroszekundum/szó) sokszorosa az elektromechanikus elődökének. Működési elve szerint a higanyoszlopban lassan terjedő

hanghullámokba kódolják az adatokat, majd egy, a hullámok intenzitását érzékelő eszköz (kvarckristályos interfész) segítségével hívják elő azokat. Az N-EDVAC memóriaegysége 1024 szavas, ahol egy szó 44 bites bontásban szerepel.

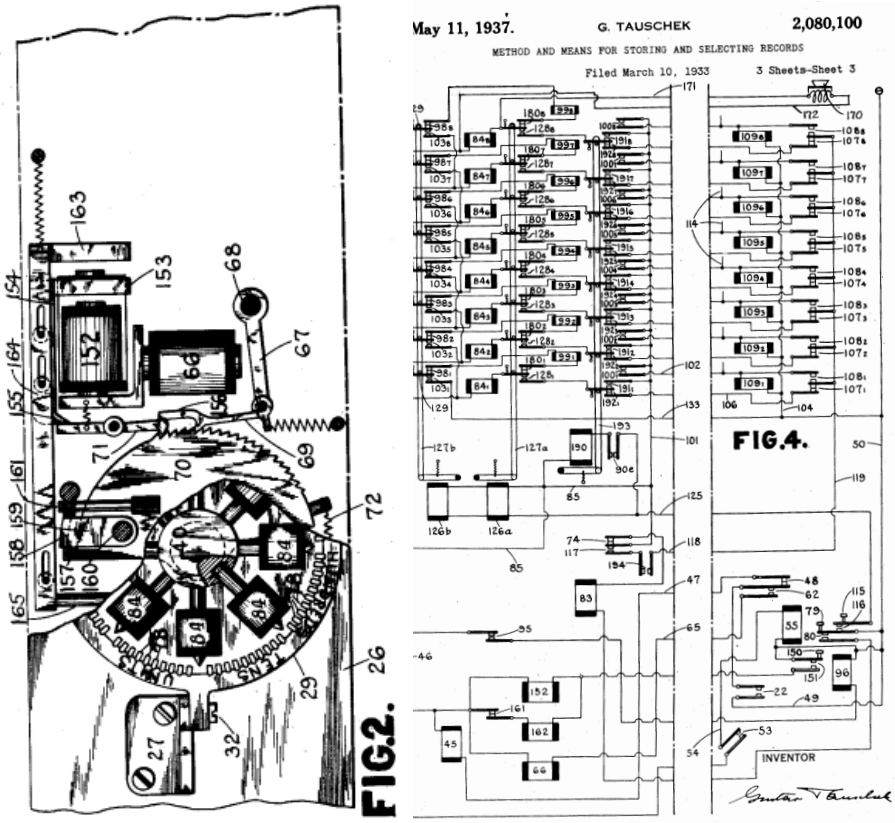


2.18. ábra. *Az akusztikus memória működési elve*

Az M-EDVAC-ot az akusztikus memória mellett mágneshengeres háttértárolóval (magnetic drum) is felszerelik. Ez a berendezés a mai merevlemezes háttértárolók elődje. Feltalálója az osztrák Gustav Tauschek, aki már 1932-ben kis kondenzátorok sokaságát fűzi fel kapcsolópontok közé egy henger belsejében, amelyeket kapcsoló armatúrák érintenek meg, és igény szerint töltenek fel vagy sütnek ki, ilyen módon tárolva a bináris információt. Később az egyes kapcsolópontokon megjelenő feszültségek mérésével rekonstruálható az eltárolt információ (szabadalmaztatva 1937-ben az Egyesült Államokban). Ezt a módszert helyettesíti a későbbiekben (1944) a ferromágneses jelenségre alapozott adattárolás és visszaolvasás, ahol a ferromágneses réteggel bevont hengerfelület lokális mágnesezését apró tekercsekkel állítják be, illetve olvassák vissza (2.19. ábra, tinyurl.com/2dysn6zx).

- First Draft of a Report on the EDVAC
- ENIAC, EDVAC

Korabeli fényképek és leírások (1946) tanúsága szerint a számítógéptermekben katódsugárcsöves oszcilloszkópok is felbukkannak, ezeket a jelformák ellenőrzéséhez és a regisztertartalmaknak elektromos jelalak formában történő megjelenítéséhez használják, mintegy előrevetítve ezzel a pár évvel később megjelenő katódsugárcsöves kijelzőket is, mint az ember-számítógép interfészek első változatát.



2.19. ábra. *Mágneshengeres háttértároló ábrája a szabadalmi dokumentációban*

Az eddig olvasottakat áttekintve megállapíthatjuk, hogy a ma használatos számítógépek összes alapvető részegysége az 1950-es évek elejére a tudósok és tervezőmérnökök rendelkezésére állt. A fejlődés útjai kijelöltettek. A számítástechnika forradalma elkezdődött, aktív meghatározójává és formálójává vált napjaink történéseinek. Kezdeti hatalmas kiterjedése és energiaigénye a félvezető alapú integrált áramkörök megjelenésével fokozatosan csökkent, mígnem napjainkra sokak számára elérhetővé és hordozhatóvá vált. Ezeket a változatokat manapság mindenhova magunkkal hurcoljuk és okostelefonnak hívjuk. Ezen eszközök számítási sebessége és memóriakapacitása több tízezerezerszerese az 50-60 évvel ezelőtti berendezéseknek, míg energiaigényük az ősökének tízezredrészét sem éri el.

3. A modern számítástechnika kialakulása

3.1. A számítógép fogalma és generációi

A számítógép egy elektronikus berendezés, amely képes előre meghatározott módon, programokat végrehajtva adatokat feldolgozni úgy, hogy a program végrehajtása során nem igényel közvetlen emberi beavatkozást. Elviekben emberi beavatkozás a program és a szükséges adatok memóriába töltéséhez, valamint a program kezdőcímének beállításához szükséges. A mai fogalmaink szerinti számítógép innen már önjáró módon, a program utasításainak meghatározott sorrendben történő végrehajtásával végzi feladatát.

A számítógépeket kezdetben egyetemi központok és nagyvállalatok tervezték és építették, többnyire a kormányok és a hadsereg részére, ezeket nem lehetett kereskedelmi forgalomban megvásárolni. Az első, kereskedelmi forgalomba hozott elektronikus számítógép az UNIVAC I volt. 1954-ben kezdték el forgalmazni az Egyesült Államokban, és az ezt követő két évtizedben közel 50 000 darabot adtak el belőle.

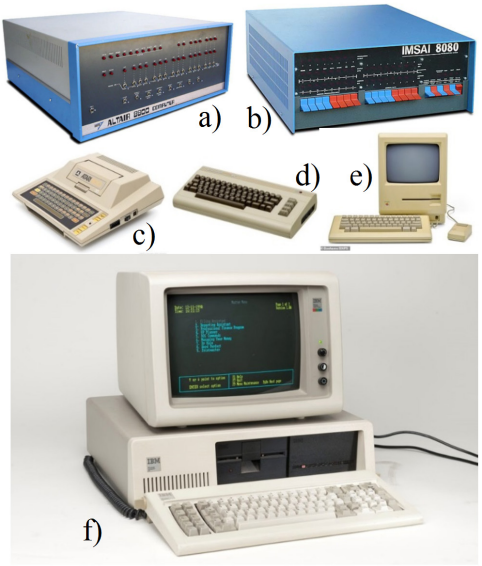
A számítógépeket a felépítésüket alapvetően meghatározó áramköri elemek és a működési elv alapján úgynevezett generációkba soroljuk. Az első generáció (1946–1959) áramköreit elektroncsövek és elektromágneses jelfogók (relék) alkotják. Ezekből alakítják ki a logikai kapukat és kapcsolják őket nagyobb egységekbe. Jellemzőjük a nagy méret (többszobányi) és a gyakori meghibásodások, műveleti sebességük körülbelül 1000 művelet másodpercenként. Az adatrögzítés papíralapú lyukkártyákkal vagy perforált papírszalaggal történik.

A második generáció (1959–1965) kapuáramköreit már tranzisztorok alkotják. Elődeiknél megbízhatóbbak voltak és nagyobb tárolókapacitással rendelkeztek. Méretük lecsökkent (több szekrénynyi), és 10 000 művelet végrehajtására voltak képesek másodpercenként. Az adatok rögzítése részben mágneses elven történt.

A harmadik generációs számítógépek (1964–1975) már integrált áramköröket (IC) használnak. Méretük még jobban lecsökkent, műveleti sebességük hozzávetőleg 500 000 összeadó művelet másodpercenként. Ezek a számítógépek már univerzális felhasználásra készültek, üzleti és műszaki-tudományos célú feladatokat egyaránt el tudtak látni. Adattárolásra mágneses elvű háttértárolót használnak. Ezeket a gépeket egy időben több felhasználó (munkaállomás) is igénybe vehette.

A negyedik generációs gépek (1975-től) fő építőeleme a mikroprocesszor és a kiegészítő funkciókat ellátó digitális integrált áramkörök. Az integrált áramkörök gyártástechnológiája (LSI – Large Scale Integration) a méret további csökkenését eredményezi. A műveleti sebességük 10 000 000 összeadó művelet másodpercenként. A kisebb vállalatok számára is elérhetővé válik az addig csak kutatóközpontok számára elérhető számítási hatékonyság.

A negyedik generációs számítógépek (3.1. ábra) kiemelt helyet foglalnak el a technikatörténetben, mivel ezek terjedtek el a legjobban. Ennek egyik alapvető oka az, hogy sikerrel integrálták és generálták azokat a technikai újításokat (LSI, majd VLSI – Very Large Scale Integration technológia, párhuzamos végrehajtási csövek – pipeline stb.), amelyek üzleti szempontokból is a legkifizetődőbbnek bizonyultak. Megjelennek a mikroszámítógépek.



3.1. ábra. *Negyedik generációs mikroszámítógépek: a) Altair, b) IMSAI, c) Atari, d) Commodore, e) Apple, f) IBM PC*

Az ötödik generációs számítógép terveit 1980-ban dolgozták ki Japánban. A negyedik generáció akkori számítástechnikai lehetőségeire építve a kidolgozási koncepciójuk, felépítési elveik – párhuzamos végrehajtás, hardver- és szoftverszerkezetük – logikai programozás, különböznek a közhasználatban lévő számítógépekétől. Ez a generáció a 80-as, 90-es években üzleti szempontból kevésbé volt sikeres, mint a negyedik generáció, túlságosan drága volt a megvalósításuk és üzemeltetésük. A negyedik generációs gépek processzorai idővel több esetben is túlteljesítik őket az alacsonyabb komplexitású műveletsorok elvégzésében. Idővel több műszaki megoldásukat beépítik az egyre komplexebbé váló mikroprocesszorokba, és ezekből a mikroprocesszorokból alakítják ki a szuperszámítógépek processzortömbjeit, a felhőalapú számítástechnikai rendszerek processzorklasztereit, szerverparkok számítástechnikai rendszereit, videoprocesszorokat.

3.2. Számítástechnikai mértékegységek

A számítógépek felépítésükből következően az adatokat kettes számrendszerben felírható értékként tárolják és kezelik. A szövegek, a képek, a hanganyag mind bináris számsorozatok formájában tárolódik és közlekedik a belső digitális áramkörök között. Ennek érdekében a betáplálandó adatokat előbb digitalizálni kell, vagyis egy erre kialakított eszköz bináris számok sorozatává alakítja a szöveget, a fényképeket, a hang- és mozgóképanyagokat, felvételeket. Ez a folyamat a digitalizálás. Ennek eredménye a felhasználó számára csak úgy válik értelmezhetővé, ha szükség szerint visszaalakítja ezeket látható vagy hallható jelzésekké. A digitális elnevezés a latin eredetiből – digitus – angol közvetítéssel digit (szám) került a magyar informatikai és számítástechnikai kifejezéstárba. A latin szó jelentése: ujj, átvitt értelemben a felmutatott ujj szimbólumára utal, mint információhordozó jelzésre.

Az adatok tárolásához szükséges memóriamodulok fő jellemzője a tárolókapacitásuk mértéke, ezért fontos, hogy ezt a mennyiségét mérni tudjuk. Az adat mennyiségének mérésére szolgáló legkisebb egység a bit, amelynek fogalmát több forrás szerint 1948-ban a **bi**nary digi**t** (kettes számrendszerbeli szám) angol szavakból hozták létre. A bit két értéke az 1-es és a 0-ás, logikai értelemben az IGAZ vagy HIGH (magas) és a HAMIS vagy LOW (alacsony) válaszminőségeket jelöli.

A bit nyolcszorosából alakult ki a bájt (byte, rövidítése B). Ezt a kifejezést képezzük más mértékegységekhez hasonló előszavakkal a többszörös memóriakapacitások leírására:

1 B = 8 bit
1 kB (kilo) = 1024 B
1 MB (mega) = 1024 KB
1 GB (giga) = 1024 MB
1 TB (tera) = 1024 GB
1 PB (peta) = 1024 TB
1 EB (exa) = 1024 PB
1 ZB (zetta) = 1024 EB
1 YB (yotta) = 1024 ZB

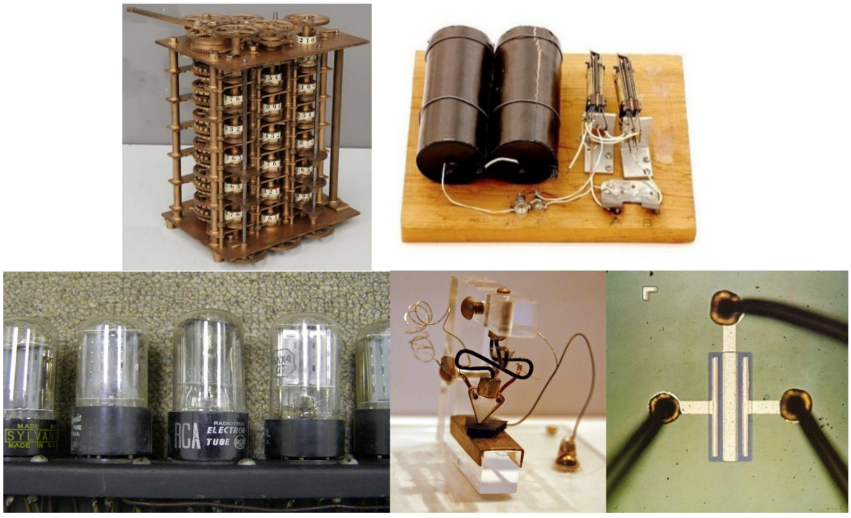
Figyeljünk arra, hogy a számítástechnikában a váltószám nem 1000, hanem 1024, mivel a kettes számrendszer helyi értékei 2 hatványai szerint alakulnak, és a kettő tizedik (kilo – ezer) hatványa 1024.

- computer generations
- memory units table

3.3. A számítástechnikai rendszerek integrált áramkörei

Az eddigi technikatörténeti visszatekintés során azt láthattuk, hogy fejlődési sorrendben a számoló-, majd később számítógépek kezdetben mechanikus alkatrészeket, fogaskerekeket, hajtórudakat (3.2. ábra 1. képe) alkalmaznak a számok fizikai leképezéséhez, a műveleteket a fogaskerék-áttételek kódolják, a számok ábrázolására a 10-es vagy 12-es számrendszert alkalmazzák (az ókortól az 1800-as évekig). Ezután térnek át a jelfogós (relé – Joseph Henry, 1835) megvalósításokra (3.2. ábra 2. képe) és a 2-es számrendszer alkalmazására (már széles körben ismert Shannon munkássága). Majd következik az elektroncsövek megjelenése (vezérelhető elektron vagy ionáramlás – John Ambrose Fleming, 1904) és kapcsolóelemként való alkalmazása a számítástechnikában (3.2. ábra 3. képe). Ezekkel az alkatrészekkel még nem lehet hosszú távon megbízhatóan működő számítástechnikai rendszereket építeni.

1947-ben az Egyesült Államokban működő Bell Laboratories kutatóinak: John Bardeennek, Walter Brattainnak és William Shockley-nek sikerül elkészíteni az első működőképes bipoláris tranzisztort (3.2. ábra 4. képe). A gyártástechnológiai nehézségek leküzdése után egy rövid ideig úgy tűnik, hogy ez az elektronikai alkatrész lesz a jövő számítógépeinek építőköve. De a Bell Laboratories munkatársai, John Atalla és Dawon Kahng újabb találmánnyal lepik meg a világot, 1959 novemberében elkészül az első MOSFET (Metal Oxide Semiconductor Field Effect Transistor), a térvezérlésű tranzisztor (3.2. ábra 5. képe), és miután kidolgozzák a gyártástechnológiai részleteket, kiderül, hogy technikai és gazdasági szempontból is kifizetődőbb lesz MOSFET-tranzisztorokból kialakítani a digitális integrált áramköröket.

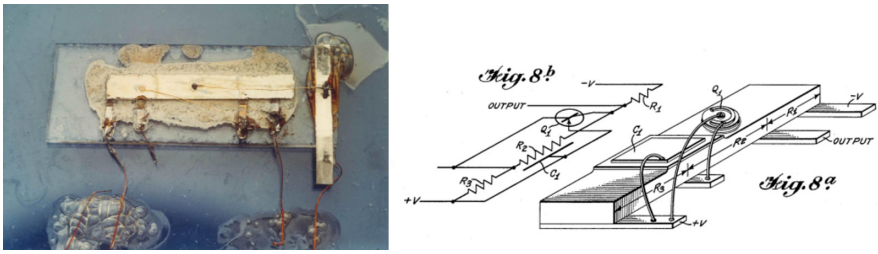


3.2. ábra. *A számítógépek fő alkatrészei megjelenési sorrendben: fogaskerekek, relék, elektroncsövek, bipoláris tranzisztorok, MOSFET-tranzisztorok*

A számítástechnikai rendszerek az alkatrészek méretének csökkenésével egyre kisebbek lesznek és villamosenergia-igényük egyre csökken. Az integrált áramkörök első, germániumlapocskákra kialakított, működő változatát 1958 őszén, az amerikai Texas Instruments vállalathoz frissen alkalmazott Jack Kilby mutatja be a vezetőségnek (3.3. ábra, tinyurl.com/mphuytsf), megnyitva ezzel az utat a félvezető technológiára alapozott elektronikus áramkörök gyártása előtt.

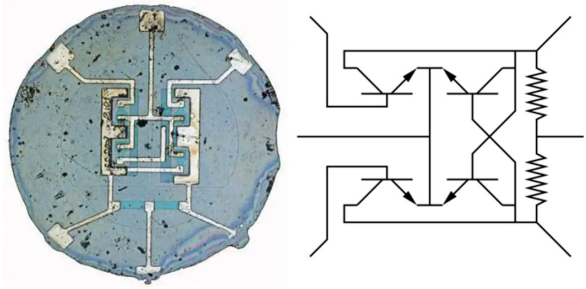
Az integrált áramkör kifejezés arra vonatkozik, hogy többféle alap elektronikai komponens hozható létre ugyanazon a félvezető felületen: dióda, tranzisztor, ellenállás, kondenzátor és tekercs.

Robert Noice ettől függetlenül, ugyanebben az időben a Fairchild amerikai vállalat alapítójaként és fejlesztőmérnökeként kísérletezik, szabadalmaztat integrált áramkört 1959-ben (teljes egészében szilíciumlapkára építve). Csapatával elsőként mutat be működőképes, monolitikus, logikai integrált áramkört, egy RS-cellát (3.4. ábra, tinyurl.com/yzz38svs).



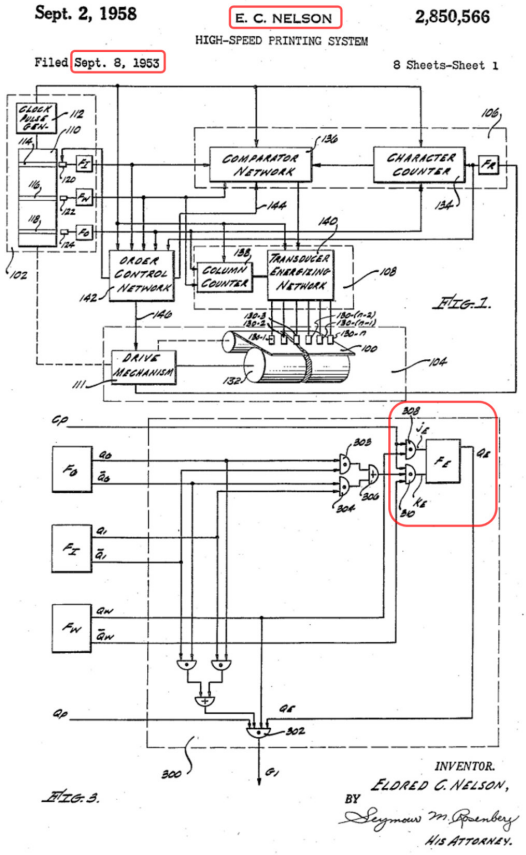
3.3. ábra. *Jack Kilby integrált áramköre 1958-ból, egytranzisztoros RC-oszcillátor*

A Texas Instruments és Fairchild vállalatok hosszas pereskedést folytatnak a találmányok szerzői jogainak elsőbbségéért, szerencsére 1967-ben sikerül kiegyezniük. E két vállalat munkatársai közül kerülnek ki a későbbi processzorgyártó vállalatok alapítói is. Ebben az időszakban az Egyesült Államok világelső a félvezető technológiák és integrált áramkörök fejlesztésében.



3.4. ábra. *Robert Noice logikai integrált áramköre 1961-ből, négytranzisztoros, monolitikus „F” flip-flop, RS-cella*

A témában utolsó érdekesség a JK flip-flop elnevezés eredete. Ez egy szakaszos vagy szekvenciális, tárolócellás logikai áramkör (későbbi fejezetben – 6.5.10 – lásd a részletes bemutatását). Egyes források Jack Kilby nevéhez kötik a bevezetését, valószínűleg nevének kezdőbetűire gondolva. A valóság azonban az, hogy az áramkör bemeneteit hagyományosan jelölő J és K betűket Eldred Nelson használja először flip-flop cellák bemeneteinek jelölésére. Nelson a Hughes Aircraft amerikai vállalatnál tevékenykedik, és 1953-ban szabadalmaztat egy, a számítástechnikához köthető találmányt, a termikus transzfert alkalmazó nyomtatót. Ennek vezérlő áramkörét számos logikai kapuból, illetve szekvenciális logikai áramkörökből, flip-flopokból tervezi meg (3.5. ábra, tinyurl.com/4ptv7fet). Érdekes belepillantani egy korabeli szabadalmi dokumentumba, amelyen piros keretben kiemelve látható a J és K jelölés. A szabadalom jó 5 évvel előzi meg Kilby első szabadalmának dátumát.



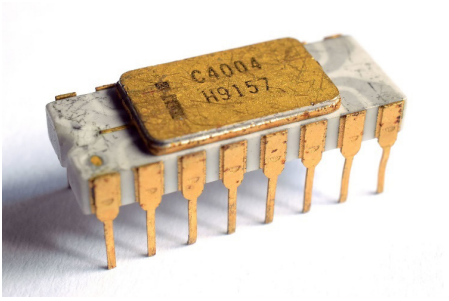
3.5. ábra. *Eldred Nelson szabadalmi dokumentumának egyik oldala a J és K jelölésekkel a flip-flop áramkör bemenetén*

- the first transistor
- the first integrated circuit

3.4. Az Intel processzorai

Az Intel szakembergárdájának egy része a Fairchild integráltáramkör-gyártó cég volt alapítói közül került ki, emlékezetes, legendás alakok Robert Noyce és Gordon Moore, az Intel alapítói. Később csatlakozik hozzájuk „Ted" Marcian Hoff, Frederico Faggin is, csak néhányat említve a kiemelkedő személyek sorából.

Az új cég az integráltáramkör-gyártás technológiájának birtokában digitális áramkörök és hamarosan processzorok fejlesztésébe, majd gyártásába kezd. Első termékük az 1971-ben megjelent 4004-es jelzésű, 4 bites processzor (3.6. ábra, tinyurl.com/2zvdjwa2), amelynek architektúráját Ted Hoff dolgozta ki, gyártástechnológiai részleteit Frederico Faggin tervezte meg. Faggin még a Fairchildnél kidolgozta az önmegvezetéses kapumaszkolás (self aligned gate-semiconductor technology) eljárást a MOSFET-tranzisztorok integrált áramköri kialakítására, ezt az elvet alkalmazza az Intel első processzorainak gyártásánál is (a módszer lényege a kapu – gate – elektródaszigetelő fészkének elsődleges kialakítása, ami után a további rétegszennyezési – diffúziós bevitel – lépések számára természetes maszkot képez a már kialakított kapufészek).



3.6. ábra. *Az Intel első, 4 bites processzora kerámiatokban, a szilíciumlapkát védő rézötvözet fedővel*

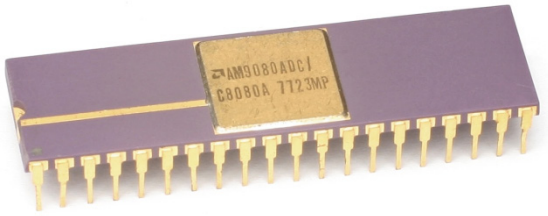
A cég sikere alapítása óta töretlen, a világ egyik legnagyobb és legelismertebb processzorfejlesztője és -gyártója. Legfőbb riválisai az AMD, TSMC, Samsung (Az AMD később felvásárolja a TSMC-t a processzorgyártó kapacitás bővítése és az új technológiák alkalmazásában elért eredményei okán). Intel gyártmányú a legendás processzorsorozat kezdve a 8080-as lapkától a 286, 386, 486, 586, majd a Pentium-lapkákon át az i3, i5, i7, i9 többmagos lapkákig.

3.7. ábra. *Az Intel processzor lapkáinak felülnézeti képei*

A 3.7. ábra (tinyurl.com/2zt6danw) az Intel internetes oldalán látható és a gyártó processzorlapkáit ábrázolja. Az egyes képeken megfigyelhető színes négyszögek a processzorok egyes funkcionálisan elkülönülő részegységeit jelzik. A megvilágítás miatt az egységek felülről látható vezetékezése rajzolja ki ezeket a sajátos mintázatokat, például a nagyobb, összefüggőnek látszó felületek belsőmemória-táblákra vagy regisztermezőre utalnak.

3.5. Az AMD processzorai

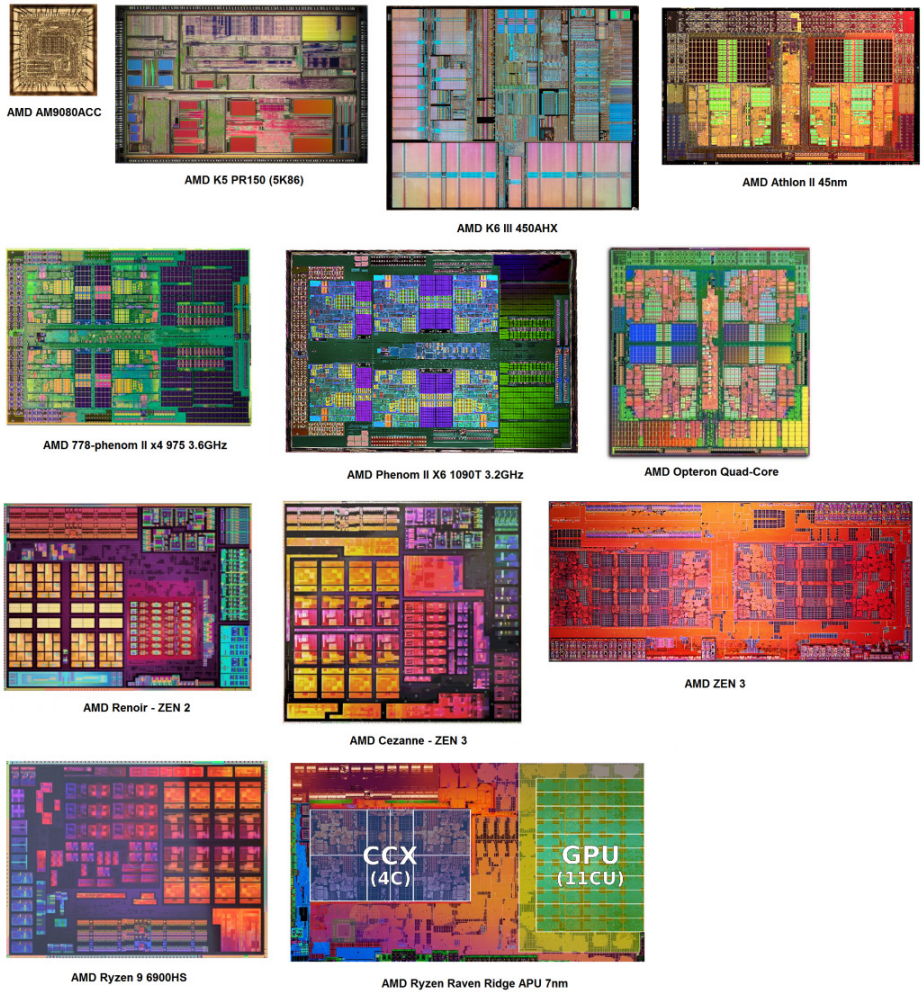
Az AMD (Advanced Micro Devices) története az Intellel közös tőből fakad, ugyanis a cég szakembergárdájának egy része az 1969-es alapításkor szintén a Fairchild integráltáramkör-gyártó cég volt mérnökei közül került ki. A hét másik alapító tag W. Jerry Sanderset (Walter Jeremiah) kérte fel a csapat vezetésére, így ő lett a vállalat vezérigazgatója. Sanders villamosmérnök, de inkább az eladási területen mozog otthonosan. Az Intel-alapító Robert Noice-hoz baráti szálak fűzik, ez az AMD későbbi történetében nagy jelentőséggel bír majd.



3.8. ábra. *Az AMD első 8 bites processzora kerámiatokban, a szilíciumlapkát védő rézötvözet fedővel*

Az integrált áramkörök gyártástechnológiájának terén szerzett tapasztalatokat a gyorsan bővülő AMD-s mérnökcsapat a korabeli, szabályozatlan szerzői jogok viszonyát kihasználva más cégek, a Fairchild és az Intel integrált áramköreinek visszafejtéséből vagy még emlékekből történő rekonstrukciója alapján tervezett logikai integrált áramkörök gyártásában hasznosítja. Ezeket Sanders zseniális üzleti érzékkel katonai besorolású paraméterekre tervezteti, így hamar piacot szerez a termékeknek. Az AMD Am9080 nevű, első, 8 bites mikroprocesszorát (3.8. ábra, tinyurl.com/epca3tvz) 1974-től kezdve gyártották. Ez az áramkör az Intel 8080 processzor klónja, amelynek visszafejtését Shawn Hailey és Kim Hailey egy korai Intel processzorlapka fotóiból kiindulva kezdték el, és sikeresen kifejlesztették a teljes kapcsolást és logikai diagramot. A másolat kissé továbbfejlesztett elektromos paraméterekkel és kisebb lapkamérettel készültek, és ebből fakadóan a gyártás során elérték a 100 lapka/ostya (nyers szilíciumkorong) mennyiséget, ami meghaladta az Intel gyártási technológiájában alkalmazott csoportosítást.

Az AMD az Intelhez hasonlóan számos gyártástechnológiai újítást fejlesztett ki, gyárakat hozott létre, vállalatokat vásárolt fel világszerte, számítógép-architektúrákat tervezett és honosított meg (Athlon XP, Athlon 64, Duron, Opteron, Turion, Phenom, FX, Ryzen).

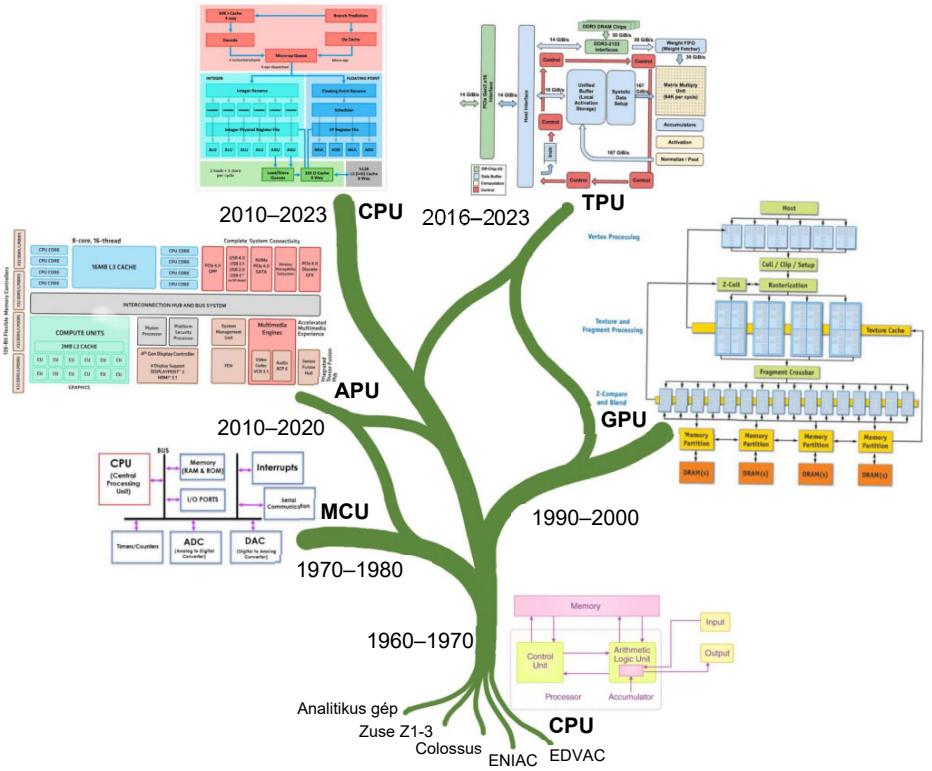


3.9. ábra. *Az AMD-processzor lapkáinak felülnézeti képei*

- Intel processors
- AMD processors

3.6. Mikroprocesszorok, mikrovezérlők, GPU-k, APU-k, TPU-k

A számítástechnikai eszközök családfáján (3.10. ábra) a mikroprocesszor (CPU) alapú főágból kiválik a mikrovezérlő (MCU) alapú, beágyazott rendszereket kiszolgáló számítástechnikai rendszerek oldalága. A főágból egy további hajtás ágazik le, a grafikus processzoroké (GPU), amelyek a számítógépes képgenerálás nélkülözhetetlen eszközei.

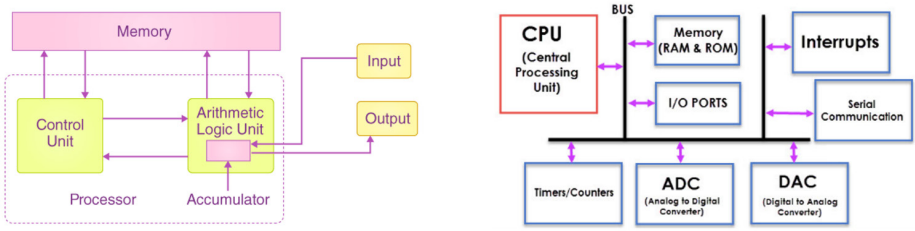


3.10. ábra. *A CPU, MCU, GPU, APU, TPU családfa*

Az ágak rövidebb ideig párhuzamosan fejlődnek, majd egy ponton, mintegy a három ág egyesüléseként egy negyedik, az előbbi hárommal jelenleg szintén párhuzamosan fejlődő ágat alkotva, létrejönnek a mai hordozható számítástechnikai rendszerekbe (mobiltelefonokba, táblagépekbe) beépített komplex mikrokörnyezetes processzorok vagy végrehajtó egységek (APU – Accelerated Processing Unit). Ez utóbbiak egy vagy több szilíciumlapkán kialakított CPU +

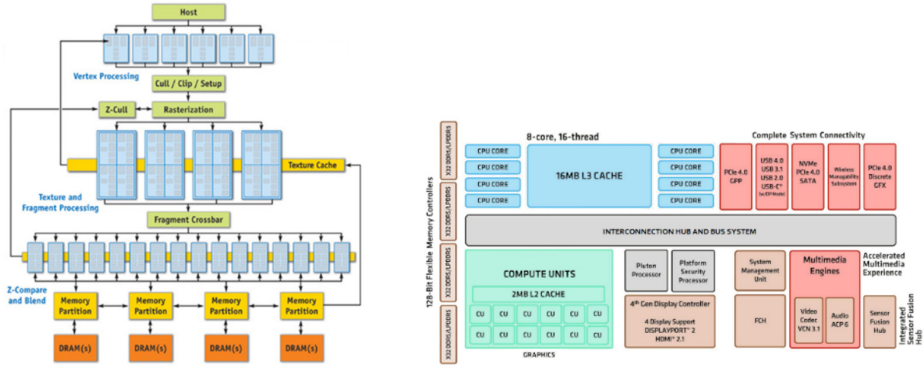
GPU (Graphical Processing Unit) + alkalmazás specifikus perifériaillesztő áramkörök együtteséből állnak. Az ötödik, legfiatalabb ágként a 2016-ban bevezetett tenzor processzorokat említhetjük (TPU – Tensor Processing Unit), amelyek a mesterséges intelligenciás alkalmazások neuronhálóinak számítási kapacitását biztosítják.

A leágazások a különböző számítási igényeknek és felhasználási területeknek megfelelően a technológia fejlődésével azonos ütemben történtek. A kezdeti egymagos processzorarchitektúra soros végrehajtási lánccal rendelkezik, a külvilág felé egy cím- és adatbusszal, valamint pár vezérlővonallal kapcsolódik (mag – az utasítások végrehajtásához szükséges aritmetikai-logikai-, vezérlőegység és munkaregiszterek). Működtetéséhez külső memória és egyéb áramkörök szükségesek (3.11. ábra. 1, tinyurl.com/y65frcck).



3.11. ábra. *Az egymagos mikroprocesszor és a mikrovezérlő architektúra*

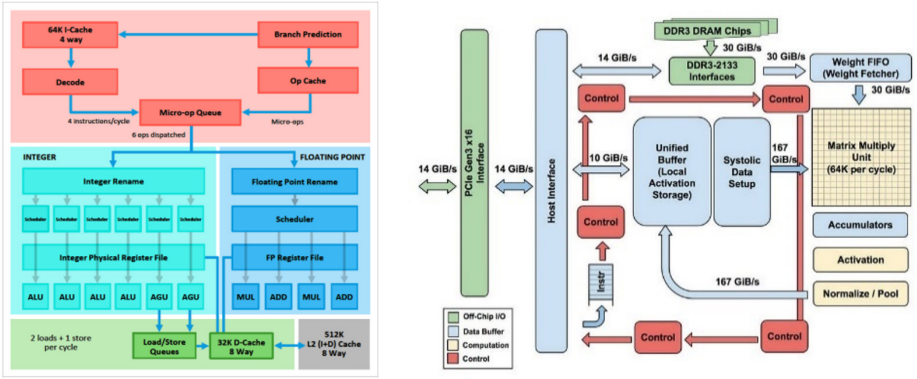
A mikrovezérlők a processzormagot egy „mikrokörnyezetbe” integrálják, ahol a cím- és adatsín sok esetben az áramkörön belül marad. A külvilág felé számos funkciót ellátó ki- és bemeneti csatornával rendelkező mikrovezérlő beépített program és adatmemóriával rendelkezik (3.11. ábra. 2, tinyurl.com/mr4tzute).



3.12. ábra. *A sokmagos grafikus processzor és a többmagos processzort és grafikus processzort integráló, alkalmazás specifikus egység*

A grafikus processzorok a digitális képek minél élethűbb megjelenítéséhez szükséges adatfeldolgozáshoz nagyon sok (64, 128 vagy még több) egyszerű processzáló magot használnak fel. Ezeket a magokat komplex adatsínrendszerek, memóriavezérlők és feladatszétosztó vezérlőegységek szolgálják ki, illetve kapcsolják a külső, dedikált memóriabankok felé (3.12. ábra. 1, tinyurl.com/mtdn6hft).

A hordozható számítástechnikai eszközök (táblagépek, mobiltelefonok) egy alkalmazásspecifikus processzálóegységet használnak. Ez egy többmagos processzor és sokmagos grafikus processzor, valamint speciális kommunikációs és multimédiás egységek egymás mellé építésével létrehozott komplex számítástechnikai egység, amely erősen optimizált energiafelhasználási mutatókkal rendelkezik az akkumulátoros energiaellátás üzemidejének maximalizálása érdekében (3.12. ábra. 2).



3.13. ábra. *Sokmagos tenzor processzor MI-rendszerekhez és napjaink többmagos processzora asztali és hordozható számítógépekhez*

Ezekkel párhuzamosan a személyi számítógépekbe beépített processzorok is tovább fejlődtek, többmagos, sok GHz-es órajel-frekvencián működő számítástechnikai „erőművekké" váltak, amelyek a csatolt rendszermemóriákat akár 10Gb/s átviteli sebességen is képesek használni (3.13. ábra. 1, tinyurl.com/27kxj3ca). Ez az átviteli sebesség teszi lehetővé a virtuális valóság és a támogatott valóság (augmented reality) élvezhető képi megjelenítését.

A mesterségesintelligencia-alkalmazások támogatására az utóbbi években kifejlesztett, új architektúra a tenzor processzor. Akár 64x64-es mátrixszerűen összekapcsolt, egyszerű aritmetikai-logikai egységből álló processzormagjainak számító kapacitása több milliárd egyszerű (összeadás, szorzás) műveletet is elérhet másodpercenként viszonylag kis, akár csak 700 MHz-es órajel-frekvencia mellett

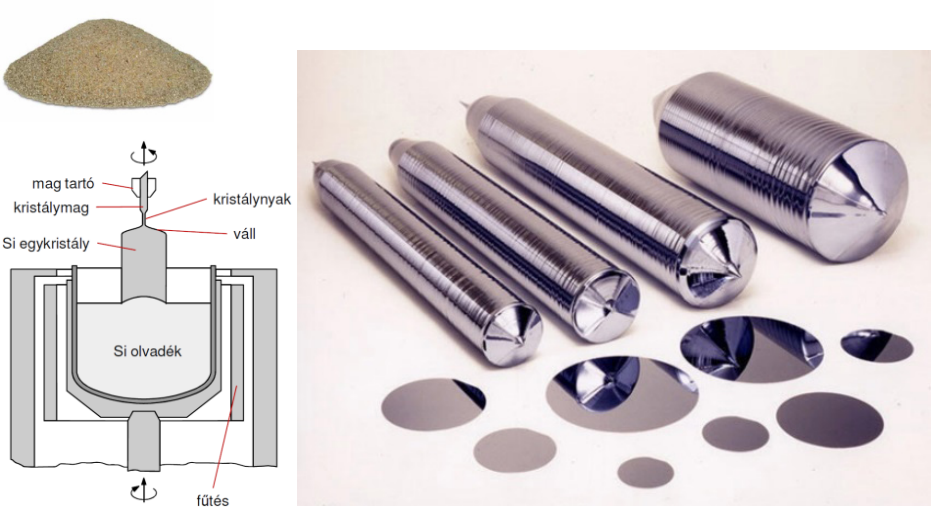
(wave processing – hullámfeldolgozás). Így hatalmas adatmennyiség átfutását teszi lehetővé, a mesterséges neuronhálók adatfeldolgozási igényét kielégítve (3.13. ábra. 2, tinyurl.com/yvp4yw5u).

- microprocessor vs microcontroller
- GPU, APU, TPU

4. Félvezetők működése és előállítási technológiájuk

4.1. Alapanyagok, a félvezetők előállítása

A természetben előforduló szilícium (vegyértéke 4) a félvezető technológia legfontosabb alapanyaga. Kvarchomok formában való kitermelés (pl. Ausztrália partjai), majd tisztítás után elektromos kemencében, magas hőmérsékleten megolvasztják.



4.1. ábra. *Kvarchomok, olvasztókemence, szilíciummonokristály-rudak*

A szilíciumolvadékból az úgynevezett kristálymag felhasználásával, ezt lassan forgatva létrehozzák a nagyon nagy tisztaságú (99,999%) szilíciummonokristály-rudat. Ezt egy következő lépésben gyémántbevonatú vágókorongokkal felszeletelik, a korongokat rendkívül simára csiszolják, majd ezekre a korongokra (4.1. ábra, tinyurl.com/3h93a4ts) alakítják ki az integrált áramkörök struktúráit.

A szilícium nem vezeti az elektromos áramot, a kristályrácsban a külső elektronhéjának minden elektronja stabil kötésben vesz részt a szomszédos atomok hasonló elektronjaival. Ezért „szennyezik”, azaz bizonyos jól meghatározott helyeken a kristályszerkezet atomjai közé „idegen” atomokat juttatnak be (magas hőmérsékleten, nagy nyomás alatt kezelik a szilíciumkorong felületét gáz formájában

bejuttatott szennyező anyagokkal, pl. arzénnal vagy bórral, közben maszkolással takarják el a szennyezni nem kívánt részeket).

Azokon a részeken, ahova **arzént** juttatnak be, amely külső elektronhéján 5 elektronnal rendelkezik, elektrontöbblet keletkezik, mivel az arzén négy elektronja kötésben vesz részt, az ötödik viszont elszakad (az arzén elektront ad, vagyis donor). Ezt nevezzük **N** típusú félvezetőnek.

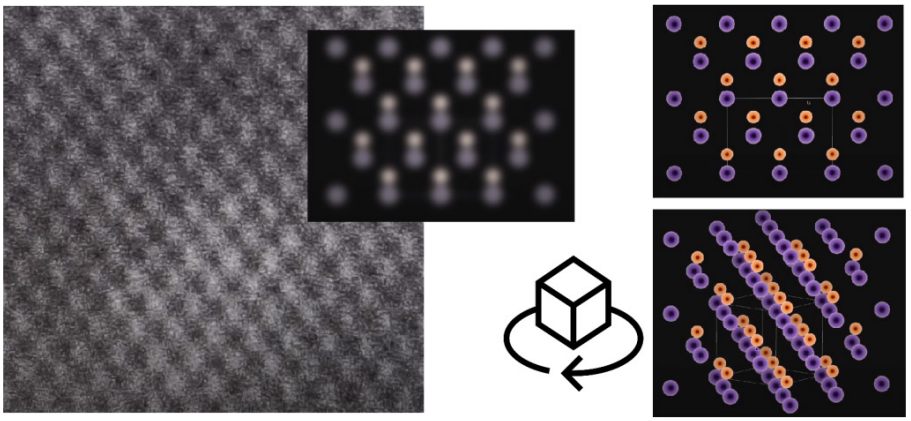
4.2. ábra. *A félvezetők típusai: az N és a P félvezető alapréteg*

Azokon a részeken, ahova **bórt** juttatnak be, amely külső elektronhéján 3 elektronnal rendelkezik, elektronhiány keletkezik („lyuk”), mivel a bór három elektronja kötésben vesz részt, a negyedik kötés itt nem jön létre (a bór a kötésben szabad elektront tudna felvenni, vagyis akceptor). Ezt nevezzük **P** típusú félvezetőnek.

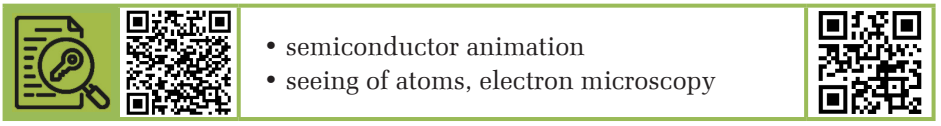
A szennyező anyagokat és azok mennyiségét kombinálva mikroszkopikus méretű (mikrométeres és nanométeres tartományban) N és P típusú rétegeket hoznak létre a szilíciumkorong felületének jól meghatározott részein (4.2. ábra, tinyurl.com/5fr44wnc).

A mai, modern logikai áramkörök alapját mikroszkopikus méretű tervezérlésű tranzisztorok (FET – Field Effect Tranzisztor – transzfer rezisztor) képezik, amelyeket az előzőekben leírt, egymás közelében kialakított P és N típusú félvezető rétegek kombinálásával, majd ezek összekötésével (alumínium- vagy rézvezetékek segítségével) alakítanak ki.

Napjaink pásztázó elektronmikroszkópjainak (STEM – Scanning Transmission Electron Microscopy) és beépített képfeldolgozó szoftvereinek segítségével a kristályrács finomszerkezete láthatóvá válik, a szabályosan sorjázó világos foltok az atomok. Az atomok finomszerkezete nem látható, a világos foltokat olyan léptékben képzeljük el az elektronokhoz képest, mint a kisebb vagy közepes csillagokat a bolygórendszerükben több fényévnyi távolságból megfigyelve (4.3. ábra). A módszer alapja az elektronsugaras mintaátvilágítás alapján kialakuló „árnykép” rögzítése és digitális feldolgozása.



4.3. ábra. *Szennyezett szilícium elektronmikroszkópos képe (a kristályrács elméleti modelljének elforgatásával kapnánk hasonló nézetű képet)*

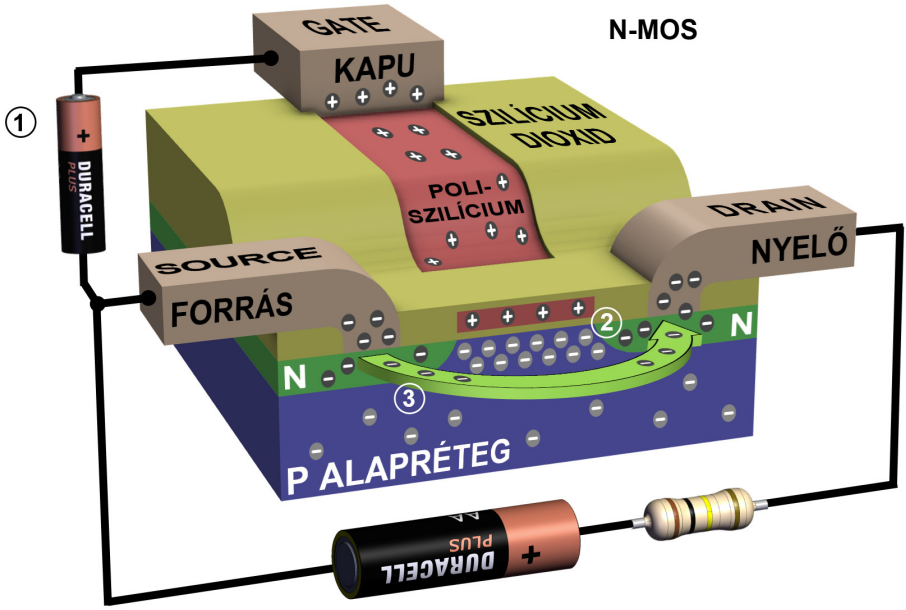


4.2. Térvezérlésű tranzisztorok (FET) működése

A térvezérlésű tranzisztorok felépítésében a tranzisztor két azonos, erősen szennyezett (P^+ vagy N^+) rétegből álló pólusát, a forrást (source) és nyelőt (drain), egy gyenge (P vagy N), ellentétesen szennyezett alapréteg (bulk vagy body) szigeteli el egymástól. Így alakulhat ki a P^+ N P^+ vagy N^+ P N^+ rétegződés.

A FET működtetéséhez (a kapcsoláshoz) a két pólus között lévő alapréteg elektronjait vagy elektron hiányait (lyukakat) a közelükben létrehozott elektromos mezővel úgy kell befolyásolni, hogy a két azonosan szennyezett réteg között az elektronáramlás megindduljon – kialakuljon egy úgynevezett vezetőcsatorna. Ez úgy érhető el, hogy a középső réteg közvetlen közelében, de ettől elszigetelve (pl. SiO_2 – szilícium-dioxid rétegbe ágyazva) egy vezérlőelektródát (gate – kapu – anyaga erősen P^+ vagy N^+ szennyezett, polikristályos szilícium – vezető anyag) alakítanak ki.

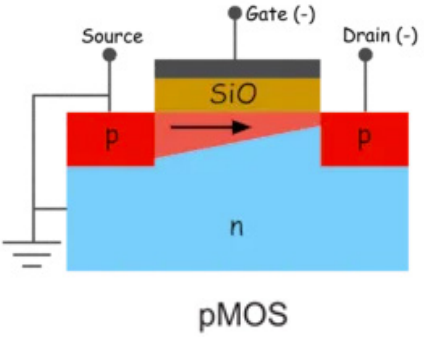
Az alábbi képen látható NPN (N-FET vagy N-MOSFET – metal oxid FET) rétegezés esetében a P alapréteg fölött lévő vezérlőelektródát (gate) a forrás (source) elektródához képest pozitívan kell polarizálni (4.4. ábra, 1). A kialakuló pozitív elektromos tér elektronokat vonz magához a nagyon közeli P alaprétegből (4.4. ábra, 2).



4.4. ábra. *N-FET működését ábrázoló modell a forrás, az elnyelő és a vezérlőelektróda térbeli elhelyezkedésének bemutatásával*

Ez az elektronáramlás a forrásra (source) kapcsolt feszültség bizonyos küszöbértéke fölött a P rétegben még több „lyukat” eredményez, amelyek helyére a szomszédos N^+ réteg, a forrás (source) szabad elektronjai „benyomulhatnak”. A hatás kiterjed a közelben lévő másik N^+ réteg irányába is, így létrejön az elektronáramlás a két N^+ réteg között (4.4. ábra, 3). A térvezérlésű tranzisztor BEKAPCSOLT – VEZET, a source és drain pólusai közötti ellenállás ($R_{DS}ON$) értéke lecsökken (több megaohm nagyságrendről pár Ohm vagy milliohm nagyságrendűre).

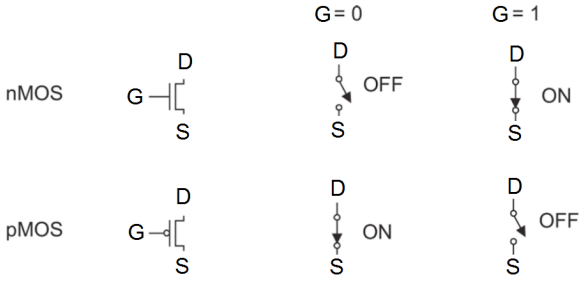
A PNP (P-FET vagy P-MOSFET) rétegezés esetében a folyamat hasonló, csak ellentétes polaritással, itt az N alapréteg fölött lévő elszigetelt N^- vezérlőelektródát (gate) a forrás (source) elektródához képest negatívan kell polarizálni. A kialakuló negatív elektromos tér az elektronokat taszítva „lyukakat” hoz létre a nagyon közeli N alaprétegben (az úgynevezett küszöbfeszültség fölött a negatív elektromos tér hatására a szilíciumatomok közötti kovalens kötések bomlani kezdenek), aminek hatására beindul az elektronáramlás (a „lyukakba” benyomuló elektronok) a forrás (source) és nyelő (drain) elektródák között (4.5. ábra, tinyurl.com/yrjemn22).



4.5. ábra. *P-FET működését sematikusan ábrázoló modell a vezetőcsatorna kialakulásának feltüntetésével*

Az előbbiekben leírt kialakítású és működésű térvezérlésű tranzisztorokat kapcsolóként használva, megkapjuk azt a rendkívüli mértékben integrálható (kicsinyíthető és egymás mellé sűríthető), kombinálható és feszültséggel vezérelhető eszközt, amelynek állapotai (NYITOTT – VEZETŐ vagy ZÁRT – SZIGETELŐ) a számítógépek áramköreinek működési alapját képező két LOGIKAI értéket:

– a megfelelő feszültségszint meglétéhez rendelt logikai 1-es (H – high – magas) állapotot,
– a feszültségszint földpotenciál közeli értékéhez rendelt logikai 0-ás (L – low – alacsony) állapotot létrehozzák.



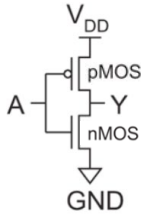
4.6. ábra. *A FET-ekkel létrehozható kapcsolások, ha a vezérlőelektróda küszöbfeszültségét logikai állapotoknak feleltetjük meg*

A 4.6. ábra (tinyurl.com/mf3s2w7u) szerint a kétféle (N és P) térvezérlésű tranzisztor vezérlése és kapcsolása egymással ellentétes logikai képletet eredményez. Az N típusú esetében a kapuelektródára kapcsolt pozitív (1-es) állapotnak megfelelő feszültség kapcsolja be az eszközt, míg a P típusú esetében a kapuelektródára kapcsolt negatív (0-ás) állapotnak megfelelő feszültség eredményezi a

bekapcsolást. Ebből következően a térvezérlésű tranzisztorokat egymáshoz kapcsolva különböző **elemi logikai áramköröket**, úgynevezett **logikai kapukat** lehet létrehozni (ezek típusait a következő fejezetekben részletesen tárgyaljuk). Ezekből a logikai kapukból épülnek fel a számítógépek bonyolult logikai áramkörei, amelyek az utasítások végrehajtásáért, az adatok tárolásáért és mozgatásáért felelnek.

4.3. Integrált áramkörök fizikai kialakításának megtervezése

A tranzisztorokat és az azokból kombinált logikai áramköröket alkotó különböző P és N rétegeket maszkolási eljárással alakítják ki, azokat a területeket, amelyeket nem akarnak változtatni, fényérzékeny festék- vagy oxidréteggel vonják be, így a szennyezés vagy maratás csak a kifedett területeket éri el. Az egyik legegyszerűbb és legalapvetőbb logikai áramkör az úgynevezett tagadó kapu (inverting gate), amelyet két komplementer (egy P és egy N típusú) térvezérlésű tranzisztorból képeznek A-val jelölt bemenettel és Y-nal jelölt kimenettel (4.7. ábra).



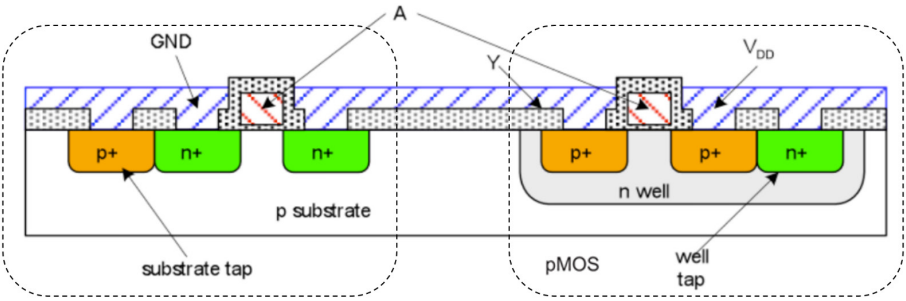
4.7. ábra. *Elementáris, komplementer FET-ekből kialakított logikai, tagadó kapu*

A 4.6. ábra (tinyurl.com/yu4ey5fk) kapcsolási táblázatából kiindulva levezethető a kapu működése:

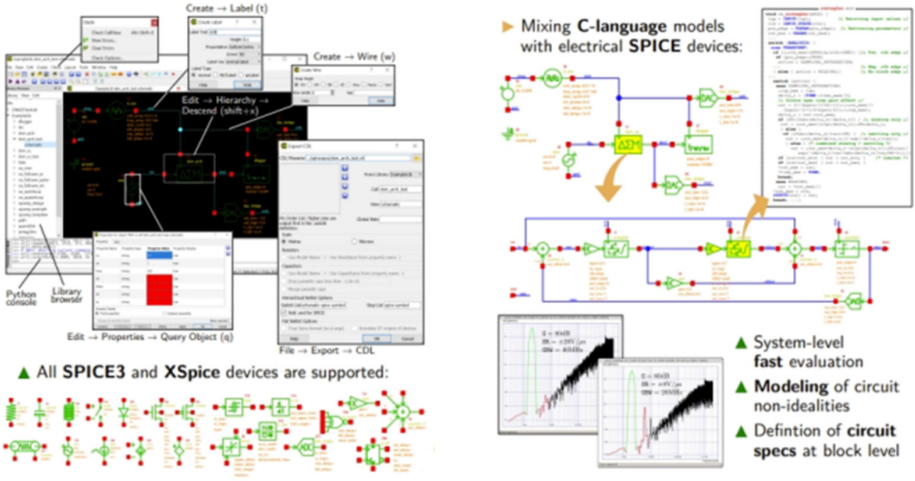
– amikor a bemenet A=”1”=VDD, akkor a PMOS kikapcsolt állapotba kerül, az NMOS viszont bekapcsolt állapotban van (mert G=1), így a kimenet Y=”0”=GND,
– amikor a bemenet A=”0”=GND, akkor a PMOS bekapcsolt állapotba kerül, az NMOS viszont kikapcsolt állapotban van (mert G=0), így a kimenet Y=”1”=VDD.

A 4.8. ábra keresztmetszetben ábrázolja a tagadó kapu félvezető rétegeit. Jól megfigyelhető a már bemutatott P és N váltakozó rétegződés és az a tény, hogy az ábra jobb oldalán a P alaprétegbe egy lokális N alaprétegeket (n well – n forrás) hoznak létre, mintegy ebbe „fészkelve” a PMOS-tranzisztort, elszigetelve a bal oldali NMOS párjától. Az ábra bal szélén egy P^+, a jobb szélén egy N^+ csatolóréteg látható (substrate tap és well tap), ezeken keresztül kapcsolódik az alapréteg bal

szélen a GND földpotenciálra, jobb szélen a VDD tápfeszültségre. A következőkben tekintsük át, hogyan jön létre ez a struktúra.



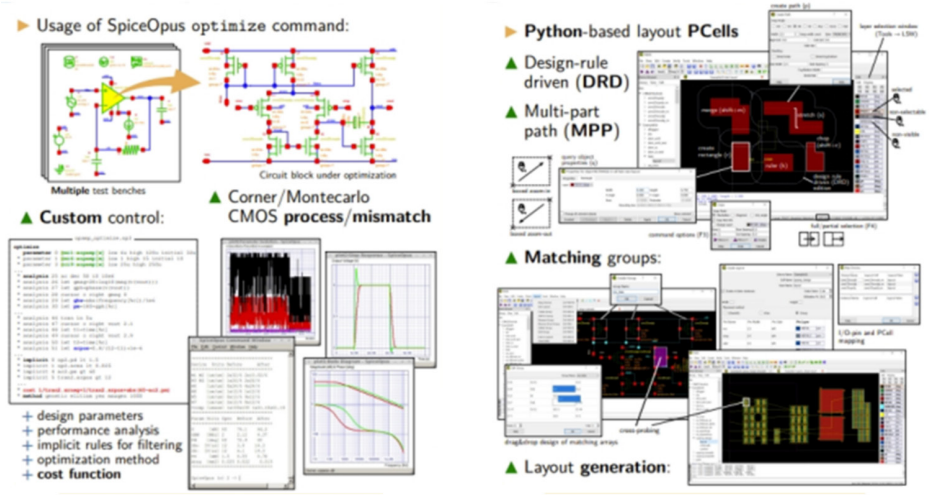
4.8. ábra. *A tagadó kapu félvezető rétegeinek keresztmetszeti kialakítása*



4.9. ábra. *Szimbolikus elemekkel való tervezés és elektromos szimuláció*

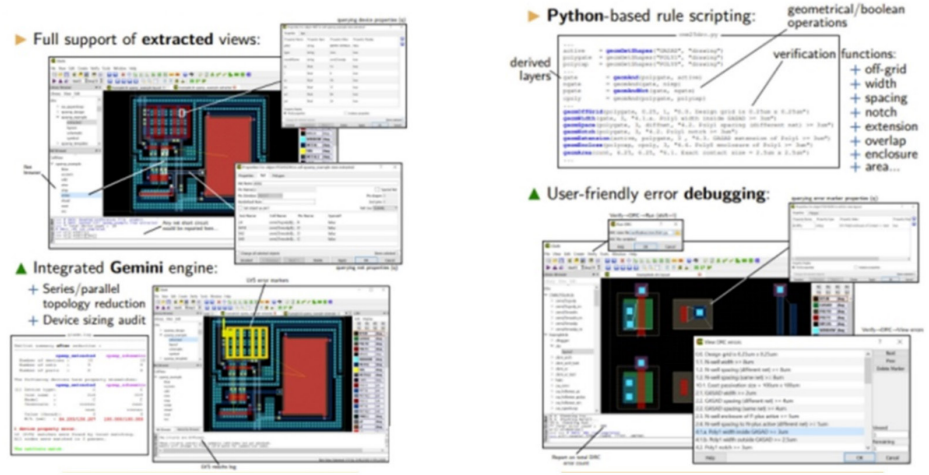
A digitális integrált áramkörök tervezése a megvalósítani kívánt integrált áramkör elektromos kapcsolási rajzának rögzítésével kezdődik, az alkatrészeket szimbolikus formában bevezetik a program elektromos kapcsolásirajz-szerkesztő részébe (4.9. ábra. 1).

A kapcsolási rajzból származtatott elektromos kapcsolati modellt leszimulálják egy program (például X-Spice) segítségével, amely képes az áramköri részek matematikai modelljei alapján a kapcsolásban betöltött szerepüket kiértékelni (4.9. ábra. 2).



4.10. ábra. *Optimalizáció és fizikai kialakítás megtervezése*

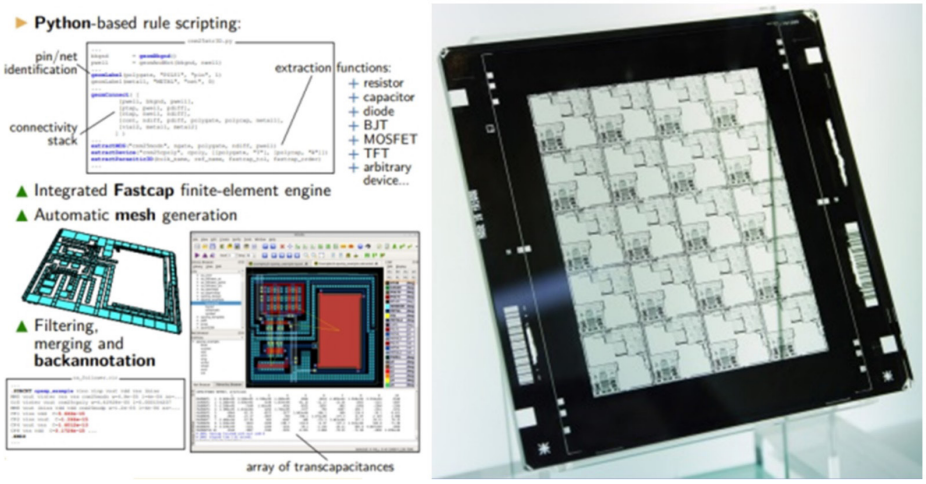
A kapcsolási rajzot ezután optimalizálják a minél kevesebb átmenet és zóna kialakításának érdekében (4.10. ábra. 1). Így a fizikai megvalósítás megbízhatóbb és követhetőbb lesz. Az optimalizált kapcsolási terv alapján megtervezik a szilíciumlapra integrálandó fizikai elemeket (4.10. ábra. 2). Ezek a strukturális elemek alkotják a félvezetőket, átfedéseik, érintkezési pontjaik érzékenyen befolyásolják a kialakuló áramkörök elektromos tulajdonságait.



4.11. ábra. *Automatikus fizikai struktúra méretezése és ellenőrzése*

A megtervezett struktúrákat a gyártástechnológiai paramétereknek megfelelően automatikusan méretezik, majd ellenőrzik a tartható és elvárt értékeket (méretek, távolságok, szögek) (4.11. ábra. 1). Aztán összevetik a kapcsolási rajzot a megvalósított struktúrákkal (4.11. ábra. 2) szimbolikus és leíró nyelvi formában is.

A programcsomag automatikusan kigenerálja a háromdimenziós struktúrákat, beleértve a félvezető rétegeket, zónákat és az összekötő vezetékeket. Ennek alapján megtörténhet a háromdimenziós szerkezet feltérképezése, és a felépítésből kialakuló parazita alkatrészek (kapacitások és induktivitások, akár RC-áramkörök) kiszűrését elvégezve (4.12. ábra. 1) végül kigenerálják a maszk állományokat (4.12. ábra. 2), amelyeket a fotolitográfiás folyamatban felhasználnak.

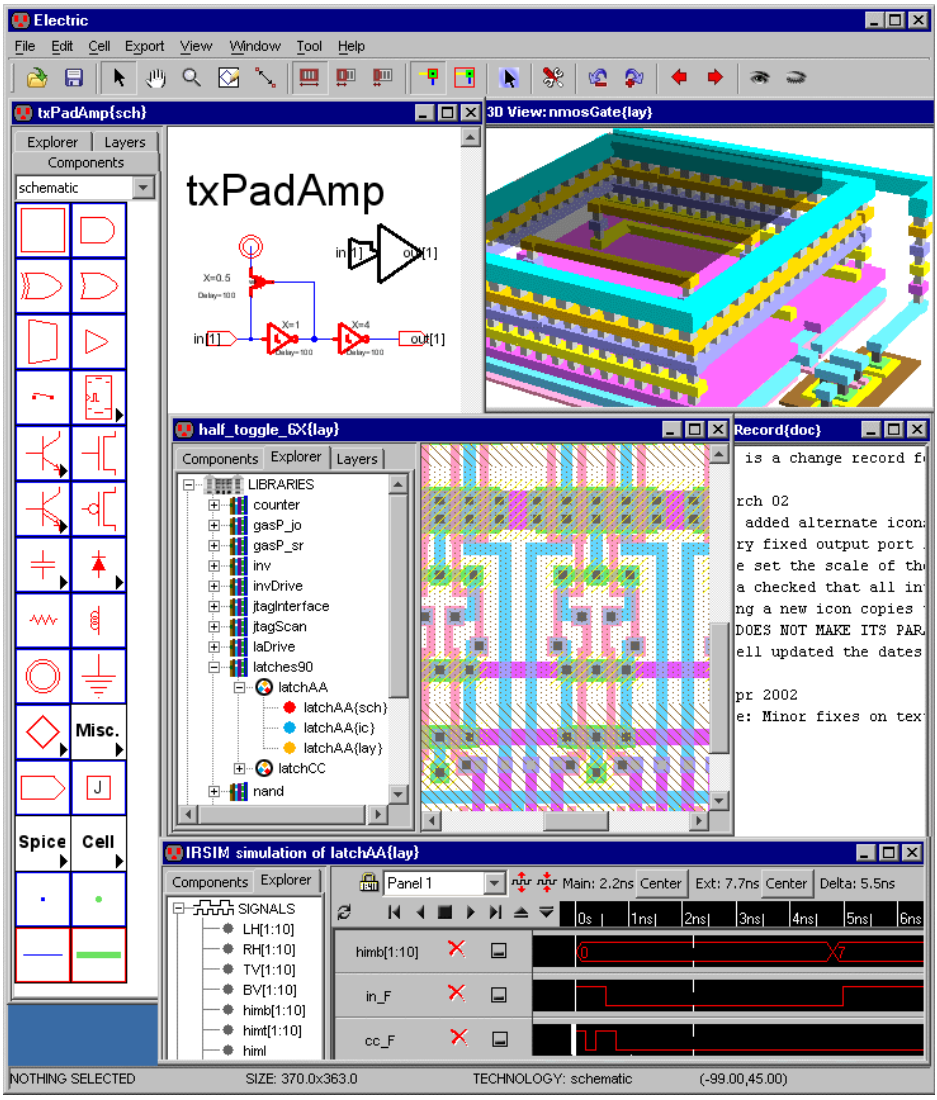


4.12. ábra. *Automatikus 3D-struktúragenerálás és parazita alkatrészek szűrése. Fotolitográfiás maszkok létrehozása*

A VLSI-áramkörök tervezését segítő programban (EDA – Electronic Design Automation) síkban (felülnézetben) és térben is láthatjuk a megtervezett áramkört (4.13. ábra, tinyurl.com/3bzdcpyc). A VLSI (Very Large Scale Integration) egy átfogó technológiát jelöl, amelynek alapját a félvezető lapkák fizikai kialakításának paraméterei és gyártási módszereinek meghatározása képezi. Ez a technológia tette lehetővé a ma forgalomban lévő, sok száz millió tranzisztorból álló processzorok létrehozását.

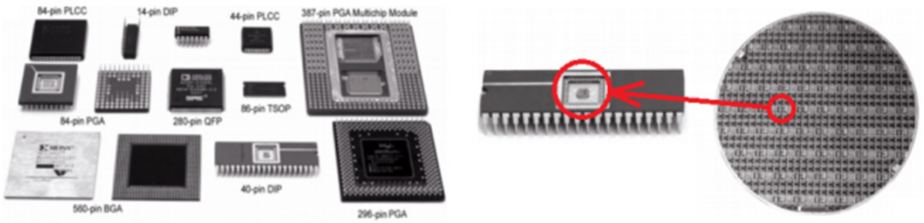
A tervezést segíti az áramkör funkcióit programírás-szerűen leíró nyelv (pl. VHDL – Very High-Speed Integrated Circuit Hardware Description Language) és szimulációs környezet is. Ez a program magába integrálja az előzőekben bemutatott tervezési szakaszokat, sok előre meghatározott struktúrát tartalmaz

(pl. alaprétegeket, a MOSFET-tranzisztorok „alkatrészeit”, összekötő vezetékeket stb.), és képes létrehozni a fotolitográfiás folyamathoz szükséges állományokat is.



4.13. ábra. *Az ingyenesen elérhető EDA-program lehetőséget nyújt a VLSI-tervezésbe való betekintésre*

Az áramkörök mérete, a szilíciumlapkák formája, a tokozás kialakítása attól függ, hogy milyen feladat megvalósítására szánják az illető áramkört. A tervezést és gyártást magába foglaló folyamat végén létrejönnek a különböző integrált áramkörök (4.14. ábra, tinyurl.com/ysyyhcxy), amelyeket egy későbbi, tervezési, kivitelezési és összeszerelési folyamat során elektromos berendezések nyomtatott áramköreire szerelnek.



4.14. ábra. *Megszülettek a különböző tokozású integrált áramkörök*

- MOSFET simulator
- integrated circuit design

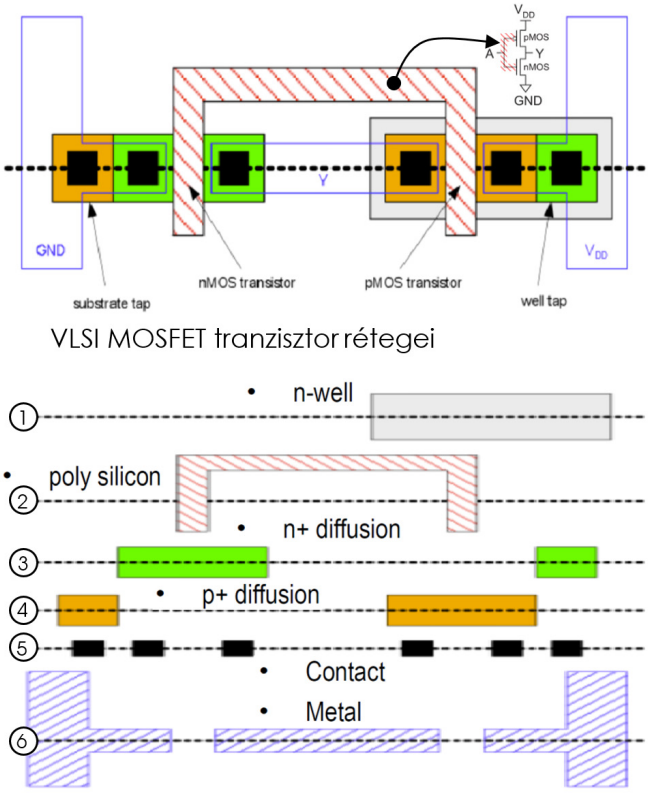
4.4. Integrált áramkörök gyártástechnológiája

A megtervezett integrált áramkörök egy hosszú és bonyolult folyamat során jönnek létre, alapanyagként használva a 4.1. alfejezetben bemutatott szilícium monokristály rúdból kivágott korongokat. Ezekre a korongokra képezik ultraprecíziós, számítógép-vezérlésű szerszámgépek, kémiai reaktortartályok és anyagmozgató rendszerek (a gyártás különböző fázisai közötti anyagmozgatás, szilíciumkorongok szállítása) segítségével az integrált áramköröket. A teljes folyamatnak a leírását nevezzük gyártástechnológiának.

A 4.15. ábra (tinyurl.com/4rz58n7y) felülnézetben mutatja a tagadó kapu kialakítását, „alaprajzát”, ahogy az az előállítás során egy mikroszkopikus méretű térbeli struktúrává, „épületté” válik. A piros dőlt vonallal kiemelt rész a poliszilícium (vezető) rétegből kialakított, egymással összekötött kapuelektródákat jelöli. Ez a sziget képezi az áramkör **A** bemenetét (az ábrán kicsiben láthatjuk a tagadó kapu szimbolikus, áramköri, kapcsolási rajzát, és rajta megjelölve a poliszilíciumréteg szerepét).

A kék körvonallal jelölt részek a fémkivezetésekre utalnak, ezeken a vezetékeken keresztül kapcsolódnak össze az egy szilíciumlapkára létrehozott különböző áramkörök, majd továbbhaladnak a tokozás kivezetéseihez (ezeket a félvezető

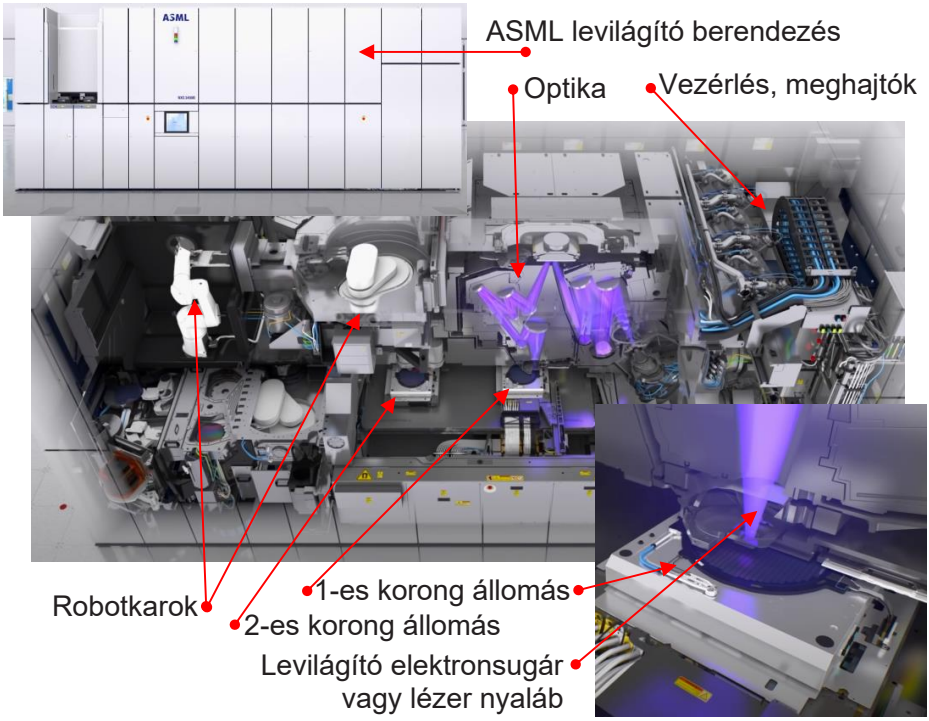
lapkán kialakított kontaktpontokat az integráltáramkör-gyártás későbbi fázisában, a tokozási szakaszban kötik össze a tok fém kivezetéseivel).



4.15. ábra. *Tagadó kapu fizikai kialakításának felülnézeti terve. Alatta a kialakításhoz szükséges maszkok láthatók*

A tervezőprogramban kialakított struktúrák részleteit a program nyomtatható képekké – **maszkokká** – alakítja. Ezután kezdődhet egy soklépéses technológiafolyamat, amelynek során a szilíciumkorong felületét különböző eljárásokkal kezelik egyre újabb és újabb szennyező vagy vezető réteget kialakítva.

Ahhoz, hogy a terv alapján alakíthassák ki a rétegeket, „szigeteket” és szinteket, és ezek méretei és anyagtulajdonságai (pl. a szennyezés mértéke) az elvártnak megfelelőek legyenek, maszkolási technikával rendre elfedik a szilícium hordozólemez felületét, amíg az egyes szennyezési (diffúzió), oxidálási vagy elektrolízises, permetezős (rétegképzés), esetleg hőkezelési fázis lezajlik.

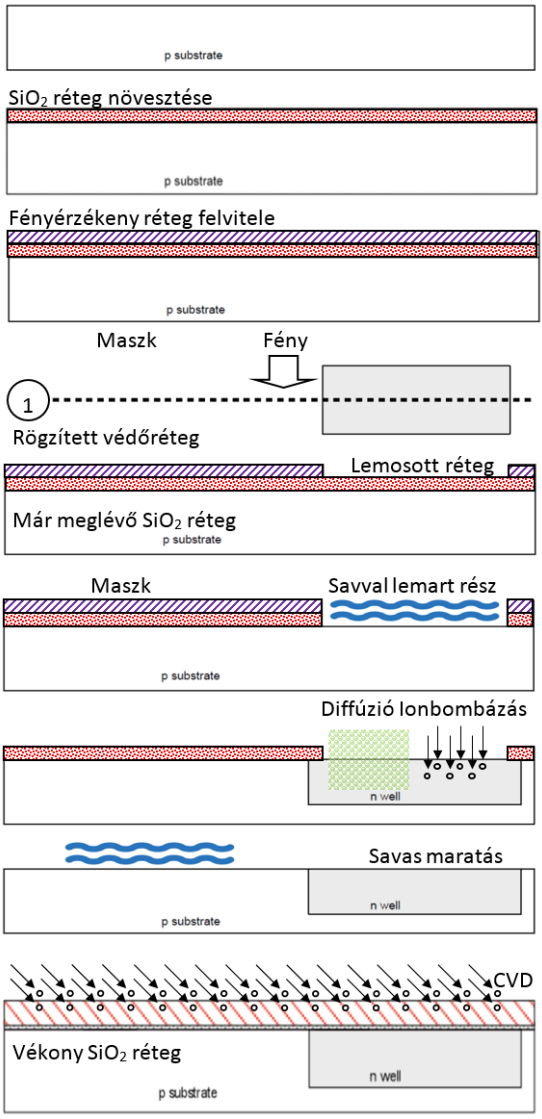


4.16. ábra. *Fotolitográfiás, maszklevilágító berendezés főbb egységei*

A lézermegmunkált maszkokat fény vagy elektronnyaláb útjába állítva átvilágítják, és a képet egy lencserendszerrel lekicsinyítik és rávetítik a szilíciumlap felületére (4.16. ábra, tinyurl.com/3hkse2vx), ahol az előzőleg felvitt fényérzékeny anyagréteg a fény vagy elektronnyaláb hatására megkeményedik, a kitakart részeken viszont lemosható marad. A védőrétegek kémiailag ellenállnak a savak hatásának.

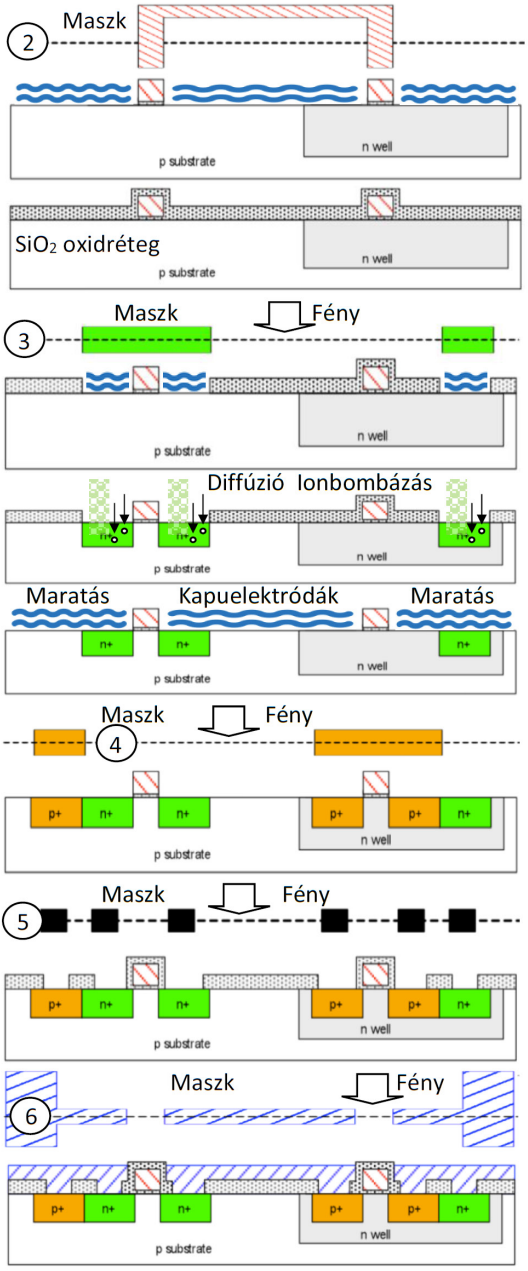
A példában szereplő tagadó kapu kialakításához a leegyszerűsítve magyarázott technológiai folyamatban az előzőleg P szennyezéssel kezelt szilíciumlemez-felületre 6 maszkréteg (4.17. ábra) segítségével fotolitográfiás vagy elektronnyalábbal végzett aktiválás módszerrel viszik fel a megfelelő védőrétegeket. A védőrétegek által nem takart részeket savas maratással távolítják el.

– Először a szilíciumkorong egy meghatározott felületét szilícium-dioxid (SiO_2) réteggel vonják be. Ez egy szigetelőréteget képez a továbbiakban.
– Ezután az oxid védőrétegre áztatással felkerül egy fényérzékeny réteg.



4.17. ábra. *A fotolitográfiás eljárás lépései, az első (1) maszk levilágítása, majd az ezt követő felületkezelési lépések*

– Megtörténik az **1**-es maszk levilágítása és a fényérzékeny réteg exponálása (rögzítése vagy megszilárdítása), majd eltávolítása lemosással, ott, ahol a fény nem érte a felületet.

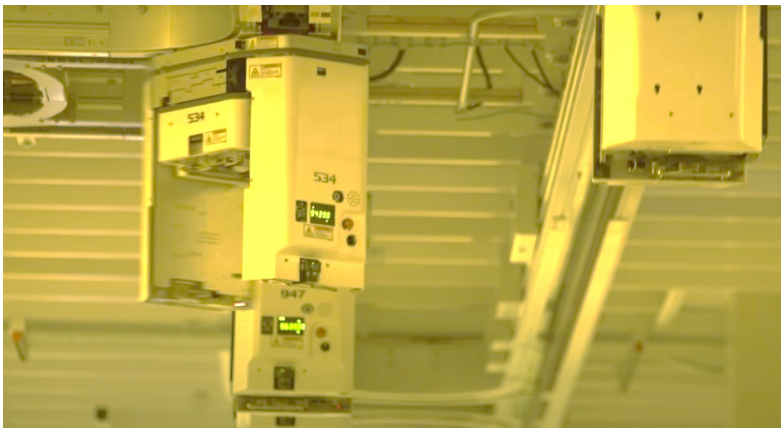


4.18. ábra. *A fotolitográfiás eljárás lépései: a 2…6-os maszkok levilágítása, majd az ezeket követő felületkezelési lépések*

– Ahol a felületen nincs védőréteg (exponált mező), ott a savas maratás során leoldódik a szilícium-dioxid réteg, alatta felbukkan a P szennyezett szilíciumkorong felülete.
– Következik ebbe a P rétegbe ágyazott N alapréteg kialakítása ott, ahol az oxidréteg nem takarja a szilícium alapréteget. Ez a művelet speciális kamrában, magas hőmérsékleten végzett diffúziós eljárással (pl. arzéngázos atmoszférában), vagy alacsonyabb hőmérsékleten végzett ionbombázással történik. Itt a szennyező atomok behatolnak a védtelen szilíciumrétegbe és beépülnek a kristályrács atomjai közé. Technikai érdekesség, hogy ezekben a reaktorkamráknak is nevezett tartályokban a hőmérséklet- és nyomásértékeket százezred pontossággal szabályozzák és ellenőrzik (pl. 698.43671°C).
– A szennyezéses rétegkialakítás után a teljes oxidvédőréteg eltávolítása következik savas maratással.
– A maratási lépések után a felületet vizes öblítéssel lemossák.
– Újabb, ezúttal vékony (6-7 atomnyi vastagságban) oxidréteg felvitele következik. Ez lesz a szigetelőréteg a kapu (gate) elektróda és a már létrehozott P és N alaprétegek között.
– A vékony oxidréteg után CVD (Chemical Vapor Deposition) eljárással, hevítő kemencében szilán (SiH_4) gáz adagolásával a vékony oxidréteg tetején létrehozzák az erősen szennyezett poliszilícium kristályréteget (a szennyezéssel erősen csökken az ellenállása, jó vezetővé válik). Ebből lesz a kapuelektróda (gate) rétege.
– A továbbiakban megtörténik a **2**-es maszk levilágítása (4.18. ábra 1) és a fényérzékeny réteg exponálása (rögzítése vagy megszilárdítása), majd eltávolítása lemosással ott, ahol a fény nem érte a felületet.
– Újabb oxidvédőréteg kerül a felületre, amely befedi az előzőleg kialakított poliszilícium-struktúrát.
– A következő rétegek szennyezési eljárásához a poliszilícium-struktúra pontos maszkként fogja meghatározni a szennyezendő felületek méretét (self aligning gate technology – Frederico Faggin és Robert Noice alkalmazza elsőként processzorgyártásban). A vezetővé szennyezett poliszilícium a magas hőmérsékletű kezelések során a fémelektródákkal ellentétben nem olvad meg.
– Megtörténik a **3**-as maszk levilágítása és a fényérzékeny réteg exponálása, majd eltávolítása lemosással ott, ahol a fény nem érte a felületet.
– Ezután a kifedett felületeken az oxidréteget savas maratással eltávolítják.
– Következik a P és N alaprétegekbe ágyazott N^+ rétegek kialakítása. Ez gázatmoszférában diffúziós módszerrel vagy ionbombázással történik.
– A szennyezéses rétegkialakítás után az oxidvédőréteg eltávolítása következik savas maratással.
– A maratási lépések után a felületet vizes öblítéssel lemossák.

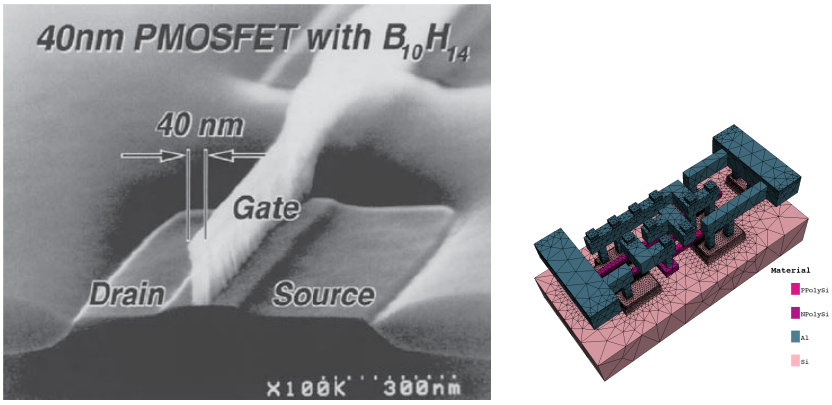
– Az eddigiekhez hasonlóan megtörténik a **4**-es maszk levilágítása. Következik a P és N alaprétegekbe ágyazott P^+ rétegek kialakítása. A szennyezéses rétegkialakítás után az oxidvédőréteg eltávolítása és a felület vizes öblítése következik.
– A kezelt felületre felkerül az újabb oxidvédőréteg.
– Az **5**-ös maszk levilágítása és a fényérzékeny réteg exponálása ebben a lépésben a fémelektródák és -vezetékek kialakítását készíti elő.
– Miután a fölös oxidréteget maratással ismét eltávolítják, következhet a folyamat utolsó lépése.
– Magas hőmérsékleten és nyomáson alumíniumgőzt vezetnek a kamrába, ezzel mintegy megszórva a kitakart felületeket és felépítve a szükséges fémelektródákat és -vezetékeket.
– Végül újabb maszkolással betakarják a fémfelületek szükséges részeit, a többit savas maratással eltávolítják.
– A lemosás és tisztítás után a kész áramkört letesztelik.

A kész integrált áramkörök struktúrái szabad szemmel láthatatlanok. Megtekintésükhöz elektronmikroszkópokat és több 10 000- vagy 100 000-szeres nagyítást kell alkalmazni (4.20. ábra, tinyurl.com/2xk52ewv). Technikai érdekesség, hogy egy processzorokat előállító gyárban a szilíciumkorongot 600-nál is több lépésben kezelik, és a gyár területén belül több száz kilométert tesz meg a különböző berendezések között az alatt a közel 3 hét alatt, amíg egy processzorlapka elkészül. Az egyes stációk között a gyár plafonjára szerelt síneken közlekedő szállítórobotok hordozzák őket (4.19. ábra, tinyurl.com/4384atvu).

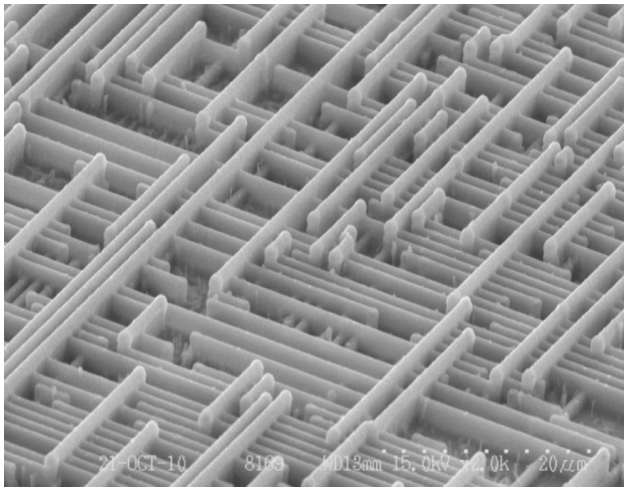


4.19. ábra. *A processzorgyár plafonjára szerelt síneken közlekedő robot targoncák a szilíciumkorongok munkaállomások közötti szállítására*

A gyártástechnológia szinte évről évre változik, új eljárásokat dolgoznak ki, és egyre lennebb szorítják a kialakítható rétegek és térbeli struktúrák méreteit. Az egyik legismertebb gyártástechnológiai jellemző a kapuelektródák szélességét durván jellemző méret. Ma a 4 nanométeres szélesség bevezetése folyik.

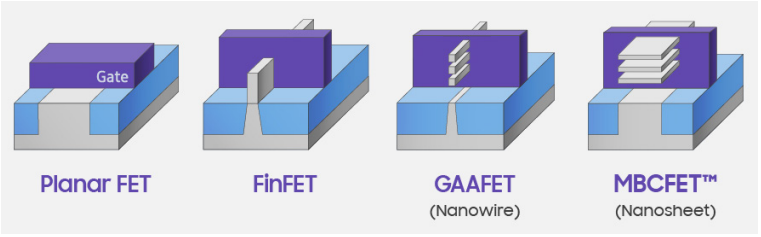


4.20. ábra. *Egy 40 nm-es technológiával előállított FET-kapu (gate) elektródája (100 000-szeres nagyításban) és egy NAND-kapu háromdimenziós, számítógépes modellje*



4.21. ábra. *Egy CMOS logikai áramkör tranzisztorait összekötő vezetékhálózatot láthatunk 80 000-szeres, elektronmikroszkópos nagyításban*

A FET-tranzisztorok kialakítása a kapcsolási sebesség növelése és a kapuelektróda elektromos jellemzőinek javítása okán több változáson is átesett. A kapuelektróda formáját úgy módosították, hogy minél pontosabban befolyásolhassa az alaprétegi vezetőcsatorna kialakulását, mintegy körülvéve azt. A legjobb eredményeket a nano-lapos csatornarétegekkel érték el, ahol az alaprétegi csatornaszakasz vékony lapok formájában, térben kiemelkedve, beépül a kapuelektróda testébe, amely így négy oldalról körbeveszi ezeket (4.22. ábra, tinyurl.com/3eh3yd27). Így még pontosabb és kisebb energiájú elektronáramlás-vezérlést lehet megvalósítani.



4.22. ábra. *A FET tranzisztorok fejlődéstörténete a klasszikus, sík-kapus kialakítástól a nano-lapos kapu változatig*

A félvezetőgyártók folyamatosan kutatják a technológiai folyamat tökéletesítésének lehetőségeit. Napjainkban már az 5 nm-es technológiai szintnél tartanak (Kína, 2022). Ez azt jelenti, hogy a bonyolult digitális áramköröket (processzorokat) alkotó FET tranzisztorok rétegeinek felbontási vastagsága 5 nm, azaz egyes elemeiket képesek létrehozni egymástól akár 5 nm távolságban. Egyes technológiai elemzők szerint a következő évtized nagy vívmánya az 1 nm-es felbontás elérése lesz.

- photolithography process
- making of a chip

5. Számrendszerek – A bináris rendszer mint a számítástechnika alapja

5.1. A bináris rendszer mint a számítástechnika alapja

A számítástechnika vonatkozásában nagy jelentősége van bizonyos számrendszerek ismeretének. A 2. fejezetbe foglalt technikatörténeti áttekintésben több alkalommal is felbukkan a kettes számrendszer. Leibniz egy 1703-ban megjelent dolgozatában vezeti be a tudományos köztudatba, de megfigyelhető, hogy a korai, mechanikus számológépek 10, 12-es vagy éppen 60-as számrendszereket használnak. Ezekben a gépekben a forgó és egymást forgató fogaskerekek fogainak száma, az áttételek aránya az alkalmazott számrendszernek megfelelően alakul. Boole 1847-ben kiadott dolgozata újabb lépést jelent a számítástechnikához köthető tudománytörténetben. IGAZ és HAMIS állítások kombinációját leíró logikai algebrájának operandusai és a logikai függvények eredményei már nagyon közel állnak a kettes számrendszerhez, egyenesen abban értelmezhetők. Shannon 1937-ben kiadott dolgozata összekapcsolja a Boole-algebra függvényeit a villamos kapcsolókkal való műszaki megvalósítás lehetőségével. A kettes számrendszer két számjegyének tökéletesen megfeleltethető NYITOTT és ZÁRT kapcsoló állapotok megnyitják az utat a bináris, digitális áramkörök és számítástechnikai rendszerek fejlődése előtt. Fizikai okokból nagyon könnyen tudjuk binárisan tárolni az információkat, mivel igen könnyű megkülönböztetni a feszültség jelenlétét és hiányát. Ennek megfelelően a feszültség jelenléte lesz a logikai 1-es és annak hiánya a 0-ás érték. A kapcsolt elektromos feszültség viszonylag tág határok között értelmezhető, az alkatrészek különböző távolságokban lehetnek egymástól, a bonyolult számológép vagy számítógép könnyen részegységekre bontható, amelyek fejlesztése a többi alkatrésztől függetlenül végezhető. Vezérelhető kapcsolóként a jelfogókat alkalmazó számítógépek éppen az 1940-es évek elején jelennek meg.

5.2. A számrendszerekről általában

A számrendszerek vagy számábrázolási rendszerek határozzák meg, hogyan ábrázolható egy szám. A számjegy egy olyan szimbólumként értelmezhető, ami létező vagy képzeletbeli fogalmak bizonyos szabályok szerint meghatározott számosságát jelöli. Ennek megfelelően egy számrendszer egységes szabályok alapján határozza meg számunkra, hogy a számjegyek sorozata milyen számokat ír le. Például a használt ábrázolási rendszer fogja meghatározni, hogy az „11” szimbólumot hogyan értelmezhetjük:

– Tízes (decimális) számrendszerben: tizenegy,
– Kettes (bináris) számrendszerben: három,
– Nyolcas (oktális) számrendszerben: kilenc,
– Tizenhatos (hexadecimális) számrendszerben: tizenhét.

A használt számrendszer alapszámának függvényében más-más értékként értelmezhetjük az adott szimbólumot. A számítástechnikai rendszerek vonatkozásában a tízes, kettes, nyolcas és tizenhatos számrendszerek lesznek kiemelten fontosak (5.1. ábra). A továbbiakban vizsgáljuk meg, hogy a felsorolt négy számrendszert hol és milyen szerepkörben használjuk.

A **tízes (decimális) számrendszer** a mindennapi életben használt kiindulási alap, így ehhez viszonyítva tudjuk a többi számrendszert értelmezni. A 10-es számrendszer értékkészlete 0-tól 9-ig tartalmazza a számjegyeket.

A **kettes (bináris) számrendszer** a számítógépek fizikai megvalósításának fontos eleme. A mai számítógépek bináris kódolást használnak, elektronikai szempontból ezt a legkönnyebb megvalósítani. Az alkalmazott informatikában minden információt kettes számrendszerbeli számok segítségével írunk le és a számjegyeket biteknek nevezzük. A 2-es számrendszer értékkészlete a 0 és 1-es számjegyekből épül fel.

A **nyolcas (oktális) számrendszer** értékkészlete 0-tól 7-ig tartalmazza a számjegyeket. Több programozási nyelv (pl. C/C++) és futtatókörnyezet (pl. bash) a nullával kezdődő számokat nem decimálisként, hanem oktálisként értelmezi.

Decimális	Bináris	Oktális	Hexadecimális
0	0000	0	0
1	0001	1	1
2	0010	2	2
3	0011	3	3
4	0100	4	4
5	0101	5	5
6	0110	6	6
7	0111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

5.1. ábra. *Számrendszerek értékkészlet-összehasonlító táblázata*

A **tizenhatos (hexadecimális) számrendszer** értékkészlete 0-tól 9-ig számjegyekből, 10-től 15-ig a latin ábécé első 6 betűjéből (A, B, C, D, E, F) áll. Főként

a programozók és informatikusok körében elterjedt a használata, bináris állományok értelmezésében van segítségünkre, mivel a bináris számrendszerben való értékmegjelenítéshez képest sokkal tömörebb formát alkalmaz, de viszonylag könnyen értelmezhető és bináris formába bontható. Régebben a gépi kódú programok írásánál is használták. Egy hexadecimális számjegy pontosan 4 bitet ír le, 2 számjegy pedig egy bájtot.

Ha megvizsgáljuk a minket körülvevő világot, akkor a tízes számrendszeren kívül több számrendszerre is ráismerhetünk. Napi szinten használjuk őket, de ritkán tudatosodik bennünk, hogy annak szabályossága egy adott számrendszerhez kötődik. Az órák és percek számítása az ókori babiloniak által használt 60-as számrendszerhez köthető, éppen 60 perc felel meg 1 órának, 60 másodperc egy percnek. A 60-as szám megjelenik a szögmérés esetében is, mint váltószám. A tucat, mint mértékegység, a 12-es darabszámot kifejezve máig közkedvelt az angolszász kultúrkörben. Ez a 12-es számrendszerhez köthető. Továbbá az angol és a német nyelvben az első 12 számnak kitüntetett neve van, az óra beosztása a 12-es számra összpontosít, valamint a naptárban is 12 hónapra oszlik egy év. A példákból látható, hogy több számrendszer használata is keveredik mindennapjaink során.

- why binary
- computer's basic number systems

5.3. Helyértékes írásmód

A helyértékes számábrázolás ugyanannak a számnak a számsorban elfoglalt helyétől függő értéket feleltet meg. Ha leírjuk a 8-as számjegyet, akkor tudjuk, hogy az nyolcnak felel meg. De ha ez a számjegy nem az egyesek helyén áll, az értéke már nem 8 lesz, hanem 80 vagy akár 800 is lehet. Ha a tízes számrendszert vesszük példának, akkor a következő helyértéktáblázatban (5.2. ábra) 10-nek a hatványai jelennek meg, ezek szorozzák az egyes helyértékeken feltüntetett számokat. Írjuk fel a 12 419-et a 10 hatványainak a segítségével.

10^4	10^3	10^2	10^1	10^0	10^{-1}	10^{-2}	10^{-3}	10^{-4}
tízezres	ezres	százas	tízes	egyes	tized	század	ezred	tízezred
1	2	4	1	9				

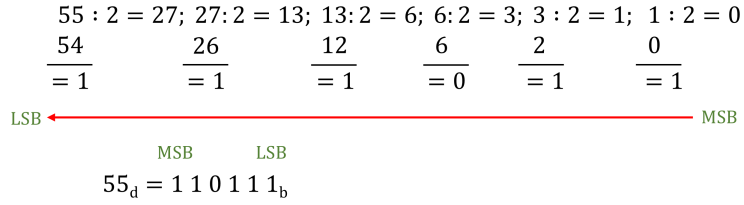
$12\,419 = 1*10^4 + 2*10^3 + 4*10^2 + 1*10^1 + 9*10^0$

5.2. ábra. *10-es számrendszerbeli helyértéktáblázat*

5.4. Számrendszerek közötti átalakítások

5.4.1. 10-es számrendszerből 2-es számrendszerbe alakítás

A 10-es számrendszerből 2-es számrendszerbe való alakítás módszere a **maradékos osztás**, mely módszer szerint az eredő számrendszer alapjával, esetükben a 2-es számmal, addig osztjuk az átalakítandó számot, amíg a hányados nulla nem lesz, az osztások maradékait gondosan leírjuk. Amint a hányados nulla, a maradékokat a sorozatos osztás irányával ellentétesen, visszafelé összegyűjtjük. A fentiek alapján alakítsuk át az 55-öt tízes számrendszerből kettes számrendszerbe (5.3. ábra).

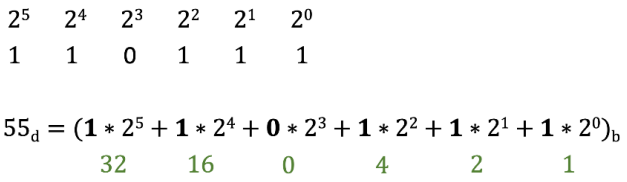


5.3. ábra. *10-es számrendszerből 2-es számrendszerbe alakítás módszere sorozatos osztással*

Az eredményként kapott 6 bit legkisebb helyértéke (Last Significant Bit – LSB) jobbról a legelső érték lesz, valamint a legnagyobb helyérték (Most Significant Bit – MSB) balról a legelső bit lesz. A továbbiakban LSB- és MSB-ként fogunk hivatkozni ezekre az értékekre.

5.4.2. 2-es számrendszerből 10-es számrendszerbe alakítás

Alkalmazva a helyértékes írásmódot, felírhatjuk egy kettes számrendszerbeli érték helyértékeihez társított hatványokat. A számrendszer alapját vesszük a hatvány alapjának, így esetünkben a 2-es számot hatványozzuk. Majd az adott hatványokat szorozzuk a helyértékkel, és ezeket összegezve megkapjuk a binárisan felírt szám tízes számrendszerbeli megfelelőjét (5.4. ábra).



5.4. ábra. *2-es számrendszerből 10-es számrendszerbe alakítás módszere alaphatványozással, helyértékszorzással és sorozatos összeadással*

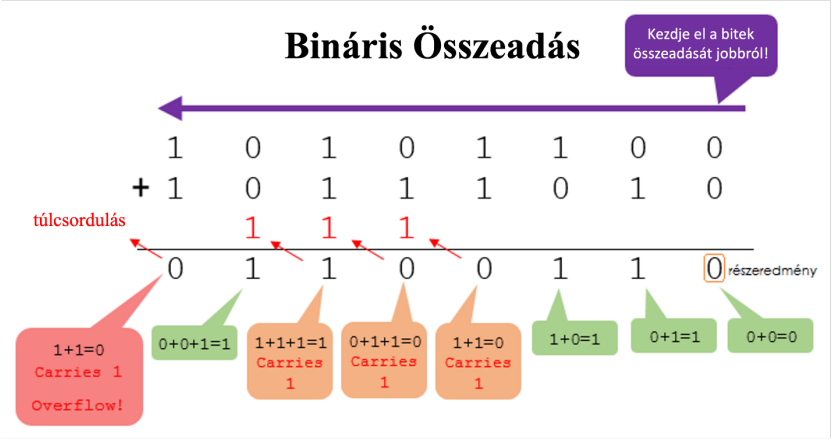
Erre a gondolatmenetre építve egy képlet alapján bármilyen számrendszerbeli számot egyszerűen átalakíthatunk tízes számrendszerbe.

$$\text{DEC} = \sum_{i=0}^{n-1} \text{helyérték} * alap^{i}$$

5.5. ábra. *Általános, 10-es számrendszerbe alakítási képlet*

5.4.3. Bináris összeadás

A bináris összeadás jobbról bal felé haladva, az azonos helyértéken szereplő 1-ek összeadása esetében generál átvitelt, amely értéket a sorrendben következő, nagyobb helyértéken lévő számok részösszegéhez adunk majd hozzá. Ha ebből is átviteli érték keletkezik, azt továbbvisszük, és majd a következő, nagyobb helyértéken lévő számjegyek részösszegéhez adjuk hozzá, míg a keletkező helyértéki részeredményeket mindig lejegyezzük. Ezekből lesz a műveletek végeztével az összeadás eredménye (5.6. ábra, tinyurl.com/mr9xmfuz). Ha a legnagyobb helyértéken is túlcsordulás keletkezik, azt jelenti, hogy az eredményt több biten kell ábrázolni, mint amennyi rendelkezésre áll. Egy processzorban ezt kitüntetett módon jelezni kell (lásd majd később a túlcsordulást jelző bitet – Carry flag).



5.6. ábra. *A bináris összeadás menete*

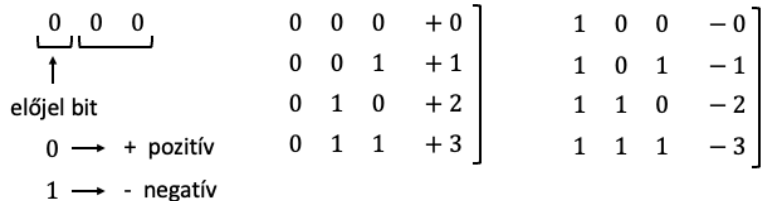
5.5. Negatív számok ábrázolása

Vizsgáljuk meg, hogyan ábrázolhatók kettes számrendszerben a negatív számok. A természetes számok tengelyének 0-tól bal felé terjedő szakaszán ábrázolt, negatív előjelű számok ábrázolására többféle módszer is létezik: egyszerű, kinevezett bithelyes előjel ábrázolás, 1-es komplemens, illetve 2-es komplemens formában való felírás. A különböző ábrázolásmódok használata között azok előnyeinek és hátrányainak figyelembevételével dönthetünk.

5.5.1. Előjeles ábrázolás

Az **előjeles ábrázolás** (5.7. ábra) alatt azt értjük, hogy egy értéket kettes számrendszerbe átírása után a legnagyobb helyértéken (MSB) egy bithellyel bővítünk, ez képviseli az előjelet. Ha ez az **előjel bit** 0, akkor pozitív számunk van, ha 1-es, akkor negatív számunk, majd ezt követi a szám abszolút értéke.

A megoldásnak hátránya, hogy lesz egy pozitív és egy negatív nullánk is, ami matematikailag értelmezhetetlen állapotot teremt. Emiatt ez a módszer nem a legjobb megoldás.



5.7. ábra. *Előjeles ábrázolás módszere*

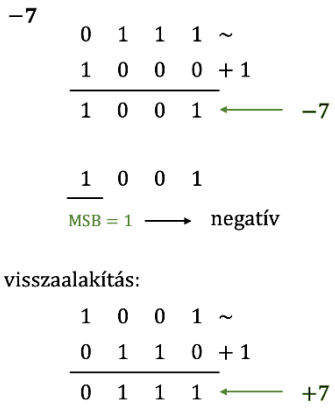
5.5.2. Komplemens képzés módszerei

Az **1-es komplemens** képzés módszere szerint a negatívnak számító bináris számjegy minden tagját tagadjuk, tehát az 1-esekből 0-ások lesznek és a 0-sokból 1-esek. Viszont ebben az esetben is hátrány a plusz (0000) és mínusz (1111) nulla megjelenése.

A **2-es komplemens** formát úgy képezzük, hogy minden bitet tagadunk, az így kapott értékhez hozzáadunk 1-et, és csak az eredeti számú bitet őrizzük meg, túlcsordulás esetén nem vesszük figyelembe a túlcsorduló bitet (5.8. ábra).

Írjuk fel a -7-et binárisan. Első lépésként fel kell írnunk a +7-et binárisan, majd alkalmaznunk kell a 2-es komplemens módszerét, vagyis bitenként tagadjuk a kiindulási értéket, majd hozzáadunk 1-et. Az MSB fogja tárolni az előjelet. Láthatjuk, hogy 1 az MSB, tehát egy negatív értéket kaptunk eredményként, a -7-et.

Ha egy bináris értékről szeretnénk megállapítani, hogy milyen tízes számrendszerbeli számot ábrázol, akkor elsőként meg kell vizsgálnunk az MSB-t. Ha az MSB 0, akkor pozitív számunk van, így az eddigi ismereteink alapján átalakítjuk a 2-es számrendszerbeli számot 10-es számrendszerbe. Ha az MSB 1, akkor tudjuk, hogy negatív lesz az eredményünk. Ebben az esetben megjegyezzük, hogy negatív számot kell kapnunk, majd a 2-es komplemens képzés módszerét alkalmazva bitenként tagadjuk a számot és az eredményhez hozzáadunk egyet, majd a kapott eredményt átalakítjuk 10-es számrendszerbe, így megkapjuk az eredményt abszolút értékben és ezt kiegészítjük a negatív előjellel.



5.8. ábra. *Negatív bináris szám képzése 2-es komplemens formába alakítással*

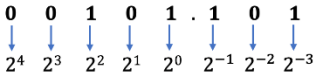

- binary addition
- two's complement method

5.6. Tizedes számok ábrázolása

A tizedes számokat, vagyis a törtszámokat is át tudjuk alakítani kettes számrendszerbe. Ebben az esetben a vessző után következő tizedes érték kettőnek a negatív hatványaiként fejezhető ki. Fordított irányban a kettes számrendszerben felírt tizedes számot is át tudjuk alakítani tízes számrendszerbe.

Kétféle ábrázolási módszert különböztetünk meg: a fixpontos és a lebegőpontos ábrázolást. A fixpontos ábrázolás (5.9. ábra) esetében egy szám kettes számrendszerbeli számjegyeit rögzített nagyságú memóriaterületen tároljuk, a szám helyi értékének megfelelően. Ilyenkor a szám egész és a tört részét elválasztó jel, a tizedesvessző elhelyezkedése előre meghatározott. Minden tizedes számjegyeket

kezelni tudó számítógép-architektúrában rögzített helye van a tizedespontnak, és a gép a tárolt bitsorozatokat ennek megfelelően értelmezi. A lebegőpontos ábrázolásnál az elválasztó tizedesvesszőnek nincs megfeleltetett memóriaterület-felosztás, a számok rögzített számú számjegyekkel ábrázolhatók, és egy kitevő (exponens) segítségével vannak skálázva.



$$0 * 2^4 + 0 * 2^3 + 1 * 2^2 + 0 * 2^1 + 1 * 2^0 + 1 * 2^{-1} + 0 * 2^{-2} + 1 * 2^{-3} = 4 + 1 + \frac{1}{2^1} + \frac{1}{2^3} = 5 + 0.5 + \frac{1}{8}$$
$$= 5.5 + 0.125 = 5.625$$

1 0 1 . 1 0 0 0 0

$$1 * 2^2 + 1 * 2^0 + 1 * 2^{-1} = 4 + 1 + 0.5 = 5.5$$

1 0 1 . 1 0 0 0 1

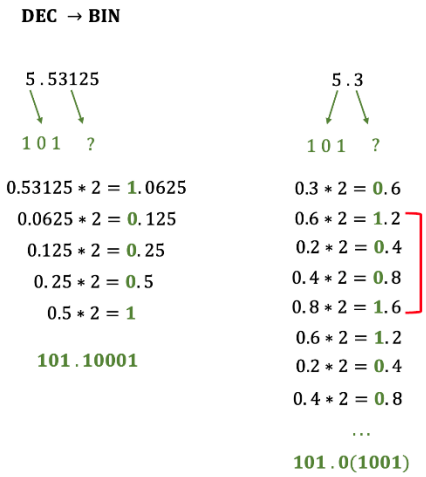
$$1 * 2^2 + 1 * 2^0 + 1 * 2^{-1} + 1 * 2^{-5} = 4 + 1 + 0.5 + \frac{1}{2^5} = 5.5 + 0.03125 = 5.53125$$

$2^5 = 32$

5.9. ábra. *Binárisan ábrázolt, fixpontos tizedes számok 10-es számrendszerbe alakításának módszere*

A kettes számrendszerben felírt tizedes számot az eddigi ismereteink alapján át tudjuk alakítani tízes számrendszerbe. A tizedesvessző bal oldalán elhelyezkedő értéket a 2-nek a pozitív hatványaival szorozzuk, míg a tizedesvessző jobb oldalán elhelyezkedő értékeket a 2-nek a negatív hatványaival, majd a részeredményeket összeadjuk.

Ha egy tízes számrendszerben felírt tizedes számot szeretnénk átírni kettes számrendszerbe, akkor az eddig alkalmazott módszer szerint a szám egész részét sorozatosan osztjuk 2-vel, a maradékokat pedig az utolsó számítástól visszafele leolvassuk. Így megkapjuk az egész szám átalakított értékét (5.10. ábra). A törtrészt megszorozzuk kettővel, és a szorzás eredményeként kapott egész rész (1 vagy 0) meghatározza a kiszámolandó tizedes szám bináris értékeit. A szorzást ideális esetben addig végezzük, amíg a tizedes vagy a törtrész nulla lesz, vagy megindul az ismétlés (szakaszos tizedes tört).



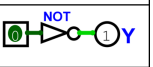
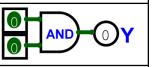
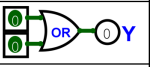
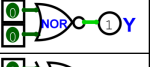
5.10. ábra. *10-es számrendszerbeli tizedes számok 2-es számrendszerbe alakításának módszere*

6. A logikai kapuk és alapáramkörök – A digitális rendszerek alapjai

6.1. Logikai alapfüggvények és kapuáramkörök

Egy számítástechnikai rendszerben a bináris formában információt hordozó jeleket LOGIKAI KAPUK vagy LOGIKAI ÁRAMKÖRÖK segítségével dolgozzuk fel, irányítjuk, szervezzük és értelmezzük. A logikai kapuk LOGIKAI ALAPFÜGGVÉNYEKET (6.1. ábra) valósítanak meg, áramköreik segítségével a bemeneteiken beállított feszültségszinteknek megfelelő kimeneteket generálnak (ez csak a feszültségszintek működés közbeni betartásával lehetséges).

A logikai kapuk működésének matematikai alapját, a logikai alapfüggvények tulajdonságait a Boole-algebra írja le (lásd 33. oldal). Később, ehhez kapcsolódóan Shannon fogalmazta meg a logikai alapfüggvények kapcsolókkal való megvalósításának módszerét (lásd 35. oldal). Ezekre az elméleti szálakra alapozva születik meg a felismerés, amely szerint egymással kombinált logikai függvényekkel leírható akár egy számítástechnikai rendszer működése is. A félvezető technológiával megvalósított logikai kapuk képezik a számítógépek bonyolult logikai áramköreit. A tagadó kapu kivételével minden más alap logikai függvény két bemenettel (A, B) és egy kimenettel (Y) rendelkezik.

Bemeneti változók / Kapu típusok	A	0	1	0	1	Áramköri jelölés
	B	0	0	1	1	
TAGADÓ (NOT)	$Y=\overline{A}$	1	0	1	0	
ÉS (AND)	$Y=A\cdot B$	0	0	0	1	
VAGY (OR)	$Y=A+B$	0	1	1	1	
NEM-ÉS (NAND)	$Y=\overline{A\cdot B}$	1	1	1	0	
NEM-VAGY (NOR)	$Y=\overline{A+B}$	1	0	0	0	
KIZÁRÓ-VAGY (XOR)	$Y=A\cdot\overline{B}+\overline{A}\cdot B=A\oplus B$	0	1	1	0	
KIZÁRÓ NEM-VAGY (XNOR)	$Y=(A\cdot B)+(\overline{A}\cdot\overline{B})=\overline{A\oplus B}$	1	0	0	1	

6.1. ábra. *Logikai alapfüggvények igazságtáblázatai és áramköri szimbólumai*

Az alap logikai kapuk segítségével, ezek kimeneteinek és bemeneteinek egymás után kapcsolásával (elektromos összekötésével) bonyolultabb logikai áramköröket hozhatunk létre. A bonyolultabb logikai áramkörök két nagy csoportra oszthatók: KOMBINÁCIÓS és SZEKVENCIÁLIS ÁRAMKÖRÖK.

A KOMBINÁCIÓS áramkörök esetében a kimenetek állapota a bemenetek aktuális állapotából vezethető le az áramkört leíró függvény alapján (a függvény a Boole-algebra szabályainak megfelelően van felírva), illetve a függvény IGAZSÁG TÁBLÁZAT formában való felírásával szemléltethető a bemenetek és kimenetek egyidejű állapota. Ilyen áramkörök a dekódolók, multiplexerek, demultiplexerek, teljes összeadók stb.

A SORRENDI vagy SZEKVENCIÁLIS áramkörök esetében a kimenetek állapota a bemenetek aktuális állapotából és az egyes bemenetekre visszacsatolt kimenetek előző állapotából vezethető le. Ilyen áramkörök az RS-tárolók, a D-tárolók, a JK-mester-szolgák, és a belőlük felépített számlálók, regiszterek stb.

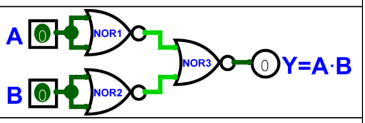
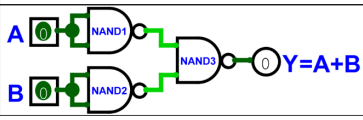
A logikai kapukból felépített, bonyolultabb logikai áramkörök kialakításának módszerei nagyban támaszkodnak a Boole-algebra és De Morgan azonossági tételeire (6.2. ábra), amelyek lényegében adott logikai műveletek többféle módon való leírását rögzítik.

Azonosság neve	ÉS forma	VAGY forma
Identitásszabály	$1 \cdot A = A$	$0 + A = A$
Nullszabály	$0 \cdot A = A$	$1 + A = 1$
Idempotens szabály	$A \cdot A = A$	$A + A = A$
Inverz szabály	$A \cdot \overline{A} = 0$	$A + \overline{A} = 1$
Kommutatív szabály	$A \cdot B = B \cdot A$	$A + B = B + A$
Asszociatív szabály	$(A \cdot B) \cdot C = A \cdot (B \cdot C)$	$(A + B) + C = A + (B + C)$
Disztribúciós szabály	$A + (B \cdot C) = (A + B) \cdot (A + C)$	$A \cdot (B + C) = A \cdot B + A \cdot C$
Abszorpciós szabály	$A \cdot (A + B) = A$	$A + A \cdot B = A$
De Morgan-szabály	$\overline{A \cdot B} = \overline{A} + \overline{B}$	$\overline{A + B} = \overline{A} \cdot \overline{B}$

6.2. ábra. *A Boole-algebra azonosságai*

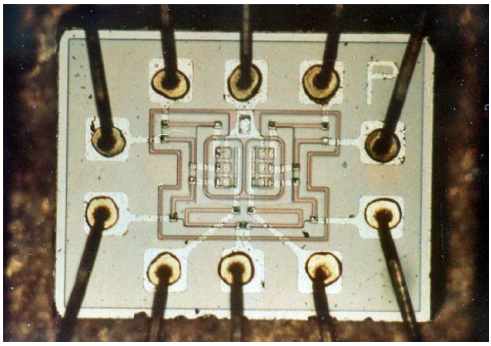
Az azonossági képletekből kiindulva megállapítható, hogy a NEM-ÉS (NAND), valamint NEM-VAGY (NOR) úgynevezett univerzális kapuk, mivel ezekből a kapukból az összes logikai függvény megvalósítható (6.3. ábra).

NAND RENDSZER		
NEM művelet	$Y = \overline{A \cdot A} = \overline{A} + \overline{A} = \overline{A}$	A NEM művelet 1 NAND kapuval megvalósítható.
ÉS művelet	$Y = \overline{\overline{A \cdot B}} = A \cdot B$	Az ÉS művelet 2 NAND kapuval megvalósítható.
VAGY művelet	$Y = \overline{\overline{A} \cdot \overline{B}} = A + B$	A VAGY művelet 3 NAND kapuval megvalósítható.
NOR RENDSZER		
NEM művelet	$Y = \overline{A + A} = \overline{A} \cdot \overline{A} = \overline{A}$	A NEM művelet 1 NOR kapuval megvalósítható.
ÉS művelet	$Y = \overline{\overline{A} + \overline{B}} = A \cdot B$	Az ÉS művelet 3 NOR kapuval megvalósítható.
VAGY művelet	$Y = \overline{A + B} = A + B$	A VAGY művelet 2 NOR kapuval megvalósítható.

LOGIKAI ALAP MŰVELET	LOGIKAI RENDSZER	
	NAND RENDSZER	NOR RENDSZER
NEM		
ÉS		
VAGY		

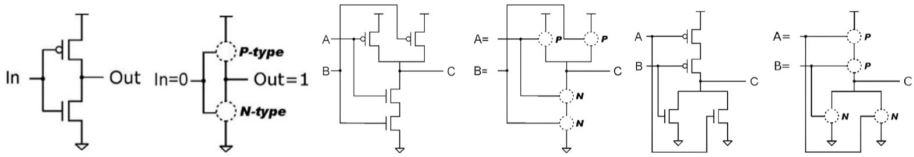
6.3. ábra. *A De Morgan-féle azonossági tételek alapján a NEM, ÉS, VAGY függvények kifejezhetőek NAND-, valamint NOR-függvényekkel*

Említésre méltó, hogy az első integrált áramköri megvalósítások éppen a NOR-kapukhoz kapcsolódnak (6.4. ábra, tinyurl.com/4w3ktzcb).

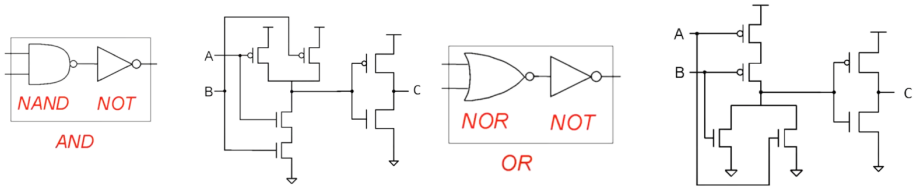


6.4. ábra. *Az első NOR-kapu integráltáramkör-implementációk egyike, az Apollo holdprogram űrhajóinak fedélzeti számítógépeinek legfőbb alapáramköre*

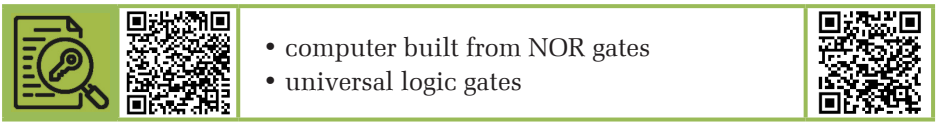
Kezdetben RTL (Resistor-Transistor Logic), majd DTL (Diode-Transistor Logic), végül TTL (Transistor-Transistor Logic) típusú struktúrákat gyártanak, majd mindezek helyét az 1970-es évek elején átveszik CMOS (Complementary Metal-Oxide Semiconductor, N és P MOSFET komplementer tranzisztorokból kialakított logikai kapuk) áramkörök. A logikai kapuk belső felépítésüket tekintve elektromosan egymáshoz kapcsolt MOSFET tranzisztorokból állnak (6.5. ábra), ezek kapcsoló üzemmódban működve, a Shannon által definiált módon valósítják meg NOR- vagy NAND-kapukból a kívánt logikai függvényeket (6.6. ábra).



6.5. ábra. *A NOT-, NAND- és NOR-kapuk kapcsolási és ekvivalens logikai rajza*



6.6. ábra. *Az univerzális kapukból (NAND, NOR) képezett logikai függvények (AND, OR) kapcsolási rajza*



- computer built from NOR gates
- universal logic gates

6.2. Logikai kapukból képezett áramkörök szimulációja

A további fejezetek során igyekeztünk minden magyarázathoz elkészíteni a tárgyalt áramkörök szimulációs rajzát. A logikai kapuk szimbólumait, a ki- és bemeneteik közötti vezetékeket és bizonyos viselkedési sajátosságokat egy grafikus felhasználói felülettel rendelkező programot használva megrajzoltunk és beállítottunk. Ezután lehetőség van a kialakított áramkörök virtuális, programon belüli működtetésére és megfigyelésére. A szimulációs környezet neve LOGISIM,

egy szabad forráskódú, egyetemi körökben igen elterjedt program. A világ számos egyetemén használják a számítógépes architektúrákkal kapcsolatos tantárgyak gyakorlati bemutatására. A szimulációs környezet támogatja a hierarchikus tervezési módot, vagyis egyszerű kapukból kialakított áramköröket funkcionális tömbökbe tudunk csomagolni. Majd ezeket a tömböket még bonyolultabb tömbökbe vonhatjuk össze, amíg a kívánt komplexitást el nem érjük. A LOGISIM használatáról részletes leírás található a könyv MELLÉKLETÉNEK 11.3-az fejezetében.

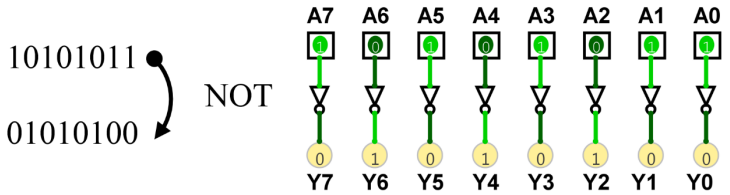
- Logisim Evolution
- free digital circuit simulator

6.3. Bitszintű műveletek logikai kapukkal

Ha egynél több logikai kapura terjesztjük ki a bitek szintjén elvégezhető logikai műveletek sorozatát, több egyszerű, de a számítástechnika területén hasznos vagy éppen nélkülözhetetlen függvényt valósíthatunk meg. A bitművelet a célzott bitsorozatot vagy bináris számot az egyes bitek szintjén változtatja meg. A következőkben nézzük meg a tagadás, a logikai **ÉS**, logikai **VAGY** és a **KIZÁRÓ VAGY** műveletek számítástechnikai rendszerekben feldogozott bináris számokra vonatkoztatott hatását.

6.3.1. Tagadás művelet

A logikai negáció vagy tagadás művelet bitszintű invertálást vagy komplemens képzést jelent (6.7. ábra). A bitek számával egyező számú tagadó (NOT) kapu segítségével az adott bináris érték 1-es komplemensét képezzük (a negatív bináris számok képzésénél van jelentősége). A bináris szám minden 0-jából 1 lesz, minden 1-éből 0.

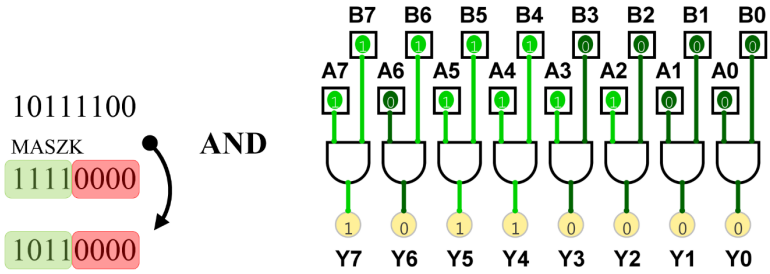


6.7. ábra. *Bináris számsor logikai negációja*

6.3.2. Maszkolási műveletek

Maszkolási műveletnek nevezzük azokat a bitszintű műveleteket, amelyekben egy bitsorozat kiválasztott bitjeinek értékét valamilyen logikai függvény szerint megváltoztatjuk. Ez a változtatás annak a függvénye, hogy milyen logikai művelet áll a maszk és a bemeneti bitsorozat bitértékei között. Maszkolási műveletet használhatunk egyes bitek értékeinek 0-ra vagy 1-re való módosításához úgy, hogy közben a bitsor többi bithelyének értéke változatlan marad. Adott esetben a bitsorozat lehet egy regiszter tartalma vagy egy memóriában tárolt változó értéke is.

Logikai **és** műveletet (konjunkció, 6.8. ábra) alkalmazva a maszk és egy tetszőleges (de a maszkkal azonos hosszúságú) bináris szám között az eredményül kapott számban azokon a helyértékeken, ahol mindkét bit 1-es értékű volt, 1-es lesz az eredmény (1 **és** 1 = 1); egyébként az eredmény 0 (1 **és** 0 = 0). Ezzel a művelettel hatékonyan „kiszűrhetjük” a bemeneti bináris számsor egy részét, akár egyetlen vizsgálni kívánt bitre „csökkentve” annak értelmezési halmazát.



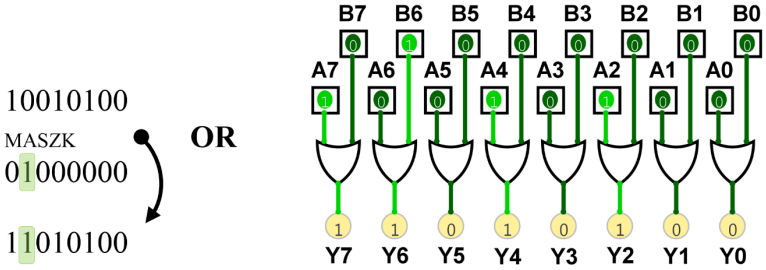
6.8. ábra. *Visszatartó vagy fosztó maszkolás **és** művelettel*

Az ábrán látható példában a bemenő érték (A) alsó négy bitjét a maszkkal (B) „kiszűrjük”, vagy áthaladását a szűrőn „nem engedélyezzük”, mindig 0-ban tartjuk.

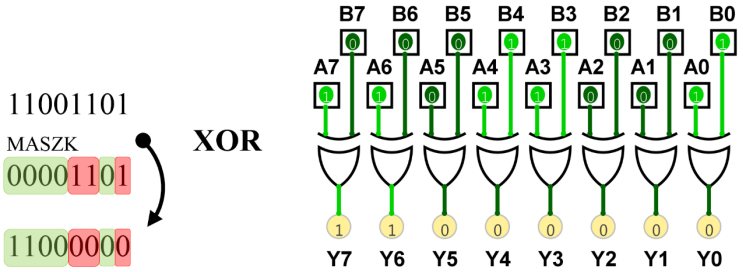
Logikai **vagy** műveletet (diszjunkció, 6.9. ábra) alkalmazva a maszk és egy tetszőleges (de a maszkkal azonos hosszúságú) bináris szám között az eredményül kapott számban azokon a helyértékeken, ahol a maszkban 1-es érték volt, 1-es lesz az eredmény (1 **vagy** bármi = 1); egyébként az eredmény a bemeneti értéket tartja (0 **vagy** bármi = bármi). Ezzel a művelettel a bemeneti bináris számsor bizonyos bithelyeit logikai 1-be állíthatjuk anélkül, hogy a számsor többi részét megváltoztatnánk.

A **kizáró vagy** műveletet alkalmazva (6.10. ábra) a maszk és egy tetszőleges (de a maszkkal azonos hosszúságú) bináris szám között az eredményül kapott számban azokon a helyértékeken, ahol a maszkban 0-ás érték van, az eredmény az eredeti bemenő értéket tartja (1 **xor** 0 = 0, 0 **xor** 0 = 0). Ott, ahol a maszkban 1-es érték van, az eredmény a bemeneti érték negáltját hozza (0 **xor** 1 = 1, 1 **xor** 1 = 0). Ezzel

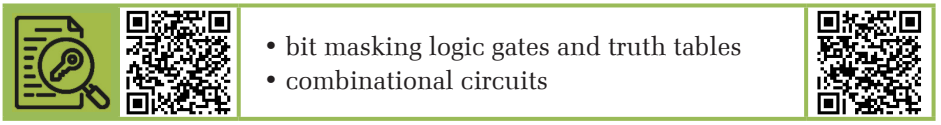
a művelettel a bemeneti bináris számsor bizonyos bithelyeinek logikai értékét tagadhatjuk, negálhatjuk anélkül, hogy a számsor többi részét megváltoztatnánk.



6.9. ábra. *Beállító maszkolás* VAGY *művelettel*



6.10. ábra. *Bithely negáló maszkolás* KIZÁRÓ VAGY *művelettel*



6.4. Kombinációs áramkörök

6.4.1. Bináris dekódoló

A kombinációs áramkörök alap logikai kapukból létrehozott hálózatai különböző, egyszerű funkciókat valósítanak meg. Ilyenek például a dekódolók, kódolók, multiplexerek és demultiplexerek. Minden ilyen áramkör a későbbiekben kialakítandó processzor és számítástechnikai rendszer összeállításában fontos szerephez jut.

Az alap logikai függvényekből kiindulva viszonylag könnyen értelmezhető egy úgynevezett BINÁRIS DEKÓDOLÓ, más néven bináris kiválasztó áramkör működése. Számítástechnikai rendszerekben ilyen áramkörök választják ki az egyes memóriacellákat egy bináris cím (szám) alapján. Egy dekódolónak n bemenete és 2^N kimenete van, amelyek közül mindig csak egy aktív, sorrendben a dekódoló áramkör bemenetén beállított bináris számnak megfelelően. Egy 3 bites dekódoló tervezését kezdjük a működését leíró igazságtáblázattal (6.11. ábra):

A_2	A_1	A_0	Y_7	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	1	0	0
0	1	1	0	0	0	0	1	0	0	0
1	0	0	0	0	0	1	0	0	0	0
1	0	1	0	0	1	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0

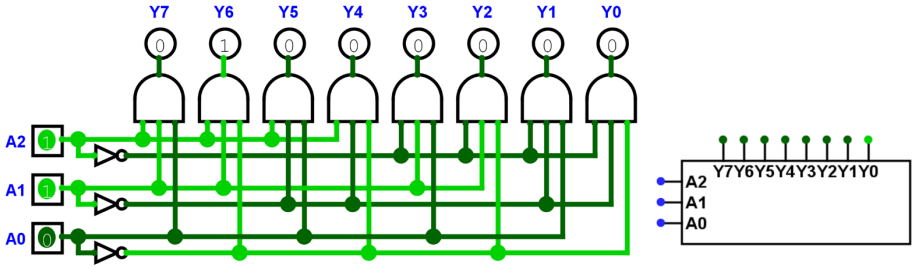
6.11. *ábra. A 3 bites bináris dekódoló igazságtáblázata*

A három bemenetet A, B és C-vel jelöljük, a nyolc kimenetet Y_0-tól Y_7-ig számozzuk. Minden bemeneti kombinációra csak egy kimenet lehet aktív, ezért a leíró logikai függvények a 6.12. ábra szerint alakulnak:

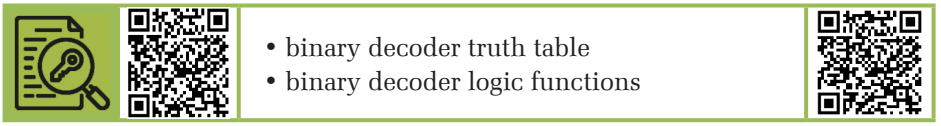
$$Y_0 = \bar{A} \cdot \bar{B} \cdot \bar{C} \quad Y_1 = \bar{A} \cdot \bar{B} \cdot C \quad Y_2 = \bar{A} \cdot B \cdot \bar{C}$$
$$Y_3 = \bar{A} \cdot B \cdot C \quad Y_4 = A \cdot \bar{B} \cdot \bar{C} \quad Y_5 = A \cdot \bar{B} \cdot C$$
$$Y_6 = A \cdot B \cdot \bar{C} \quad Y_7 = A \cdot B \cdot C$$

6.12. ábra. *A bináris dekódoló logikai függvényei*

Így olvashatjuk ki őket: Y_0 akkor 1-es, ha $\bar{A}$ **és** $\bar{B}$ **és** $\bar{C}$ igaz, Y_1 akkor 1-es, ha $\bar{A}$ **és** $\bar{B}$ **és** C igaz, stb. Észrevehető, hogy mindhárom bemenet szerepel direkt és tagadott formában is. A megvalósítást a három bemeneti vonal direkt és tagadott formáinak kialakításával kezdjük, ez utóbbiakhoz **tagadó** kapukat használva. Az egyes függvényeket ezután a megfelelő bemenetekkel ellátott három bemenetű **és** kapukkal valósítjuk meg (6.13. ábra).

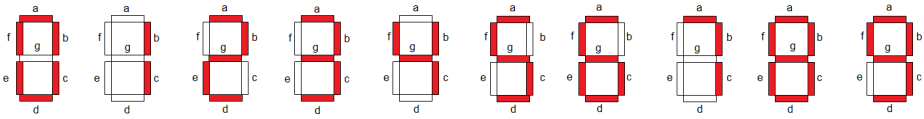


6.13. ábra. *A 3 bites bináris dekódoló kapcsolási rajza és szimbóluma*



6.4.2. 7 szegmenses kijelző dekódoló

A bináris dekódoló a memóriacellák kiválasztásához nélkülözhetetlen, ha viszont számjegyeket kell megjeleníteni, egy másik változatát használjuk a megvalósításhoz. Mindenki találkozott már az úgynevezett 7 szegmenses kijelzővel, egy olyan eszközzel, amely LED-ek segítségével apró, sajátos térbeli elrendezésű vonalkákat vagy szegmenseket világít meg, amelyek egy 0-tól 9-ig terjedő arab számjegy formáját jeleníti meg (6.14. ábra).



6.14. ábra. *A 7 szegmenses kijelzőn megjeleníthető arab számjegyek*

A dekódoló tervezését a kimenetek összeszámlálásával kezdjük, a 7 világító szegmensnek 7 kimenet felel meg, amelyek 10-féle kombinációban lehetnek aktívak. 10 számjegynek megfelelő kombinációt 4 biten tudunk bináris formában megjeleníteni, ezért 4 bemenetünk lesz. A szegmenseket „a”-tól „g”-ig jelöljük, a bemeneteket A, B, C és D-vel.

Következő lépésben feltérképezzük az egyes számjegyek megjelenítéséhez szükséges szegmenseket, és megállapíthatjuk, hogy például a „0”-ás számjegy megjelenítéséhez az a, b, c, d, e, f szegmenseknek kell világítaniuk, azaz a dekódoló ezen kimenetei logikai 1-ben kell álljanak, míg az „5”-ös számjegy esetében az a, c, d, f, g szegmensek világítanak. Ezen megállapítások alapján felírjuk a 7 szegmenses dekódoló igazságtáblázatát a 10 számjegyre, ahogy a 6.15. ábra mutatja.

D	C	B	A	a	b	c	d	e	f	g
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	1	0	1	1

6.15. ábra. *A 7 szegmenses dekódoló igazságtáblázata a „0”-ás és az „5”-ös számjegyek megjelenítéséhez tartozó, 1-es állapotú kimenetek megjelölésével*

Az igazságtáblázat alapján felírhatjuk az egyes kimenetek logikai függvényeit (6.16. ábra), megfigyelve, hogy mely bemeneti kombinációk esetében 1-es az adott kimenet:

$$a = \bar{A}\cdot\bar{B}\cdot\bar{C}\cdot\bar{D} + \bar{A}\cdot\bar{B}\cdot C\cdot\bar{D} + \bar{A}\cdot\bar{B}\cdot C\cdot D + \bar{A}\cdot B\cdot\bar{C}\cdot D + \bar{A}\cdot B\cdot C\cdot\bar{D} + \bar{A}\cdot B\cdot C\cdot D + A\cdot\bar{B}\cdot\bar{C}\cdot\bar{D} + A\cdot\bar{B}\cdot\bar{C}\cdot D$$

$$b = \bar{A}\cdot\bar{B}\cdot\bar{C}\cdot\bar{D} + \bar{A}\cdot\bar{B}\cdot\bar{C}\cdot D + \bar{A}\cdot\bar{B}\cdot C\cdot\bar{D} + \bar{A}\cdot\bar{B}\cdot C\cdot D + \bar{A}\cdot B\cdot\bar{C}\cdot\bar{D} + \bar{A}\cdot B\cdot C\cdot D + A\cdot\bar{B}\cdot\bar{C}\cdot\bar{D} + A\cdot\bar{B}\cdot\bar{C}\cdot D$$

$$c = \bar{A}\cdot\bar{B}\cdot\bar{C}\cdot\bar{D} + \bar{A}\cdot\bar{B}\cdot\bar{C}\cdot D + \bar{A}\cdot\bar{B}\cdot C\cdot D + \bar{A}\cdot B\cdot\bar{C}\cdot\bar{D} + \bar{A}\cdot B\cdot\bar{C}\cdot D + \bar{A}\cdot B\cdot C\cdot\bar{D} + A\cdot\bar{B}\cdot\bar{C}\cdot\bar{D} + A\cdot\bar{B}\cdot\bar{C}\cdot D$$

$$d = \bar{A}\cdot\bar{B}\cdot\bar{C}\cdot\bar{D} + \bar{A}\cdot\bar{B}\cdot C\cdot\bar{D} + \bar{A}\cdot\bar{B}\cdot C\cdot D + \bar{A}\cdot B\cdot\bar{C}\cdot D + \bar{A}\cdot B\cdot C\cdot\bar{D} + A\cdot\bar{B}\cdot\bar{C}\cdot\bar{D} + A\cdot\bar{B}\cdot\bar{C}\cdot D$$

$$e = \bar{A}\cdot\bar{B}\cdot\bar{C}\cdot\bar{D} + \bar{A}\cdot\bar{B}\cdot C\cdot\bar{D} + \bar{A}\cdot B\cdot C\cdot\bar{D} + A\cdot\bar{B}\cdot\bar{C}\cdot\bar{D}$$

$$f = \bar{A}\cdot\bar{B}\cdot\bar{C}\cdot\bar{D} + \bar{A}\cdot B\cdot\bar{C}\cdot\bar{D} + \bar{A}\cdot B\cdot\bar{C}\cdot D + \bar{A}\cdot B\cdot C\cdot\bar{D} + A\cdot\bar{B}\cdot\bar{C}\cdot\bar{D} + A\cdot\bar{B}\cdot\bar{C}\cdot D$$

$$g = \bar{A}\cdot\bar{B}\cdot C\cdot\bar{D} + \bar{A}\cdot\bar{B}\cdot C\cdot D + \bar{A}\cdot B\cdot\bar{C}\cdot\bar{D} + \bar{A}\cdot B\cdot\bar{C}\cdot D + A\cdot\bar{B}\cdot\bar{C}\cdot\bar{D} + A\cdot\bar{B}\cdot\bar{C}\cdot D$$

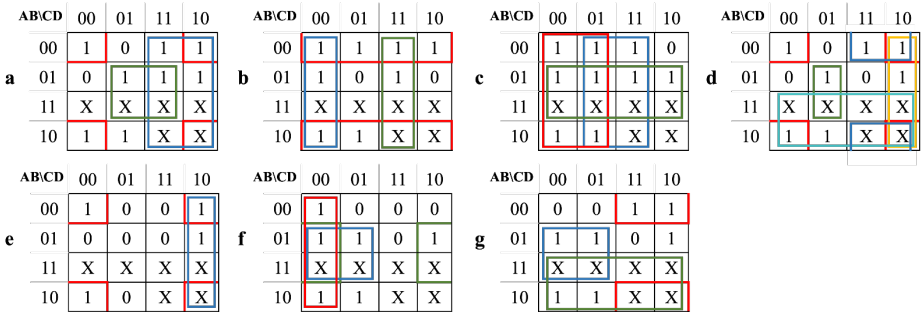
6.16. ábra. *A 7 szegmenses dekódolót leíró logikai függvények színessel kiemelt ismétlődő logikai összefüggésekkel*

A logikai áramkörök tervezése során gyakran előfordul, hogy egy logikai függvény több ismétlődő logikai összefüggést tartalmazhat. Ezeket a megépítés hatékonysága (minél kevesebb logikai kapuból legyen megvalósítható) érdekében MINIMIZÁLNI próbáljuk a Karnaugh-diagram segítségével. Maurice Karnaugh 1953-ban publikálja *The Map Method for Synthesis of Combinational Logic Circuits* című dolgozatát. Ebben egy olyan vizuális megközelítésen alapuló módszert közöl, amely táblázatos, sűrített formában jeleníti meg a vizsgált logikai függvény kimeneteit a

bemenetek függvényében. A több bemeneti változós függvények esetében a bemenetek bináris kombinációi csoportosítva jelennek meg, például 4 bemeneti változó esetében vízszintesen az C és D, függőlegesen az A és B bemenetek (00, 01, 11, 10 – Gray-kódolás szerinti) érték kombinációi. A táblázatokban szereplő kimeneti értékek különböző szabályok szerinti csoportosításával az ismétlődő logikai összefüggések kiszűrhetők, így a kimenet állapotait leíró logikai függvények minimizálhatóak (egyszerűsíthetők), és ezáltal kevesebb logikai kapu felhasználásával építhetők meg.

A dekódoló áramkör esetében 7 Karnaugh-diagramot tudunk felírni, minden kimenetre (szegmensre) vonatkoztatva egyet-egyet. Azokhoz a bemeneti kombinációkhoz, amelyeket nem veszünk figyelembe (a 9-nél nagyobb számjegyeket), „X”-et írunk a táblázatba.

A Karnaugh-módszer szabályai alapján a táblázatokban megkeressük azokat a kimenetcsoportokat, amelyek egymás közvetlen szomszédságában 2-es, 4-es, 8-as csoportokban 1-es vagy X értékeket tartalmaznak. Kereshetünk 0-ás csoportokat is, ha a kimenetek szempontjából ez a logikai érték a meghatározó, illetve ki lehet egyetlen kimenetből álló „izolált csoportot” is jelölni. Nemcsak a táblázat belsejében kereshetünk ilyen csoportokat, hanem a táblázat szélein átlépve, az ellentétes oldali szélekkel és sarkokkal is kombinálhatunk (mintegy térben „behajlítva”, képzeletbeli csővé, tórusszá vagy éppen gömbbé változtatva a táblázatot). Egy adott pozícióban lévő, 1-es értékű cella egyszerre több csoport tagja is lehet.



6.17. ábra. *A 7 szegmenses dekódoló Karnaugh-diagramjai*

A 6.17. ábra táblázataiban rendre zöld, kék, piros és sárga színekkel jelöltük az összes olyan csoportot, amelyekben a 2 hatványainak megfelelő számú, szomszédos 1-es értékeket találtunk.

Vegyük az első, az „a” szegmensnek megfelelő táblázatot. Felírhatjuk a zöld csoportnak megfelelő függvényrészt, azokat a bemeneteket kiválasztva, amelyek nem változtatják meg az értéküket a csoporton belül (invariánsok), ez itt a: B ÉS D. A kék csoportnak megfelelő részre a C-t írjuk fel. Következik a sárga csoport A invariánssal. Végül a piros csoport a $\overline{B}$ ÉS $\overline{D}$-vel zárja a sort. A kijelölt csoportok a logikai függvényben VAGY függvénnyel kapcsolódnak össze.

A függvény így alakul: a = A + C + B·D + $\overline{B}$·$\overline{D}$.

A „b” szegmensnek megfelelő táblázatban csak három, a szabályoknak megfelelő csoportot találunk. A zöld csoportban a CD invariáns. A kék csoportban a $\overline{C \cdot D}$ az invariáns, míg a piros csoportban a B az invariáns bemeneti érték.

A függvény így alakul: b = $\overline{B}$+$\overline{C}$·$\overline{D}$+ C·D.

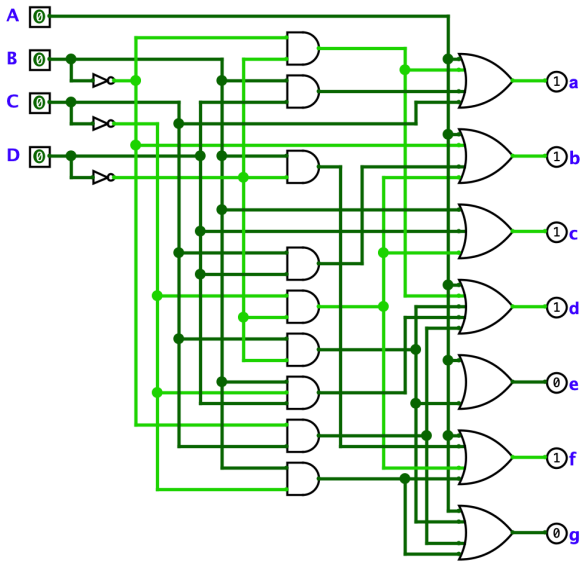
Az ily módon egyszerűsített függvények (6.18. ábra) láthatóan jól különböznek a 6.16. ábra formáitól, sokkal kevesebb logikai kombinációt tartalmaznak, ennélfogva kevesebb logikai kapuból építhető meg az áramkör.

$$a = A + C + B \cdot D + \bar{B} \cdot \bar{D} \qquad b = \bar{B} + \bar{C} \cdot \bar{D} + C \cdot D \qquad c = B + \bar{C} + D$$

$$d = \bar{B} \cdot \bar{D} + C \cdot \bar{D} + B \cdot \bar{C} \cdot D + \bar{B} \cdot C + A \qquad e = \bar{B} \cdot \bar{D} + C \cdot \bar{D}$$

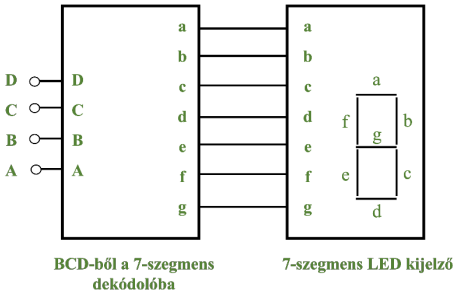
$$f = A + \bar{C} \cdot \bar{D} + B \cdot \bar{C} + B \cdot \bar{D} \qquad g = A + B \cdot \bar{C} + \bar{B} \cdot C + C \cdot \bar{D}$$

6.18. ábra. *A 7 szegmenses kijelző dekódolójának Karnaugh-táblázatokkal és -módszerrel egyszerűsített kimeneti függvényei*

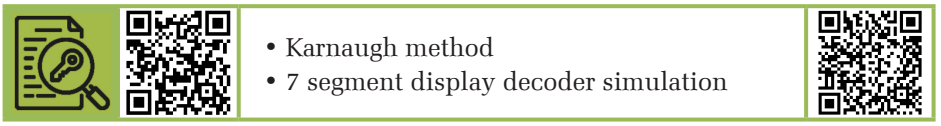


6.19. ábra. *A 7 szegmenses kijelző dekódolójának kapcsolási rajza*

Eredményképpen egy olyan kombinációs, logikai áramkört kapunk, amely a bemeneteire kapcsolt bináris számkódokat emberi felhasználó számára látható és értelmezhető módon képes megjeleníteni. Az így kialakított áramkört egy funkcionális tömbben absztraktizálva így ábrázolhatnánk (6.20. ábra):

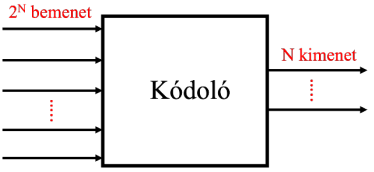


6.20. ábra. *A 7 szegmenses kijelző dekódolójának és a hozzá kapcsolt kijelzőnek rendszer szintű ábrázolása*



6.4.3. Bináris kódoló

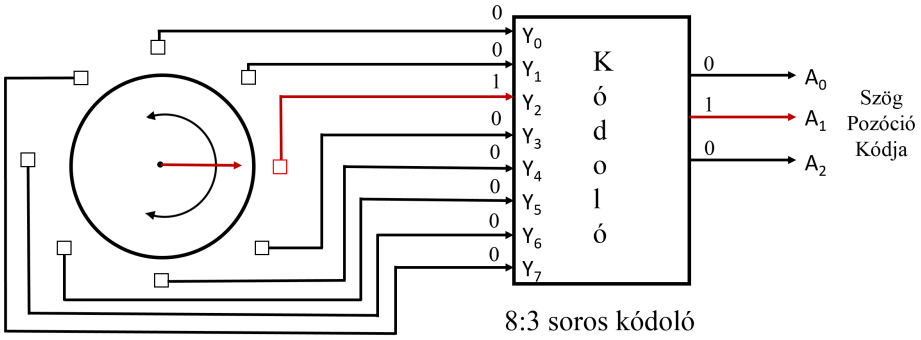
A dekódoló áramkör ellenpárjaként KÓDOLÓ áramköröket használnak számrendszerbeli átalakításoknál (decimálisból bináris vagy BCD-kódba) vagy egy jelcsoport elsőbbséget élvező jelének kiválasztásához (prioritáskódoló). A kódoló áramkörök a bemeneteiken megjelenő digitális kódokat valamilyen másik formába alakítják át – kódolják.



6.21. ábra. *A kódoló áramkör tömbvázlata a bemenetek és kimenetek száma közötti összefüggés hangsúlyozásával*

Az egyszerű kódoló a sorszámozott egyedi bemeneteket átkódolja a sorszám bináris vagy BCD-kódjává, a prioritáskódoló több, egyidejűleg 1-es értékű bemenete közül képes a legnagyobb helyértékűnek megfelelő bináris kódot (számot) generálni. Általánosan elmondható, hogy a 2^N-diken bemenettel rendelkező kódoló áramkörnek N kimenete lehet (6.21. ábra).

Képzeljünk el egy forgó tengelyre szerelt apró mágnest, amely egy mágneses pozíció szenzor fölött tud elfordulni. A pozíció szenzor 8 mágneses térerősség érzékelője sorban egy kódoló áramkör bemeneteire csatlakozik. Ily módon a tengely elfordulását 8 pozíciós bontásban, a kódoló kimenetein megjelenő bináris számként tudjuk követni egy számítástechnikai rendszerben (6.22. ábra).



6.22. ábra. *A kódoló áramkör tömbvázlata a bemenetek és kimenetek száma közötti összefüggés hangsúlyozásával*

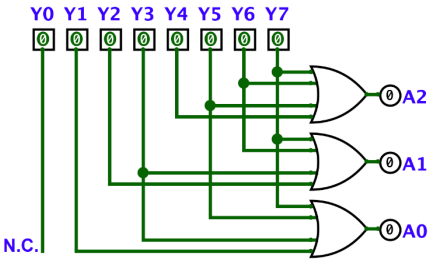
A 8:3-hoz bináris kódoló áramkör igazságtáblázata (6.23. ábra) egy könnyen értelmezhető, sajátos összefüggést ad meg. A fentebb leírt szerkezet mechanikai sajátosságából következően egyszerre csak egy bemenet lehet aktív.

BEMENETEK								KIMENETEK		
Y_7	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0	A_2	A_1	A_0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1

6.23. ábra. *A 8 bemenetű bináris kódoló igazságtáblázata*

Ezeket az összefüggéseket először logikai függvényként írjuk fel: $\mathbf{A_2}=Y_4+Y_5+Y_6+Y_7$; $\mathbf{A_1}=Y_2+Y_3+Y_6+Y_7$; $\mathbf{A_0}=Y_1+Y_3+Y_5+Y_7$; majd ebből felrajzolhatjuk a kombinációs áramkör kapcsolási rajzát. Látható, hogy az $Y_0=1$-re a kimenetek 0-ban állnak, ennek a bemenetnek csak közvetett hatása van a kimenetek

állapotára, amelyeket VAGY függvényekként írunk fel. Ennek alapján vázoljuk fel a 8 bites, bináris kódoló kapcsolási rajzát (6.24. ábra).



6.24. ábra. *A 8 bemenetű bináris kódoló kapcsolási rajza*

- binary encoder
- priority encoder

6.4.4. Bináris többségi kódoló

A TÖBBSÉGI KÓDOLÓ (priority encoder) az előző kapcsolás továbbgondolt változata, felkészülve arra az esetre, amikor több bemenet is egyszerre lehet aktív. Ilyen esetben mindig a legnagyobb helyértékű 1-ben álló bemenet bináris kódja jelenik meg a többségi kódoló kimenetén. Ilyen helyzet könnyen előállhat, amikor egy számítástechnikai rendszer több, egymástól független bemeneti forrást, például megszakításkérést (lásd egy későbbi fejezetben kifejtve) figyel, és ezeket fontossági (prioritás) sorrendben kívánja kezelni. A következő példa az egyszerűség elvét követve a 4:2-höz többségi kódoló kialakítását mutatja be. A négy bemenet közül az Y_3 a legnagyobb, míg az Y_0 a legkisebb prioritású.

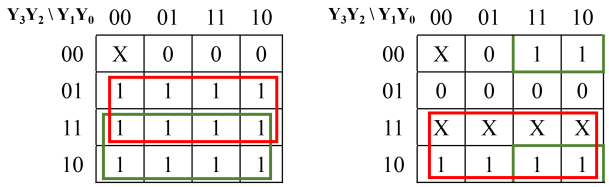
Az igazságtáblázat (6.25. ábra) feltöltése az egyszerű, 8:3-hoz bináris kódolóhoz képest megváltozik, figyelembe veszi az egy időben aktív bemeneteket is, de a kimenetek minden esetben CSAK a legnagyobb helyértékű bemenet kódját fogják megadni. A táblázat kitöltésekor észrevehető, hogy abban az esetben, ha minden bemenet 0-ban áll, nem lehet egyértelműen meghatározni a kimenetek állapotát. Ezért bevezetünk egy V („valid” – érvényes) segédkimenetet, amely ebben az esetben 0, minden más esetben 1, vagyis a kimenetek állapota csak akkor érvényes, ha legalább egy bemenet nem 0.

Mivel a legnagyobb helyértékű, aktív bemenet alatti helyértékek is lehetnek 1-es állapotban, olyan logikai függvényt fogunk kapni, amelyben egyes bemenetek többször is szerepelnek: $\mathbf{A_1}=Y_3+Y_2+Y_1+Y_0+Y_2+Y_1+Y_0$ és $\mathbf{A_0}=Y_3+Y_2+Y_1+Y_0+Y_1+Y_0$. A megoldáshoz egyszerűsítésre van szükségünk, ezért ismét a Karnaugh-diagramot

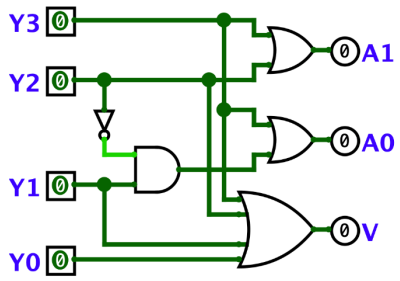
(6.26. ábra) alkalmazzuk és minél nagyobb csoportokat keresünk a táblázatokban (figyelem: a bemeneti állapotok Gray-kód szerint felírva).

BEMENETEK				KIMENETEK		
Y3	Y2	Y1	Y0	A1	A0	V
0	0	0	0	x	x	0
0	0	0	1	0	0	1
0	0	1	X	0	1	1
0	1	X	X	1	0	1
1	X	X	X	1	1	1

6.25. ábra. *A többségi kódoló igazságtáblázata*



6.26. ábra. *A többségi kódoló Karnaugh-diagramjai*



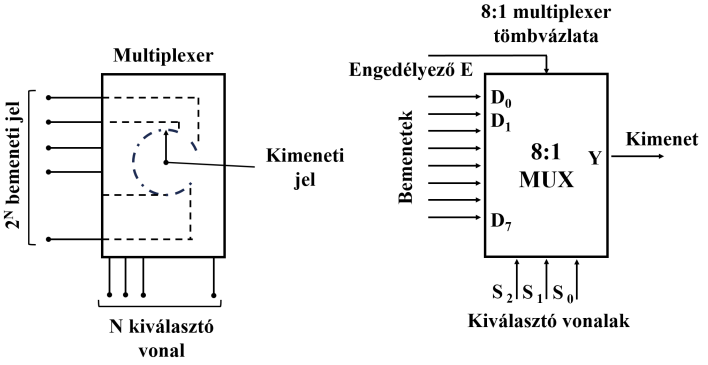
6.27. ábra. *A többségi kódoló kapcsolási rajza*

Az egyszerűsített kimenetek így alakulnak: $\mathbf{A_1}=Y_3+Y_2$ és $\mathbf{A_0}=Y_3+\overline{Y_2}\cdot Y_1$. Az ebből felrajzolható kapcsolási rajz a bemenetek érvényességét szavatoló kimenettel (V) kiegészítve a 6.27. ábra szerint alakul.

6.4.5. Digitális multiplexer

Vizsgáljuk meg egy újabb, a számítástechnikában nélkülözhetetlen logikai áramkör, a MULTIPLEXER felépítését is. A multiplexer adatútválasztó funkciót

valósít meg, egy vagy több bináris értékből álló kód határozza meg, hogy adatbemenetei közül melyik haladhat tovább a multiplexer kimenete felé. Elektromechanikus megfelelője egy körkapcsoló lehetne (6.28. ábra).



6.28. ábra. *A multiplexer elvi megfelelője egy többpólusú körkapcsoló*

Példának válasszunk egy 4:1-hez multiplexert és írjuk fel az igazságtáblázatát (6.29. ábra). Jelöljük S_0, S_1-gyel a kiválasztó bemeneteket, D_0, D_1, D_2 és D_3-mal az adatbemeneteket, Y-nal a kimenetet. A táblázatban X-szel jelöljük az adott állapotban nem számító bemeneteket.

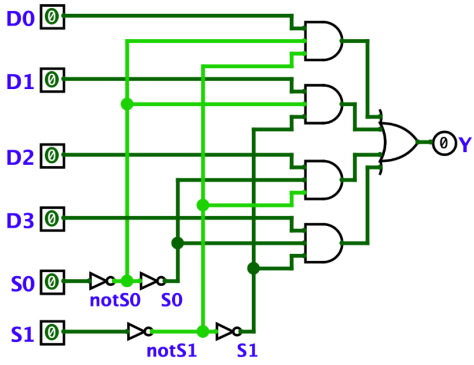
A táblázat alapján fel tudjuk írni a 4:1-hez multiplexer logikai függvényét: $Y = \overline{S0} \cdot \overline{S1} \cdot D0 + \overline{S0} \cdot S1 \cdot D1 + S0 \cdot \overline{S1} \cdot D2 + S0 \cdot S1 \cdot D3$. Mivel nincsenek ismétlődő logikai összefüggések, máris rátérhetünk a logikai kapukkal való megvalósításra.

Megfigyelhető, hogy a kiválasztó bemenetek jelei (S_0, S_1) szerepelnek direkt és tagadott formában is. A megvalósítást a két bemeneti vonal direkt és tagadott formáinak kialakításával kezdjük, ez utóbbiakhoz TAGADÓ kapukat használva.

S0	S1	D0	D1	D2	D3	Y
0	0	0	X	X	X	0
0	0	1	X	X	X	1
0	1	X	0	X	X	0
0	1	X	1	X	X	1
1	0	X	X	0	X	0
1	0	X	X	1	X	1
1	1	X	X	X	0	0
1	1	X	X	X	1	1

6.29. ábra. *A 4:1-hez multiplexer igazságtáblázata*

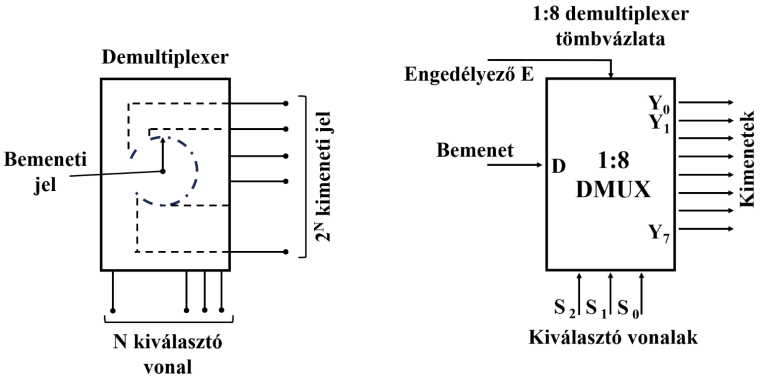
Ezt követően az egyes függvényrészeket három bemenetű **és** kapukkal valósítjuk meg, majd ezek kimenetét egy négybemenetű **vagy** kapuban fogjuk össze (6.30. ábra).



6.30. ábra. *A 4:1-hez multiplexer kapcsolási rajza*

6.4.6. Digitális demultiplexer

A multiplexer funkciójának ellenpárjaként, egy bemeneti adatvonal kiválasztott adatúton történő továbbítását, egy vagy több bináris értékből álló kód alapján, a DEMULTIPLEXER (DMUX) végzi (6.31. ábra). Elektromechanikus megfelelője, a multiplexerhez hasonlóan, szintén egy körkapcsoló lehetne. Példának válasszunk egy 1:4-hez demultiplexert, és írjuk fel az igazságtáblázatát (6.32. ábra).



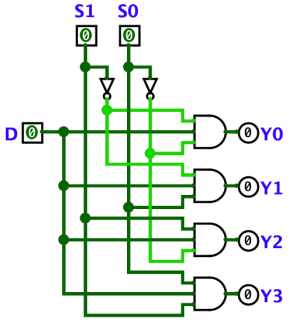
6.31. ábra. *A demultiplexer elvi megfelelője egy többpólusú körkapcsoló*

Adat bemenet	Kiválasztott Bemenetek		Kimenetek			
D	S_1	S_2	Y_3	Y_2	Y_1	Y_0
D	0	0	0	0	0	D
D	0	1	0	0	D	0
D	1	0	0	D	0	0
D	1	1	D	0	0	0

6.32. ábra. *Az 1:4-hez demultiplexer igazságtáblázata*

A táblázat alapján fel tudjuk írni az 1:4-hez demultiplexer kimeneteinek logikai függvényeit. A kimenetek mindig az S_0, S_1 jelektől függenek: $Y_0 = \overline{S_0}\cdot\overline{S_1}\cdot D$, $Y_1 = \overline{S_0}\cdot S_1\cdot D$, $Y_2 = S_0\cdot\overline{S_1}\cdot D$, $Y_3 = S_0\cdot S_1\cdot D$. Mivel nincsenek ismétlődő logikai összefüggések, máris rátérhetünk a logikai kapukkal való megvalósításra (6.33. ábra). Megfigyelhető, hogy a kiválasztó bemenetek jelei (S_0, S_1) szerepelnek direkt és tagadott formában is. A megvalósítást a két bemeneti vonal direkt és tagadott formáinak kialakításával kezdjük, ez utóbbiakhoz TAGADÓ kapukat használva. Ezt követően az egyes függvényrészeket hárombemenetű ÉS kapukkal valósítjuk meg.

A multiplexer és a demultiplexer áramkörök a számítástechnikai eszközök, processzorok, periféria áramkörök belső alkatrészeiként, adatút-kiválasztó funkciójuk mellett akár szinkron soros adatátviteli eszközként is felhasználhatók. Ehhez az alkalmazási példához azonban szükségünk van még néhány alapvető szekvenciális áramkör ismeretére is, amilyen például a bináris számláló áramkör és a D tároló. Ezekkel később foglalkozunk.



6.33. ábra. *Az 1:4-hez demultiplexer kapcsolási rajza*

- multiplexer
- demultiplexer

6.4.7. Bináris összeadó áramkörök

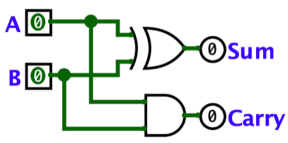
A bemeneti jelek között kombinációs logikai áramkörökkel nemcsak logikai, hanem aritmetikai összefüggéseket is képezhetünk. Például összeadhatunk vagy kivonhatunk két bináris értéket. Kezdjük az összeadással. Ehhez ismernünk kell a tízesből bináris számrendszerbe való átalakítási eljárást, valamint a bináris összeadás módszerét.

A példában 172-höz adunk hozzá 186-ot, és 358-at kapunk eredménynek, átvitellel a műveletsor legnagyobb helyértékén felül. Leegyszerűsítve azt látjuk, hogy a 0 + 0 = 0, 0 + 1 = 1, 1 + 0 = 1, 1 + 1 = 0 (ÉS keletkezik egy átviteli érték: 1) műveletek a **KIZÁRÓ VAGY** kapu igazságtáblázatának megfelelően történnek. A keletkező átvitel ellenőrzéséhez egy **ÉS** kapu igazságtáblázata szerint kéne eljárni. A két kimenetet meghatározó logikai függvény a következő:

Sum= A ⊕ B és Carry=A·B.

Ebből következően felrajzolhatjuk a FÉLÖSSZEADÓ ÁRAMKÖRT (6.34. ábra). Egy ilyen áramkör túlcsordulást hozhat létre, de fogadni nem tud.

B	A	SUM	CARRY
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



6.34. ábra. *A bináris összeadás egy helyértékre vonatkoztatott igazságtáblázata és a félösszeadó áramkör kapcsolási rajza*

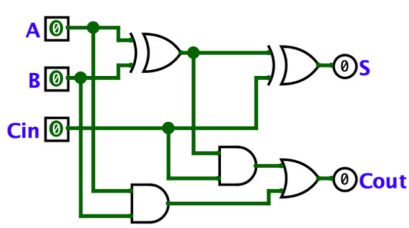
A több biten ábrázolt számértékek összeadásához viszont szükség van arra, hogy az előző helyértéktől származó túlcsordulást be lehessen számolni az összeadás részeredményébe, ezért a félösszeadó áramkört ki kell bővíteni úgynevezett TELJES ÖSSZEADÓ áramkörré (6.35. ábra). Az igazságtáblázatot kiegészítjük arra az esetre, amikor a fogadott túlcsordulás (C_{IN}) 1-es (a 6.35. ábra piros kerettel kiemelt része).

Ebben az esetben a kimeneteket leíró logikai függvényeket így írjuk fel:

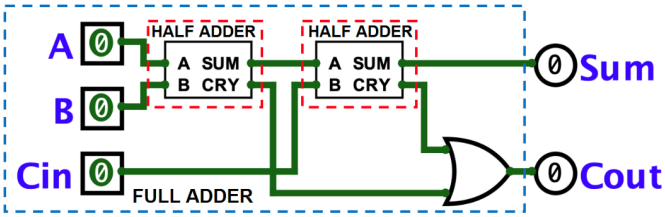
Sum= (A ⊕ B) ⊕ C és C_{OUT} = (A·B) + C_{IN}·(A ⊕ B)

A túlcsordulás kimenet az első félösszeadó cella bemenetei közötti **ÉS** összefüggés **VAGY** a második félösszeadó cella túlcsordulás értéke, ami az első cella részeredménye **ÉS** a bemeneti, kapott túlcsordulás eredménye. Ebből következően a két félösszeadó cellából létrehozható egy teljes összeadó áramkör (6.36. ábra).

Cin	B	A	SUM	Cout
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

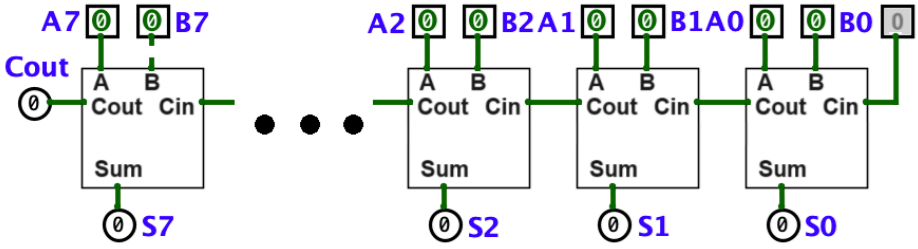


6.35. ábra. *Teljes összeadó áramkör egy helyértékre vonatkoztatott igazságtáblázata és kapcsolási rajza*



6.36. ábra. *Két félösszeadóból kiépíthető egy teljes összeadó modul*

Teljes összeadók sorozatából tetszőleges szóhosszúságú összeadó áramkör építhető ki (6.37. ábra). A legkisebb helyértéket képviselő modul túlcsordulás bemenetén állandó 0-ás logikai értéket biztosítunk.

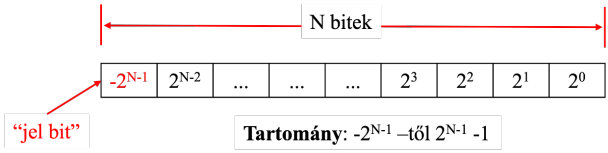


6.37. ábra. *8 bites összeadó modul tömbvázlata*

Utolsó kombinációs áramkörként vizsgáljuk meg egy 8 bites, teljes ÖSSZEADÓ-KIVONÓ MODUL megvalósítását és működését. Mivel minden vizsgálatunk a számítástechnikai rendszerek felől közelít a témához, egy olyan megoldást keresünk, amely áramköri szempontból egységes, egyszerű kiépítést jelent. Felvethető, hogy két pozitív számot úgy is ki lehet vonni egymásból, hogy a kivonandó pozitív számot negatív számmá alakítjuk, majd a két számot összeadjuk, például:

$96 - 12 = 96 + (-12) = 84$

Ugyanez a művelet a számok bináris formájára vonatkoztatva is elvégezhető, ha sikerül a kivonandót negatív számmá alakítani. Erre szolgál a 2-es komplemens formába való alakítás módszere. Gondoljunk arra, hogy egy számítástechnikai rendszerben adott szóhosszúságok tárolására alkalmas rekeszek vagy közvetítésére alkalmas adatutak vannak, és nincs lehetőség egy számérték előjelét a szóhosszon kívül tárolni vagy jelezni, csakis az adott kereten belül. Ha például 8 bit áll rendelkezésünkre, amelyen 2^N-en, azaz 2^8=256, vagyis 0-tól 255-ig tudunk pozitív egész számokat ábrázolni, akkor ugyanekkora szóhosszon -2^{N-1}-től 2^{N-1}-1 között tudunk negatív és pozitív egész számokat ábrázolni (6.38. ábra).



6.38. ábra. *N biten való negatívtól pozitívig terjedő számábrázolás*

Azért van 2^{N-1}-1 pozitív értékünk, mert a pozitív oldali számsorban egy „semleges helyet" elfoglal a 0-s szám. Mivel a negatív számok legnagyobb helyértékű bitje mindig 1-es lesz, egy számítástechnikai rendszerben figyelhetjük ezt a helyértéket és megállapíthatjuk belőle, hogy adott esetben egy számérték pozitív vagy negatív.

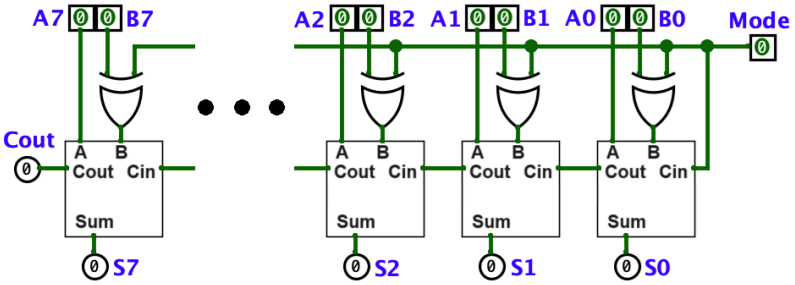
2-es komplemensbe való alakítás az adott bináris számérték minden számjegyének tagadott formában való felírásával kezdődik, ezt nevezzük 1-es komplemens formának. Ehhez a formához adunk hozzá 1-et. Az eredmény a kiindulási pozitív, egész számérték negatív megfelelője. Ezt a bináris összeadás szabályai szerint hozzáadjuk a csökkenteni kívánt számértékhez.

```
  0 0 0 0 1 1 0 0 (+12)
  1 1 1 1 0 0 1 1 (1-es komplemens)          0 1 1 0 0 0 0 0 (+96)
+ 0 0 0 0 0 0 0 1 (+1)                     + 1 1 1 1 0 1 0 0 (- 12)
-------------------------------            -------------------------------
 1 1 1 1 0 1 0 0 (2-es komplemens)           0 1 0 1 0 1 0 0 (+84)
```

Amennyiben a csökkenteni kívánt számérték kisebb, mint a kivonandó, az eredmény negatív szám lesz. De ugyanez az eljárás alkalmazható két negatív számérték összeadása vagy kivonása esetében is.

```
  0 0 1 0 1 1 0 1 (+45)          1 1 0 1 1 0 1 1 (-37)
+ 1 0 1 1 1 1 1 0 (- 66)      + 1 1 1 0 0 0 1 1 (-29)
--------------------------     --------------------------
  1 1 1 0 1 0 1 1 (-21)          1 0 1 1 1 1 1 0 (-66)
```

A teljes ÖSSZEADÓ-KIVONÓ MODUL megvalósításhoz szükségünk van egy 2-es komplemens képző logikai áramköri részre, amely a művelet kódjától függően csak kivonás művelet esetében aktív, és csak az egyik bemeneti számértéket alakítja át. Keresnünk kell egy olyan logikai függvényt, amely az egyik bemeneti értékét a másik bemenet 0-ás állapota esetén változatlanul továbbengedi, 1-es állapota esetén meg negálja. Az alap logikai függvények közül csak egy ilyent találunk, a jól ismert KIZÁRÓ-VAGY (XOR) kaput. Ezt a kaput használjuk fel a B bemeneti értékek vezérlőjelhez kötött (0-ás, ha összeadás, 1-es, ha kivonás művelet) negálásához. A 2-es komplemens képzéshez szükséges 1-es érték hozzáadását a negált értékhez az első teljes összeadó túlcsordulás bemenetére bevezetett vezérlőjellel valósítjuk meg.



6.39. ábra. *8 bites összeadó-kivonó modul tömbvázlata*

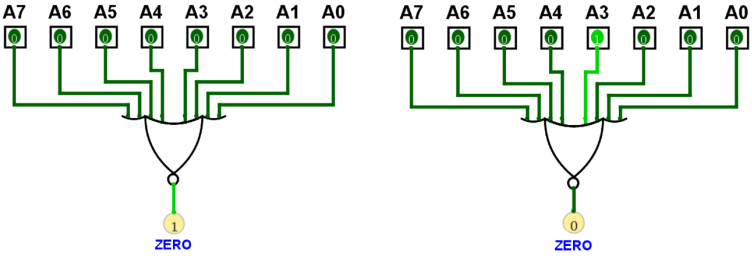
Az ily módon kialakuló áramkör (6.39. ábra) egymás után kapcsolt (túlcsordulás kimenet a következő cella kölcsön bemenetéhez csatolva) teljes összeadókból egy 8 bites összeadó-kivonó áramkört képez.

- binary adder
- binary adder subtractor circuit

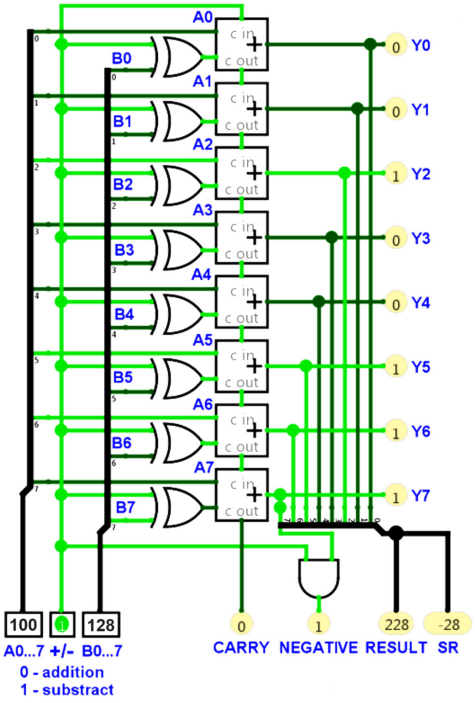
6.4.8. Nulla érték, túlcsordulás, negatív eredmény figyelése

Az aritmetikai műveletek eredményeit a számszerű forma mellett más szempontok szerint is vizsgálhatjuk.

Figyelhetjük, hogy egy művelet elvégzése nullát eredményez. Ehhez a TAGADOTT VAGY függvényt és kaput használjuk (6.40. ábra). A kapu bemeneteinek számát a figyelt értékkel egyezően határozzuk meg (ha ilyen kapu nem áll rendelkezésünkre, mindig kombinálhatunk több, kevesebb bemenetű kapuból). Amennyiben a bemeneti érték minden bitje 0, az áramkörünk kimenete 1-es lesz, máskülönben 0 marad. Ez az eredménybit lehet a későbbiekben a processzorok ZÉRÓ jelzőbitje (ZERO Flag).



6.40. ábra. *8 bites nullafigyelő modul kialakítása*



6.41. ábra. *Túlcsordulás és negatív eredmény figyelése*

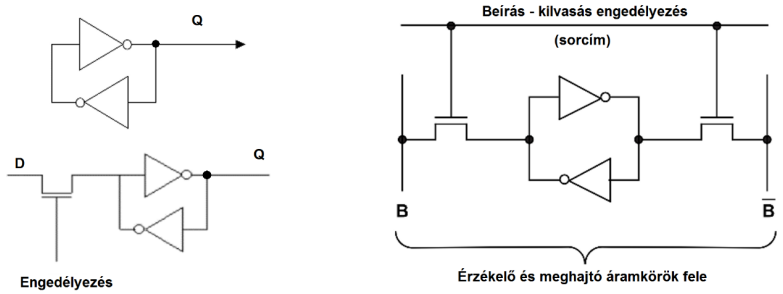
A túlcsordulás figyelése az összeadó-kivonó áramkör legnagyobb helyértékű cellájának a túlcsordulás kimenetén (C_{out} – 6.39. ábra) történik. Amennyiben a művelet elvégzése nagyobb helyértéket generál, mint az adott 8 biten ábrázolható szám, ez a bit 1 lesz. Ez az eredménybit lehet a későbbiekben a processzorok **TÚLCSORDULÁS** jelzőbitje (CARRY Flag).

A negatív eredmény figyelése a műveletvezérlő jel (+/-) és az eredmény legnagyobb helyértékű bitjének (Y_7) egyetlen **ÉS** kapuba való összevonásával is megoldható. Ez az eredménybit lehet a későbbiekben a processzorok **NEGATÍV** jelzőbitje (NEGATIVE Flag). A 6.41. ábra abban az állapotban mutatja a kapcsolást, amikor 100-ból vonunk ki 128-at. Az eredmény, előjelesen értelmezve, -28, a **NEGATÍV** jelzőbit 1-ben áll.

6.5. Sorrendi vagy szekvenciális áramkörök

Számítástechnikai rendszerek fontos építőelemeként tekintünk a SORRENDI vagy SZEKVENCIÁLIS alapáramkörökre, mivel ezekből építhető elektronikus, digitális adattárolásra vagy adatmozgatásra alkalmas regiszter vagy számláló. A sorrendi áramkörök fő tulajdonsága, hogy kimeneteiken képesek megőrizni egy előzőleg létező bemenet által létrehozott állapotot. Erre a kombinációs áramkörök nem képesek. Az alap logikai kapukból kiépíthető kapcsolás sajátossága, hogy a kimenetek visszacsatolódnak az áramkört alkotó kapuk ellentétes bemeneteire.

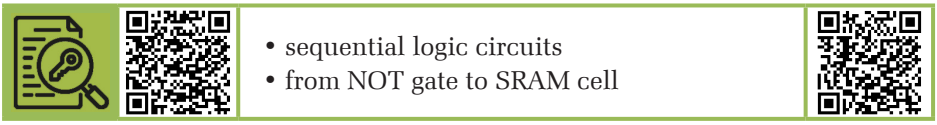
A vizsgálódást kezdjük az egyszerű tagadó kapuk egymásnak fordított összekapcsolásával, bár ez csak elméleti megközelítésre szolgál (6.42. ábra bal oldali része).



6.42. ábra. *Tárolócella NOT kapukból kialakított változatai az elméleti megközelítéstől a SRAM celláig*

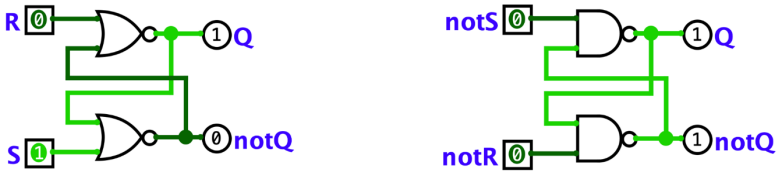
Kijelenthető, hogy a Q ponton beállított logikai állapot a kapuk tápellátásának megszűntéig vagy a Q pont állapotváltozásáig változatlan marad. Az elméleti verziót kiegészítve egy FET-tranzisztorral, egy olyan cellát kapunk, melynek belső

állapota a tranzisztort megvezérelve (Enable) a D bemeneti pont felől megváltoztatható. Ezt tovább bővítve a Q kimenet felé egy SRAM cellát (6.42. ábra jobb oldali része) kapunk, melynek részletesebb leírását egy későbbi fejezetben találjuk meg.



6.5.1. RS tárolók

Vegyük további példának az RS ELEMENTÁRIS TÁROLÓT (Reset-Set latch). Ezt a kapcsolást számos szekvenciális áramkör építőelemeként használjuk majd. NEM-ÉS (NAND) vagy NEM-VAGY (NOR) kapuk sajátos módon való összekapcsolásából alakítható ki az alábbi ábráknak megfelelően:



6.43. ábra. *RS tárolócella NOR és NAND kapukból kialakított változatai*

Az áramkörnek (6.43. ábra) két bemenete van, az S (Set) – a tároló tartalmának 1-be állítására és az R (Reset) – a tároló tartalmának 0-ba állítására. A két kimenet egymásnak logikai ellenpárjaként működik, amikor Q=1, $\overline{Q}$ =0 és fordítva. Alkalmazástól függően csak egyik vagy mindkét kimenetet használják.

A NOR RS tároló esetében induljunk ki az S=0, R=0, Q=0 állapotból. Mivel a Q vissza van vezetve az alsó NOR kapuba, annak mindkét bemenete 0, így a kimenete /Q=1. Az 1 érték vissza van vezetve a felső kapuba, amelynek a bemenetei most 1 és 0-ban állnak, ebből Q=0 következik. Ez egy stabil, állandó állapot, a tároló 0-ás állapota. Ha most S-et 1-be állítjuk, Q=1-et kapunk, ami szintén egy stabil állapot, akkor is, ha S=0-ra áll vissza, ilyenkor valósul meg a TÁROLÁS.

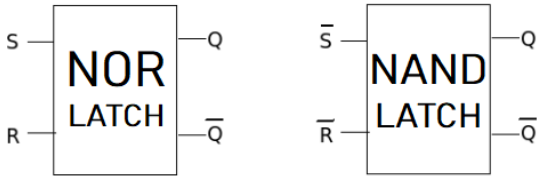
Most vizsgáljuk meg az S=0, R=0 és Q=1 kiindulási esetet. Így az alsó kapunak a két bemenete 0 és 1-es, a kimenete $\overline{Q}$ =0, amit visszavezetünk a felső kapuba, amelynek a bemenetei most 0 és 0-ban állnak, ebből Q=1 következik. Ez az állapot a tároló 1-es állapota, amely szintén stabil. Ha most R-et 1-be állítjuk, Q=0-át kapunk, ami szintén egy stabil állapot, akkor is, ha R=0-ra áll vissza és a cella ismét TÁROL.

S	R	Q	$\overline{Q}$	Megjegyzések
0	0	T	T	TÁROLÁS
0	1	0	1	RESET
1	0	1	0	SET
1	1	0	1	ÉRVÉNYTELEN

$\overline{S}$	$\overline{R}$	Q	$\overline{Q}$	Megjegyzések
0	0	1	1	ÉRVÉNYTELEN
0	1	1	0	SET
1	0	0	1	RESET
1	1	T	T	TÁROLÁS

6.44. ábra. *RS tároló NOR és NAND változatainak igazságtáblázata*

Az igazságtáblázatok (6.44. ábra) a kapuk működésének megfelelően írhatók fel (tekintsük át újra a NAND és NOR kapuk igazságtáblázatát is a 6. fejezet első részében). A NOR kapus megvalósítás az R=S=0 állapotban tárol, a NAND változat az $\overline{R}=\overline{S}=1$ állapotban. A cellatípusok rendszerszintű ábrázolásához a következő jelöléseket használhatjuk (6.45. ábra):



6.45. ábra. *RS tárolócella NOR és NAND változatainak szimbólumai*

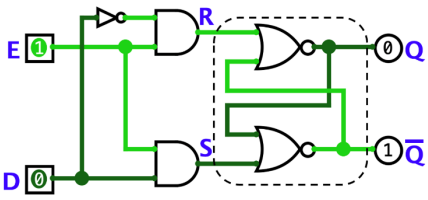
Mindkét típus esetében érvényes, hogy csak akkor működik jól az áramkör, ha az R és S bemeneteik olyan állapotokban vannak, amelyek ellentétes értékű kimeneteket adnak, Q soha nem lehet egyenlő $\overline{Q}$-val. A NOR változat esetében R=S=1, a NAND változat esetében $\overline{R}=\overline{S}=0$ állapotok az áramköröket oszcillációra késztetik, melynek frekvenciája a kapuk átfutási sebességétől függ. Ebben az állapotban a tárolócellák kimeneti értéke ÉRVÉNYTELEN, ezért biztosítani kell, hogy a bemenetek soha ne kerüljenek ilyen állapotba.

- RS flip flop
- D flip flop

6.5.2. Időzített RS cella, a D tároló

Az R=S kedvezőtlen állapotát a bemenetek közé illesztett tagadó kapuval zárhatjuk ki, így egyetlen „adat” bemenetünk lesz, míg a beírás műveletet az eddigi R és S bemenetek **és** kapus kondicionálásával, egy engedélyező jellel (E - Enable)

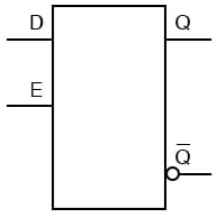
oldhatjuk meg, ahogy a 6.46. ábra mutatja. Az így kapott új szekvenciális áramkör az IDŐZÍTETT D TÁROLÓ.



E	D	Q	$\overline{Q}$
0	0	tárol	tárol
0	1	tárol	tárol
1	0	0	1
1	1	1	0

6.46. ábra. *Időzített D tároló és igazságtáblázata (RS mag kiemelve)*

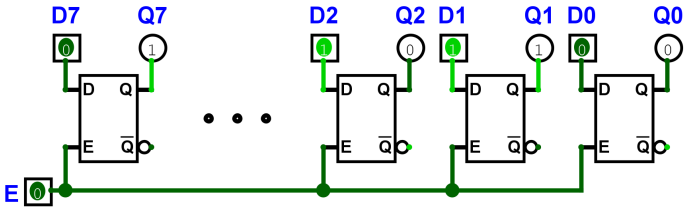
Az igazságtáblázatból (6.46. ábra) látható, hogy az RS tárolónál tapasztalt ÉRVÉNYTELEN eset itt nem következik be. SZINTVEZÉRELT üzemmódban a D bemenet 0-ás vagy 1-es állapota csak akkor halad tovább a cella belső kapui felé, amikor az engedélyező jel állapota E=1-gyel. Ellenkező esetben a kimenetek értékei nem változnak, a cella TÁROLJA az előzőleg beállított értéket. Ez a tulajdonsága az időzített (engedélyező jelhez kötött) D tárolót egy univerzális alkatrésszé teszi. Szimbolikus ábrázolása az alábbi ábrán látható (6.47. ábra):



6.47. ábra. *Időzített D tároló szimbóluma*

6.5.3. Adattároló regiszter

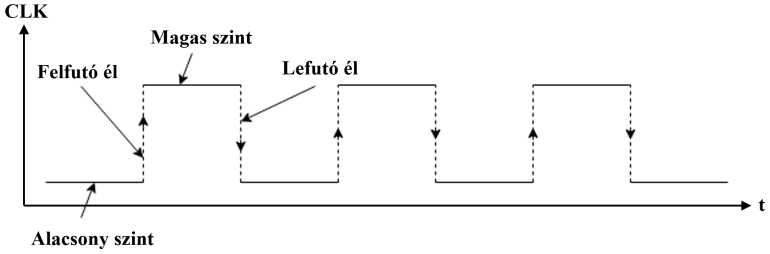
Abból kiindulva, hogy egy időzített D tároló 1 bit információ tárolására alkalmas, az összeadó és kivonó áramkörök mintájára ezeket is hálózatba kapcsolhatjuk, így 8 vagy akár több bites adattárolókat, úgynevezett REGISZTEREKET nyerhetünk. Az alábbi kapcsolási rajzon (6.48. ábra) egy 8 bites regisztert alakítunk ki 8 darab időzített D tárolóból, közös vonalra kapcsolva az engedélyező (E- Enable) jeleiket.



6.48. ábra. *8 bites, időzített D tárolókból kialakított regiszter*

6.5.4. Az órajel állapotai és eseményei

További vizsgálódásaink során ki kell térnünk a számítástechnikai rendszerek működéséhez elengedhetetlen ÓRAJEL szerepére. Láthattuk, hogy a regiszter áramkörök esetében a cellába írást engedélyező jellel (E) kondicionálják. Ennek a jelnek adott logikai állapota határozza meg, hogy a beírás megtörténik vagy nem. Ahhoz, hogy a számítástechnikai rendszerben a hasonló áramkörök egymással szinkronban tudjanak működni, feltétel, hogy egy meghatározott időpontban történjen meg a bemenet „mintavételezése”, illetve tárolása. Az időpontot az ÓRAJEL állapotváltozása határozhatja meg. Egy adott jelvonalon észlelhető stabil 1-es vagy 0-ás értéket ÁLLAPOTNAK tekintjük, az állapotok közötti változást ESEMÉNYNEK. Két ilyen eseményt határozunk meg, a 0-ból 1-be váltást FELFUTÓ ÉLNEK, az 1-ből 0-ba váltást LEFUTÓ ÉLNEK nevezzük (6.49. ábra).



6.49. ábra. *Órajel magas és alacsony állapotai, felfutó és lefutó élei (eseményei)*

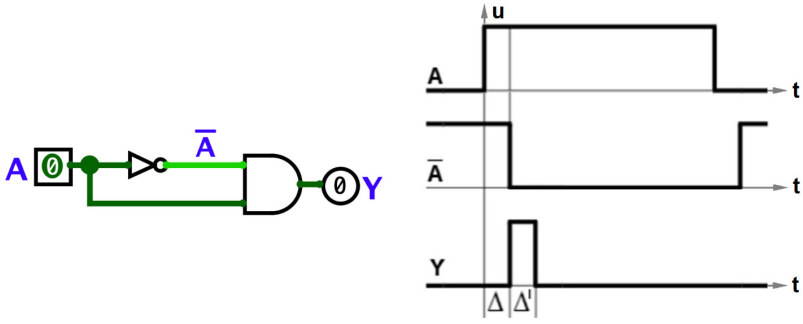
Az órajel élváltásaihoz köthető eseményekkel szinkronban működő szekvenciális logikai áramköröket bistabil billenőköröknek vagy flip-flop áramköröknek is nevezzük. Az órajel stabil állapotai nem okoznak állapotváltást ezekben az áramkörökben. Következésképpen a flip-flop áramkörök ÉLVEZÉRELTEK.

6.5.5. Pulzusgenerátor

Rövid, pár nano- vagy pár száz pikoszekundumos impulzusokat hozhatunk létre egy olyan logikai áramkörrel, amelynek két bemenetét egymáshoz képest késleltetjük. A jelkésleltetést az egyik bemeneti vonalba kapcsolt, tagadó logikai kapu átfutási ideje is létrehozhatja. Az így kapott áramkör a PULZUSGENERÁTOR (a 6.50. ábra bal oldalán). Működését a legjobban úgy érthetjük meg, ha adott időegység alatt át tudjuk tekinteni az áramkör különböző pontjain létrejövő feszültségszinteket.

Grafikusan, egymás alatt ábrázolva a különböző jelek állapotait és éleit, láthatóvá válik ezek időbeni viszonya. Ezt az ábrázolásmódot nevezzük IDŐDIAGRAMNAK (lásd a 6.50. ábra jobb oldalán). A grafikonok vízszintes tengelye az idő, függőleges tengelye a feszültség. Bejárási irányuk az idő előrehaladtát követve balról jobbra történik. A vékony függőleges vonalak az események időbeni viszonyítását segítik.

A példában egy ÉS meg egy TAGADÓ kapu kombinációjaként kapott pulzusgenerátort vizsgálunk. Az A bemenet kezdetben 0-ban áll, a /A köztes jel ennek megfelelően 1-ben. Innen indítjuk a megfigyelést és a jelek rögzítését az időidiagramban.



6.50. ábra. *Pulzusgenerátor bemeneti (A), köztes (Ā) és kimeneti (Y) jelállapotainak idődiagramja a késleltetési idők (Δ, Δ') feltüntetésével*

Amikor A 1-be vált, Ā még mindig 1-ben van. A tagadó kapu MOSFET tranzisztorainak szükségük van egy kis időre, amíg a kapuelektróda (gate) töltéshordozói kiürülnek, a tranzisztorok lezárnak és a kimenet 0-ra változik. Ez alatt a Δ (delta) idő alatt az ÉS kapu mindkét bemenete 1-ben áll, következésképpen az Y kimenet is 1-es lesz, de ennek hatása csak újabb Δ' (delta jelzett) idő után jelentkezik, ami az ÉS kapu átfutási idejét jelenti. Miután az ÉS kapu bemenetei stabilizálódnak, kimenete ismét állapotot vált, ezúttal 0-ba, létrehozva egy igen rövid impulzust a kimeneten. Az újabb impulzus létrehozásához az A bemenet újabb állapotváltás-ciklusára van szükség (0-ba, majd újra 1-be). Az átfutási időket befolyásolja a kapuk közötti vezetékek

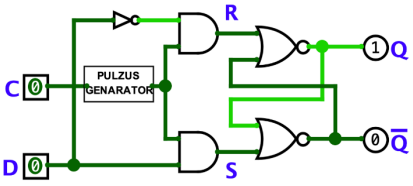
hossza és egyéb jelenségek is, de ezektől most eltekintünk. Az impulzus hosszát, a Δ mértékét több tagadó kapu sorba kapcsolásával tudjuk beállítani.

Az ÉS kaput egy TAGADOTT VAGY kapura cserélve az A lefutó élére kapjuk a kimeneti impulzust.

A számítástechnikai rendszerek tervezésekor mindig figyelembe kell venni a kapuk átfutási idejét, és gondosan ki kell egyenlíteni az így keletkezett jelkésleltetést.

6.5.6. Élvezérelt D tároló

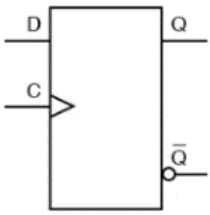
A pulzusgenerátort összekombinálva egy időzített D tárolóval egy ÉLVEZÉRELT D TÁROLÓT vagy más néven D flip-flopot kapunk. Az áramkör az órajel (CLK) felfutó élére fog aktiválódni (kiszámítható késleltetéssel). Ameddig az órajel frekvenciája és ebből következően periódusideje nagyobb, mint az eredő impulzus szélessége, az élvezérelt D tároló szinkronban lesz a felfutó élekkel. Az alábbi, 6.51. ábra az ismerős D tároló áramkört mutatja a pulzusgenerátorral kiegészítve.



C	D	Q	$\overline{Q}$
⌐	0	0	1
⌐	1	1	0
X	0	tárol	tárol
X	1	tárol	tárol

6.51. ábra. *Élvezérelt D tároló kapcsolási rajza és igazságtáblázata*

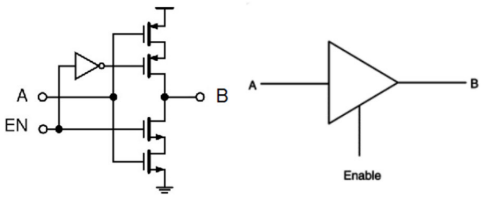
Az igazságtáblázatból látható, hogy a D bemenet 0-ás vagy 1-es állapota – ez az ÉLVEZÉRELT üzemmód – csak akkor halad tovább a cella belső kapui felé, amikor az órajel 0-ból 1-be vált, azaz felfutó él következik. Ellenkező esetben a kimenetek értékei nem változnak. Ez a tulajdonsága az élvezérelt D tárolót egy univerzális, számítástechnikai rendszerek kiépítésében alkalmazható alkatrésszé teszi. Szimbolikus ábrázolását az alábbi, 6.52. ábra mutatja:



6.52. ábra. *Élvezérelt D tároló szimbóluma*

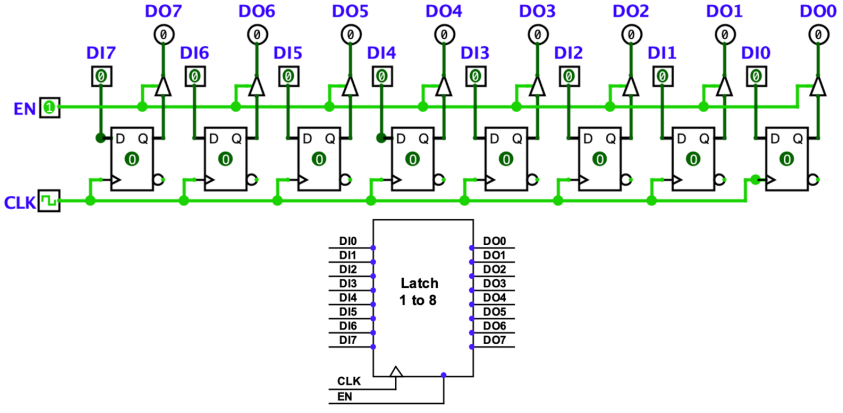
6.5.7. Sínmeghajtók, pufferek

A regisztereket a számítógép-architektúrák vonatkozásában párhuzamos adatutak szétválasztására is használhatjuk. Ilyenkor kimeneteiket le kell választani az adatútról, ezért háromállapotú (three-state) kimeneti meghajtó fokozatokkal, más néven PUFFEREKKEL látjuk el a cellákat. Az ilyen puffert komplementer MOSFET-tranzisztorokból alakítják ki úgy, hogy egy logikai jellel meg lehessen határozni, hogy a puffer kimenete magas impedanciás állapotban legyen, vagy a bemenetén megjelenő digitális jelet továbbítsa. Magas impedanciás állapotot úgy tudunk elérni, ha két MOSFET-tranzisztort úgy kapcsolunk össze, hogy a testdiódáik (body diode) azonos pólusuk irányából érintkezzenek, ilyenkor az egyik MOSFET-forrás (source) elektródja a másik elnyelő (drain) elektródjával kerül közös csomópontra. Ez a csomópont lesz a puffer kimenete is. A következő, 6.53. ábra egy nem invertáló puffer áramkör kapcsolási rajzát és szimbólumát mutatja az igazságtáblázatával kiegészítve. Megfigyelhető, hogy az engedélyező (EN – Enable) jel 0 értéke esetén a kimenet magas impedanciás (Z vagy HZ – high Z) állapotban van, 1-es érték esetén pedig követi a bemenetére kapcsolt jel állapotát.



Enable	A	B
0	0	Z
0	1	Z
1	0	0
1	1	1

6.53. ábra. *Nem invertáló, háromállapotú puffer áramkör kapcsolási rajza, szimbóluma és igazságtáblázata*



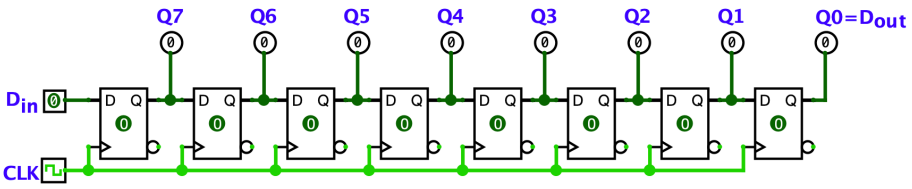
6.54. ábra. *Vezérelhető kimenetű regiszter kapcsolási rajza és szimbóluma*

Gyakorlati megvalósításhoz az élvezérelt D tárolókból álló regiszter áramkört az adott követelménynek megfelelő pufferrel (invertáló vagy nem invertáló) egészítjük ki a kimeneti oldalon.

A 6.54. ábra egy ily módon kiegészített, 8 bites regiszter áramkör kapcsolási rajzát és annak szimbólumát mutatja be. Ezzel a szimbólummal a további fejezetekben gyakran fogunk találkozni.

6.5.8. Soros léptetőregiszter

A regiszterhez hasonló módon csoportosított, élvezérelt D tárolókból, de attól eltérő kapcsolási módban működik a SOROS LÉPTETŐ REGISZTER. Ebben az esetben az egyes élvezérelt D tárolók kimenete (Q_n) a közvetlen szomszéd tároló bemenetéhez (D_{n-1}) kapcsolódik, miközben az órajelbemenetekre minden cella fel van fűzve (6.55. ábra). Az ilyen típusú regiszter az órajel (CLK) felfutó élére a nagy helyértéktől (Q_7) a kisebb felé (Q_0), balról jobbra lépteti a bemenetén megjelenő értékeket. Nyolc órajel alatt a D_{in} bemenetre elsőként vezetett bitérték megjelenik a D_{out} kimeneten.

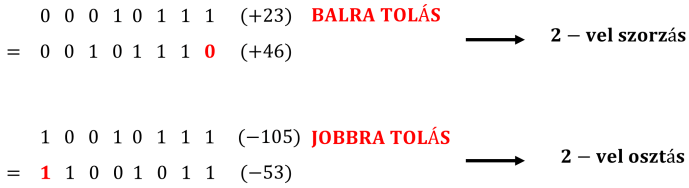


6.55. ábra. *8 bites, élvezérelt D tárolókból kialakított soros bemenetű, jobbra toló, párhuzamos és soros kimenetű léptető regiszter*

6.5.9. Biteltolások

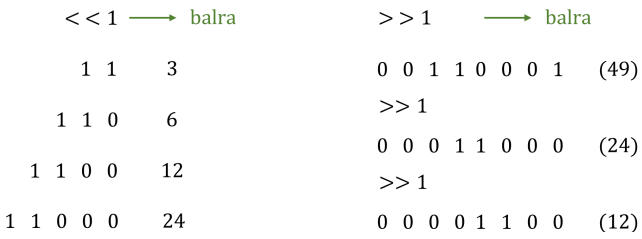
A biteltolások vagy léptetések (bit shift) nem a számértékeken, hanem a bitek sorozatán fejtik ki hatásukat. A biteltolás során a bináris számjegyek jobbra (right shift, vagy balra tolódnak el (right shift, left shift). Mivel a processzor regisztereinek bitszélessége adott, ezért egy vagy több bit ki fog tolódni a regiszter valamelyik végén, a másikon pedig ugyanannyi bit kell belépjen. A biteltolási műveletek közötti különbséget a beléptetett bitek értéke határozza meg.

Aritmetikai eltolás esetén (6.56. ábra) a bármelyik irányba kiléptetett bitek értéke elvész. Balra tolásnál 0-ák lépnek be a számsorba a jobb oldalon. Egy eltolási lépés az eredő számérték kettővel való szorzását eredményezi. Jobbra tolásnál előjeles számérték esetén 1-es érték, előjel nélküli számérték esetén 0-ás mint előjelbit lép be a bal oldalon, megőrizve ezzel az operandus előjelét. Egy eltolási lépés az eredő számérték kettővel való osztását eredményezi.



6.56. ábra. *Aritmetikai, előjelmegtartó jobbra-balra léptetés*

Logikai balra eltolás esetén (6.57. ábra) 0-ák lépnek be a kiléptetett bitek helyére, ezért a logikai és az aritmetikai eltolás bal irányban egyező eredményt ad. A logikai jobbra eltolás esetén viszont eltérés van, hiszen az előjelbit helyett minden esetben nulla értékű bit íródik be a regiszterbe.



6.57. ábra. *Logikai, előjelelhagyó jobbra-balra léptetés*

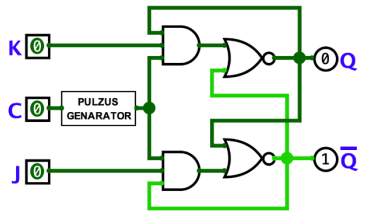
Ezért a logikai eltolást előjel nélküli bináris számoknál, az aritmetikai eltolást előjeles, kettes komplemens alakú bináris számok esetében használják nagyobb gyakorisággal.

6.5.10. JK flip-flop

Utolsó fontos alkatrészünk a SZÁMLÁLÓ. Többféle kialakítási módozata létezik, aszinkron és szinkron bináris számlálók, dekád számlálók, modulo számlálók, gyűrűs számlálók. A teljes paletta áttekintésére nincs módunk, csak a könyv további fejezeteihez kapcsolódó legszükségesebbeket *és azok működésének megértéséhez szükséges elméleti és gyakorlati megoldásokat* vizsgáljuk meg.

A számláló áramkörök általános építőkockája a JK FLIP-FLOP és annak változatai (a cella elnevezéséről érdekes részletek az előző fejezetben). Jack Kilby (fizikai Nobel díj 2000-ben) a Texas Instruments vállalatnál 1958-ban valósította meg integrált áramkör formájában. A neve kezdőbetűivel történetesen egyező elnevezésű bemenetekkel rendelkező bistabil billenőkör az RS cella „rokona”, annak zavaró, oszcilláló állapota nélkül (amikor R=S=1 volt a NEM-VAGY kapus változatban).

A J (beíró) és K (törlő) bemenetek az S és R jeleknek megfelelő funkcióval rendelkeznek, de úgy módosítva, hogy a külön engedélyező *órajellel* és a kimenetek visszacsatolt *állapotaival* is kondicionálva vannak. Ezt egy-egy hárombemenetű **és** kapu felhasználásával lehet megvalósítani (6.58. ábra).



C	J	K	Q	$\overline{Q}$
⎍	0	0	tárol	tárol
⎍	0	1	0	1
⎍	1	0	1	0
⎍	1	1	tagad	tagad
X	0	0	tárol	tárol
X	0	1	tárol	tárol
X	1	0	tárol	tárol
X	1	1	tárol	tárol

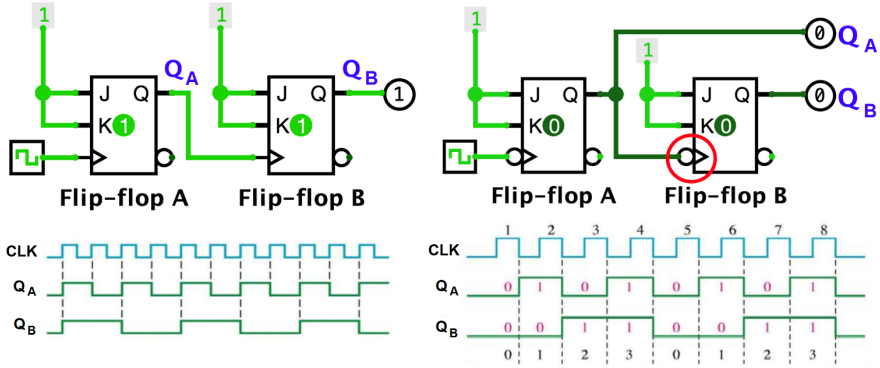
6.58. ábra. *Élvezérelt JK-tároló kapcsolási rajza és igazságtáblázata*

A kimenetek állapotával kondicionált beíró- és törlőjelek egymásra való visszahatásukat kölcsönösen kizárják. Ha a Q kimenet éppen 1-es, a J-t kondicionáló $\overline{Q}$ kizárja a J=1 hatását a cellára, és ha J=K=1 eset áll fenn, csak a K=1 hat, törölve a kimenetet. Ugyanígy, ha a Q kimenet éppen 0-ás, a K-t kondicionáló Q kizárja a K=1 hatását a cellára, és ha J=K=1 eset áll fenn, csak a J=1 hat, 1-be állítva a kimenetet. Ezért, ha a J=K=1 eset tartósan fennáll, az órajel minden felfutó élére változás következik be, a kimeneten negálva előző értékét: 0, 1, 0, 1… sorrendben.

- register in digital logic
- JK flip flop

6.5.11. Frekvenciaosztó és aszinkron bináris számláló

Ez a tulajdonsága a JK flip-flop áramkört kiválóan alkalmassá teszi FREKVENCIAOSZTÓ funkció megvalósítására. Egy cella kimenetén 2-vel osztja az órajel frekvenciáját. Több cellát sorba kapcsolva (előző cella kimenete a következő cella órajel bemenetére) 2 hatványai szerinti osztási értéket állíthatunk be. Ha a kimeneteket hozzáférhetővé tesszük, egy ASZINKRON BINÁRIS SZÁMLÁLÓT kapunk. A 6.59. ábrán a JK flip-flop szimbolikus ábrázolását alkalmazva adunk példát a fenti két áramkörre.



6.59. ábra. *Frekvenciaosztó és aszinkron bináris számlálólánc kialakítása élvezérelt JK-tárolókból*

A két áramkör kapcsolási rajza majdnem azonos, van azonban pár különbség. A frekvenciaosztónál az utolsó kimenetet használjuk fel a 2^N-nel osztott frekvenciájú jel továbbvitelére, a számlálónál minden cella kimenetére szükségünk van. De a legfontosabb, hogy a számlálónál az órajel LEFUTÓ élére történik a JK cella kimeneti állapotváltása (itt alkalmazzuk a pulzusgenerátor lefutó élre működő változatát, a 6.59. ábra jobb oldalán pirossal keretezett részlet). Erre a kis módosításra azért van szükség, mert felfutó élre működve a számláló visszafele, 3… 2… 1… 0 irányba számolna. A számláló esetében a legkisebb helyértéket generáló JK flip-flop az, amelyik a külső órajelet fogadja, az utána következők egyre nagyobb helyértékek szerinti értéket jelenítenek meg a kimeneteiken.

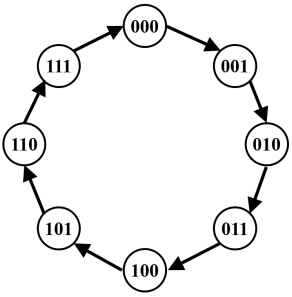
6.5.12. Szinkron bináris számláló

Az aszinkron számlálóláncok esetében is figyelni kell a kapuk átfutási idejére, ha az eredő késleltetés nagyobb lesz, mint az órajelimpulzus időtartama (nagy frekvenciájú órajel esetében), *a számláló kimenetén egyes állapotok hibásan jelenhetnek meg.* Ezért a gyakorlati megvalósításokban inkább SZINKRON SZÁMLÁLÓKAT használnak.

Példánkban a három JK flip-flop-ból álló, szinkron számláló minden cellájának órajelbemenete közös pontba kapcsolódik. Kialakításukhoz a szekvenciális áramkörök vagy állapotgépek tervezési lépéseit alkalmazzuk, felrajzolunk egy állapotdiagramot, amelyben 000-tól 111-ig terjednek binárisan növekvő sorrendben a kimenetek lehetséges értékei.

Ezután felírjuk a JK flip-flop kimeneti állapotváltásainak, Q_n és Q_{n+1}-nek a J és K bemenetek függvényében felállított igazságtáblázatát. A táblázatban megvizsgáljuk azokat a sorokat, amelyekben Q_n és Q_{n+1} azonos értékeket mutatnak (6.60. ábra

– színekkel kiemelve), *és* ezekre a kombinációkra egy újabb, úgynevezett excitációs táblázatba rögzítjük a J és K bemenetek azon értékeit, amelyek a kimeneti állapot változását vagy megtartását kódolják. Ilyenkor csak a vizsgált esetekben változatlan bemeneti értékek számítanak, a változó bemenetet **x**-szel jelöljük.



J	K	Qn	Qn+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

Qn	Qn+1	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

6.60. ábra. *Szinkron bináris számlálólánc állapotdiagramja és a JK flip-flop igazság és excitációs táblázatai*

CLK	Aktuális állapot			Következő állapot			Flip-flop bemenetek					
	Q_C	Q_B	Q_A	Q_{C+1}	Q_{B+1}	Q_{A+1}	J_C	K_C	J_B	K_B	J_A	K_A
1	0	0	0	0	0	1	0	X	0	X	1	X
2	0	0	1	0	1	0	0	X	1	X	X	1
3	0	1	0	0	1	1	0	X	X	0	1	X
4	0	1	1	1	0	0	1	X	X	1	X	1
5	1	0	0	1	0	1	X	0	0	X	1	X
6	1	0	1	1	1	0	X	0	1	X	X	1
7	1	1	0	1	1	1	X	0	X	0	1	X
8	1	1	1	0	0	0	X	1	X	1	X	1

6.61. ábra. *Hárombites, szinkron, bináris számlálólánc összetett excitációs táblázata (a JK-cellák ki- és bemenetei azonos színnel jelölve)*

Az így kapott táblázatból képezzük a szinkron számlálólánc összetett excitációs táblázatát (6.61. ábra). Felírjuk mindhárom JK flip-flop aktuális (Q_A, Q_B, Q_C) és következő kimeneti állapotainak kombinációit (Q_{A+1}, Q_{B+1}, Q_{C+1}), majd további oszlopokba feljegyezzük az összes bemenet állapotát (J_A, K_A stb.).

A táblázat első oszlopában az órajel ütemeit jelöljük, de ennek nincs befolyása a további műveletek szempontjából. A táblázatból JK-cellákra bontva képezzük a bemenetek Karnaugh-diagramjait (ne feledjük a Gray-kód szerinti oszlopszámozást alkalmazni), és próbáljuk kialakítani a lehető legnagyobb, összefüggő 1-es vagy x értékcsoportokat (6.62. ábra).

J_C	Q_BQ_A\Q_C	00	01	11	10
	0	0	0	1	0
	1	X	X	X	X

$J_C = Q_BQ_A$

J_B	Q_BQ_A\Q_C	00	01	11	10
	0	0	1	X	X
	1	0	1	X	X

$J_B = Q_A$

J_A	Q_BQ_A\Q_C	00	01	11	10
	0	1	X	X	1
	1	1	X	X	1

$J_A = 1$

K_C	Q_BQ_A\Q_C	00	01	11	10
	0	X	X	X	X
	1	0	0	1	0

$K_C = Q_BQ_A$

K_B	Q_BQ_A\Q_C	00	01	11	10
	0	X	X	1	0
	1	X	X	1	0

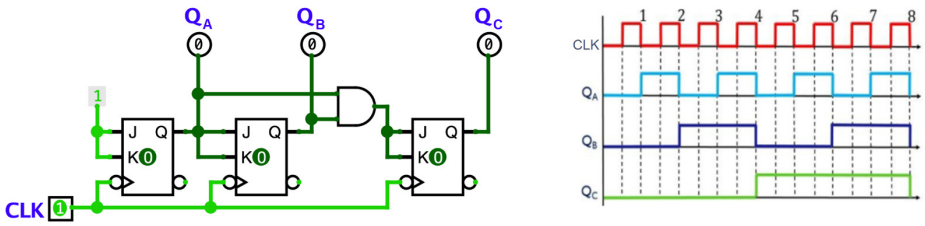
$K_B = Q_A$

K_A	Q_BQ_A\Q_C	00	01	11	10
	0	X	1	1	X
	1	X	1	1	X

$K_A = 1$

6.62. ábra. *Hárombites, szinkron, bináris számlálólánc bemeneteinek Karnaugh-táblázatai*

A kialakított csoportok alapján felírhatjuk a J_C, K_C, J_B, K_B, J_A, K_A bemenetek logikai függvényeit, majd ezeknek megfelelő logikai áramköri megoldásokat alakítunk ki. Minden JK-cellára igaz a J=K egyenlőség. A csupa 1-es csoportok által meghatározott bemeneteket logikai magas (H) állapotban fogjuk tartani. Figyeljük meg, hogy az órajel itt is a lefutó élre aktív a számlálás irányának növekvő sorrendje érdekében.



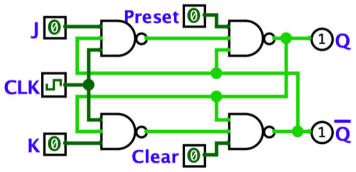
6.63. ábra. *Hárombites, szinkron, bináris számlálólánc kapcsolási rajza és kimeneteinek idődiagramja*

A szinkron bináris számlálónál az órajel felfutó vagy lefutó éle NEM befolyásolja a számlálás sorrendjét, ez a bemutatott áramkör esetében mindig növekvő irányban történik. A kapcsolási rajzon (6.63. ábra) az egyszerűség kedvéért az órajel lefutó élére működő JK-cellákat használtuk.

6.5.13. Johnson- vagy gyűrűs számláló

Egyszerűbb kivitelű számítástechnikai rendszerekben az utasítás-végrehajtó egység vezérlőjeleinek szinkronizálásához egy olyan számlálóra van szükség,

amelyik növekvő sorrendben, egymás után aktiválja egy-egy kimenetét, a folyamat végén pedig újrakezdi a ciklust. Az ilyen áramköröket GYŰRŰS SZÁMLÁLÓKNAK nevezzük. Ezek is SZINKRON típusú számlálók, celláik közös órajelre kapcsolódnak. Működésük megértéséhez egy eddig nem tárgyalt részletet vezetünk be a JK-cellák vonatkozásában, az aszinkron beállító bemenetek fogalmát.



Preset	Clear	CLK	J	K	Q	$\overline{Q}$
X	1	X	X	X	0	1
1	X	X	X	X	1	0
0	0	↑	1	0	1	0
0	0	↑	0	1	0	1
0	0	↑	1	1	1*	0
0	0	↑	0	0	1~	0

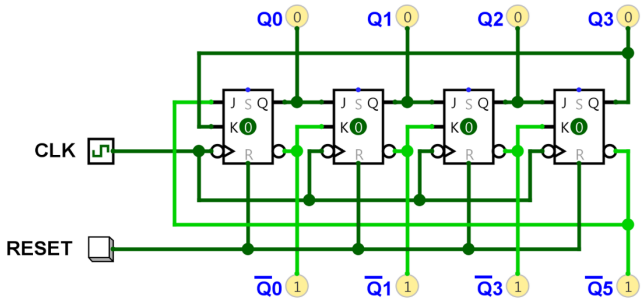
6.64. ábra. *JK-cella aszinkron beállító jelbemenetekkel és a hozzá tartozó igazságtáblázat*

Ez azt jelenti, hogy a JK-cella kimenete az órajeltől, a J és K bemenetektől függetlenül 1-be (PRESET) vagy 0-ba (CLEAR) állítható ezeknek a speciális bemeneteknek az aktiválásával. Egy ilyen kialakítású áramkör és igazságtáblázata alább látható (6.64. ábra).

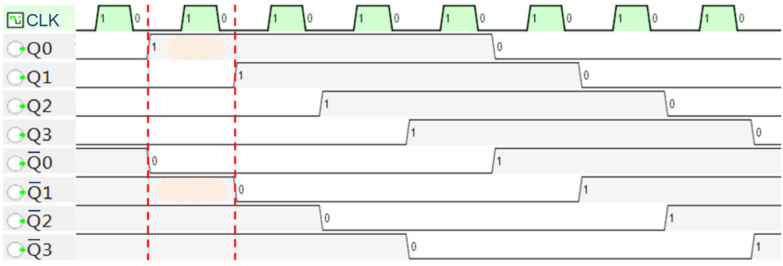
A gyűrűs számláló kialakításánál fontos szempont, hogy amíg az órajel pulzál, addig a számlálási folyamat ciklikusan újra és újra lefusson. Robert Royce Johnson 1953-ban szabadalmaztatott egy azóta is róla elnevezett gyűrűs számlálót. A Johnson-számlálóban a JK-cellák Q és $\overline{Q}$ kimenetei az utánuk következő, nagyobb helyértéken lévő cella J és K bemeneteire kapcsolódnak ($Q_n=J_{n+1}$, $\overline{Q_n}=K_{n+1}$). A számlálólánc utolsó JK-cellájának a kimenetei keresztben csatolódnak vissza a legkisebb helyértéken lévő cella bemeneteire ($Q_{MSB}=K_{LSB}$, $\overline{Q}_{MSB}=J_{LSB}$). A számlálólánc induláskor nem igényel aszinkron módú alapállapot-beállítást. Ezt a megoldást mintaszámlálónak is nevezik, mert a kimenetek egy kialakuló bináris mintát forgatnak *körbe*, ahogy a 6.65. ábra igazságtáblázata is mutatja. A RESET bemenet aktiválásával mindig lehetőség van a kimenetek azonnali 0-ba állítására.

A gyűrűs számláló kialakításánál fontos szempont, hogy amíg az órajel pulzál, addig a számlálási folyamat ciklikusan újra és újra lefusson. Az időidiagram (6.66. ábra) könnyebben láthatóvá teszi, hogy az órajel ütemére (ismét a lefutó élre aktív) a kimenetek egyenként, növekvő helyérték sorrendben 1-be állnak, majd a ciklus végére sorban 0 állapotba térnek vissza. Figyeljük meg, hogy a teljes folyamat 8 órajelimpulzust igényel.

CLK	Q0	Q1	Q2	Q3
0	0	0	0	0
1	1	0	0	0
2	1	1	0	0
3	1	1	1	0
4	1	1	1	1
5	0	1	1	1
6	0	0	1	1
7	0	0	0	1
8	0	0	0	0



6.65. ábra. *Johnson-számláló igazságtáblázata és kapcsolási rajza*



6.66. ábra. *Johnson-számláló idődiagramja a Q_n és $\overline{Q}_n$ kimenetekkel*

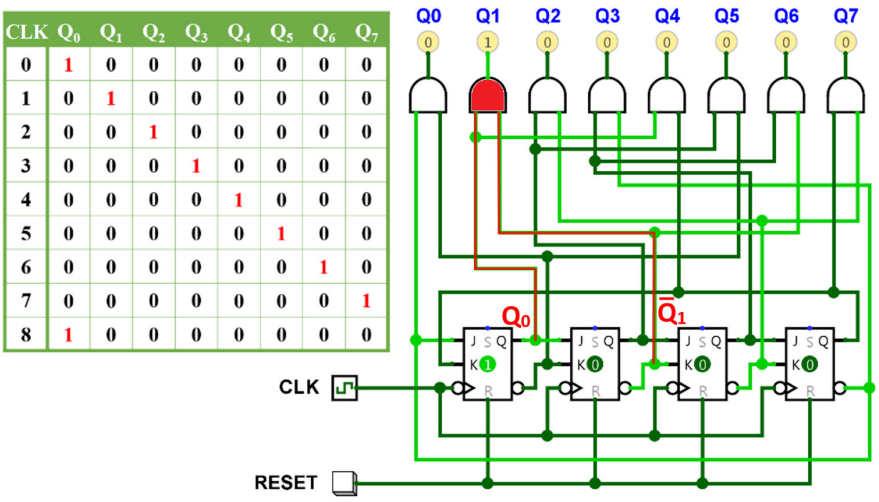
6.5.14. Módosított Johnson-számláló

Ez a minta még nem felel meg a gyűrűs számlálók bevezetésénél megfogalmazott követelménynek, miszerint egyszerre csak egy kimeneten szeretnénk 1-es értéket látni. De az idődiagramot figyelmesen vizsgálva észrevesszük, hogy vannak „lépcsőfokok” a direkt (Q) és tagadott ($\overline{Q}$) kimenetek közül, amelyek ugyanabban az időszeletben 1-ben állnak.

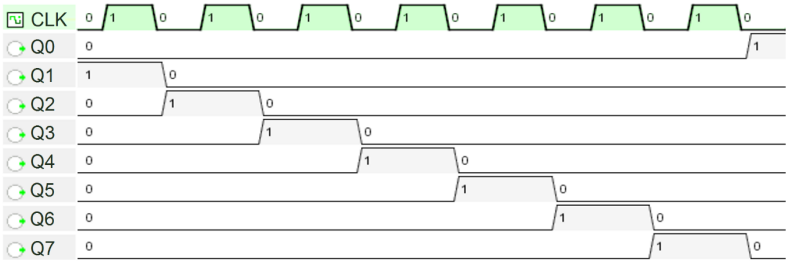
Ha a megfigyelést kihasználva, ezeket a Q és $\overline{Q}$ kimeneteket egy-egy ÉS kapuban összefogjuk, pontosan az elvárt eredményt fogjuk kapni.

A 6.67. ábra a kapcsolási rajzban pirossal kiemelve mutatja azt az ÉS kaput, amely a 6.66. ábra szerinti idődiagramban, a piros szaggatott vonalak között kiemelt időintervallumban, a Q_0 és $\overline{Q}_1$ (nQ_1) jelek 1-ben való együttállását hasznosítja. Az összes fennálló kombinációt sorba véve megkapjuk a módosított Johnson-számláló kapcsolási rajzát. Az eredményül kapott idődiagramot a 6.68. ábra mutatja.

CLK	Q_0	Q_1	Q_2	Q_3	Q_4	Q_5	Q_6	Q_7
0	1	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0
2	0	0	1	0	0	0	0	0
3	0	0	0	1	0	0	0	0
4	0	0	0	0	1	0	0	0
5	0	0	0	0	0	1	0	0
6	0	0	0	0	0	0	1	0
7	0	0	0	0	0	0	0	1
8	1	0	0	0	0	0	0	0



6.67. ábra. *Módosított Johnson-számláló igazságtáblázata és kapcsolási rajza*

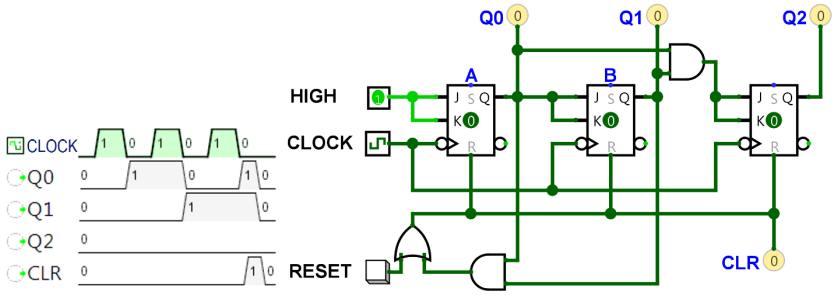


6.68. ábra. *A módosított Johnson-számláló idődiagramja*

6.5.15. Modulo számláló

Az utolsó számlálótípus, amit megvizsgálunk, az úgynevezett bináris MODULO SZÁMLÁLÓ. Ez az áramkör egy kiválasztott értékig számol, majd újrakezdi a ciklust. Egyszerűbb kivitelű számítástechnikai rendszerekben ilyen számlálókat is alkalmazhat az utasítás-végrehajtás lépéseinek ütemezésére.

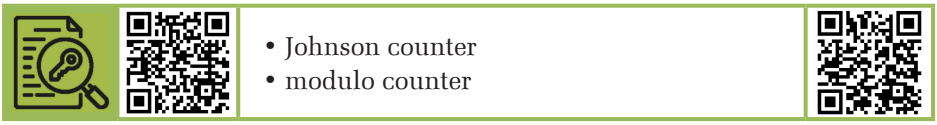
Példának alakítsunk ki egy modulo 3 bináris számlálót, amely 0, 1, 2 után 3-ra 0-ba löki vissza a számlálólánc kimeneteit. Előzőekben már tanulmányoztuk a szinkron bináris számlálókat, egy ilyen áramkör egy kis módosítással éppen megfelelő lesz.



6.69. ábra. *A Modulo 3 számláló idődiagramja és kapcsolási rajza*

A módosítás lényege, hogy egy bináris érték „szűrőt” alakítunk ki, amely áramkör a számlálólánc kimeneteit követve a 3-as szám (binárisan 11) elérésekor, még ugyanabban az órajelciklusban (!) aktiválja a JK-cellák aszinkron RESET bemenetét, így a számláló kimenete 0-ra vált és újrakezdi a számolást.

Ha két kimenet 1-ben való együttállását kell figyelnünk, egy ÉS kaput használhatunk, ennek kimenete fogja vezérelni a cellák RESET bemenetét. A 6.69. ábra idődiagramján látható, hogy a $Q_0=Q_1=1$ esetben a CLR jel 1-be billen és RESET-eli a számlálót. A kapuk átfutási idejéig a $Q_0=Q_1=1$ kimeneti érték jelenik meg, de az órajel ciklusidejének nagyobb részében a $Q_0=Q_1=0$ dominál.



- Johnson counter
- modulo counter

7. Számítógéparchitektúra-típusok, mikroarchitektúra-tervezés

7.1. A számítógép architektúrája

Az eddigi fejezetekben áttekintett áramköri megoldások, kombinációs és szekvenciális kapcsolások megalapozták a most következő részeket. Az eddig különálló egységekként kezelt áramköröket egy nagyobb, bonyolultabb rendszerbe szervezzük, egy számítógépet hozunk létre belőlük.

Nagyon leegyszerűsítve a számítógép egy központi utasítás-végrehajtó egységből (CPU – Central Processing Unit), egy utasításokat és adatokat tároló központi memóriából (System Memory), valamint ki- és bemeneti eszközök csatolására alkalmas áramkörökből (I/O Interface) áll. Ezeket az egységeket párhuzamos vezetékek kötegei (adat-, cím- és vezérlősínek vagy buszok) kapcsolják össze egymással. A számítógép architektúrája a felsorolt egységek egymáshoz kapcsolásának módozatát, az egész gép és részegységeinek működését és viselkedési sajátosságait írja le vagy rögzíti valamilyen szimbolikus ábrázolásmóddal.

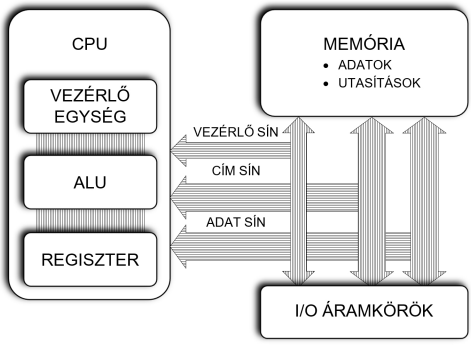
Az egységesebb számítógép-architektúra (computer architecture) kifejezést csak az 1970-es évektől kezdődően vonatkoztatják számítástechnikai rendszerek felépítésének leírására. Ez előtt a tárgyban megjelenő, kiemelten fontos dokumentum a 2. fejezetben már bemutatott, Neumann János által 1946-ban kidolgozott leírás (First Draft of a Report on the EDVAC), amely egy számítástechnikai rendszert és annak utasítás-végrehajtási módszerét írja le. Ez talán az első olyan összefüggő tudományos munka, amely fogalomkörönként tárgyalja és határozza meg az alkotóelemeket, írja le a közöttük fennálló matematikai összefüggéseket és von le következtetéseket az írás tárgyát képező rendszerről. Megjegyzendő, hogy ebben és az ezt követő írásokban még évtizedekig nem szerepel majd a számítógép-architektúra kifejezés.

Az elmúlt évtizedek számítástechnikai rendszereinek fejlődéstörténetében többféle irány jelentkezik, amelyek az alkotóelemek közötti kapcsolatokat és ezek működését részben hasonló, részben eltérő módon határozzák meg. A különbségek és hasonlóságok rendszere igencsak szövevényes, de kiemelhető néhány jól észrevehető sajátosság, amelyek mentén napjaink számítástechnikai szakirodalma szétválaszt bizonyos fejlődési irányokat. Ezek az irányok nem zárják ki egymást, napjaink számítástechnikai rendszerei gyakran ötvözik ezeket a műszaki megoldásokat.

- computer architecture
- evolution of computer architecture

7.1.1. Neumann- és Harvard-architektúrák

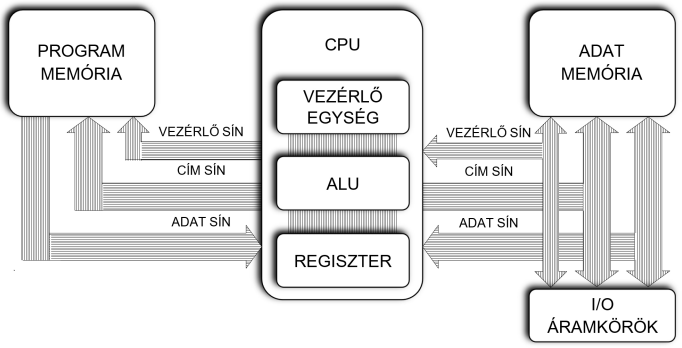
Az első megkülönböztetés a memória számítástechnikai rendszerbe kapcsolásának módozata mentén lehetséges. Az 1944 és 1952 közötti időszakban két alapvető rendszert dolgoztak ki. Az egyiket a 7.1. ábra mutatja, ez az elrendezés a Neumann-architektúra néven ismert (kidolgozója Neumann János – John von Neumann – 1946, ENIAC, EDSAC) rendszer. Sajátossága, hogy egyetlen, központi memóriát határoz meg, amelyben adatok és utasítások egyaránt tárolhatók. A központi végrehajtóegység ebből a memóriából olvassa be az utasításokat, a hozzájuk tartozó adatokat, és ebbe a memóriába írja vissza az utasítások végrehajtása során keletkező adateredményeket.



7.1. ábra. *A Neumann-féle számítógép architektúrája*

Mivel az adatok és utasítások bináris megjelenítése között nincs különbség, elképzelhető, hogy olyan programot írjunk erre a rendszerre, amely a memória bizonyos részeinek átírásával újabb utasítássorozatot hozhat létre, akár megváltoztatva ezzel az eredeti, úgynevezett létrehozó program utasításait is.

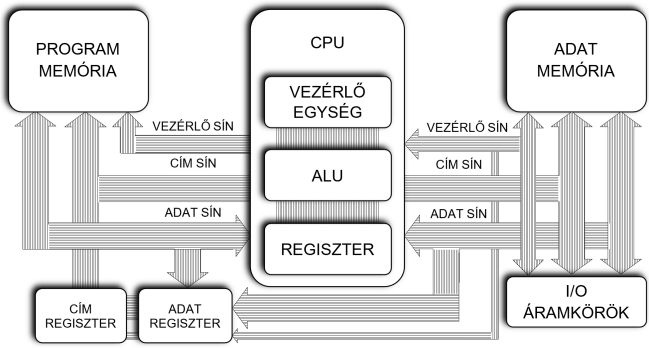
A másik rendszer Howard Aiken nevéhez fűződik, aki az egyesült államokbeli Harvard Egyetem számításokkal foglalkozó laboratóriumának vezetőjeként (Harvard Computing Laboratory – HCL) több számítógéprendszer kidolgozását is irányítja (MARK-generációk). Rendszereinek sajátossága, hogy két memóriát határoz meg, az egyikben kizárólag adatok, a másikban csak utasítások tárolhatók, ahogy a 7.2. ábra mutatja. Mindkét memóriát külön cím, adat és vezérlősín kapcsolja a központi végrehajtóegység (CPU) vezérlő áramköréhez (CU). A vezérlőegység rendre beolvassa az utasításokat, de még a végrehajtás legelső fázisában, a külön cím, adat és vezérlősínnek köszönhetően hozzáfér az utasításokhoz tartozó adatokhoz is, miközben az utasításmemóriából már egy újabb utasítás kódját olvashatja be. Így az utasítás-végrehajtás gyorsabb is lehet, mint a Neumann-rendszerben.



7.2. ábra. *Az Aiken-féle Harvard-számítógép architektúrája*

Aiken rendszerének nem feltétlen hátránya, hogy a két memória között nincs átjárás, azaz a végrehajtott program nem képes a programmemóriát írni, először vélhetően műszaki kényszermegoldás lehetett. Gondoljunk arra, hogy a kezdeti számítógéprendszerek mechanikus alegységek segítségével lyukkártyákról vagy papírszalagról olvasták be az utasításokat, míg a dinamikusan változó adatokat már elektromágneses vagy éppen elektroncsöves kialakítású memóriaregiszterekben tárolták. Mégis a kezdeti operációs rendszerek felbukkanása idején kifejezetten hátrányossá vált ez az elképzelés, az utasítások végrehajtása által átírható memóriát használó Neumann-megoldás hasznosabbnak bizonyult.

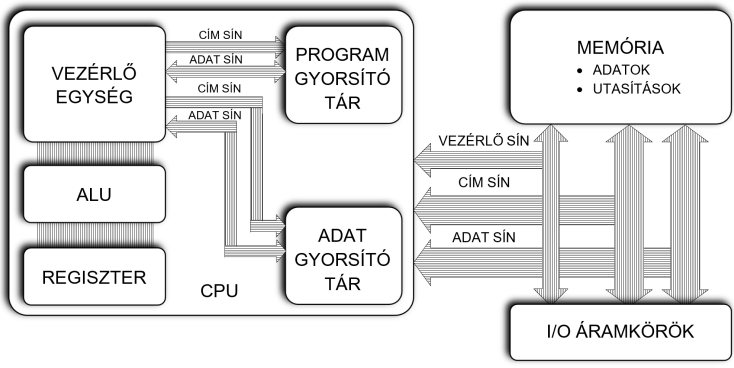
Az utasítás-végrehajtás gyorsításának igénye viszont a két rendszer ötvözésével létrehozta a gyakran „módosított”, Harvard-architektúra néven ismert változatot, ahol a programmemória is írhatóvá és olvashatóvá válik, és ily módon az utasítások végrehajtása által is átírható (7.3. ábra).



7.3. ábra. *Módosított Harvard-architektúra. A program- és adatmemória között átjáró nyílik külső cím- és adatregiszteren keresztül*

A mai számítógép-architektúrák komplex rendszerei továbbfejlesztették a három kezdeti elképzelést. Megmaradt a rendszerben a központi memória, de megjelent a gyorsítótár (cache memory), a processzor szilíciumlapkájára integrált memória, amely közvetlen, úgynevezett belső cím-, adat- és vezérlőjelsíneket használva, kapcsolatban van a központi végrehajtóegységgel. A központi memória általában egy memóriavezérlő, külső (de újabban már belső), illesztő áramkörön (interfészen) keresztül, külön cím-, adat- és vezérlőjelsíneket használva kapcsolódik a processzorhoz. Méréseredmények bizonyítják, hogy gyorsítótár alkalmazásával, amelyben vagy csak adatokat, vagy csak utasításokat tárolnak a központi végrehajtóegység közvetlen, fizikai közelében, a programvégrehajtás sebességét általánosan 1,25-szörösével (adat cache), illetve 1,95-szörösével (utasítás cache) növeli a gyorsítótárat nélkülöző megoldáshoz viszonyítva.

További fejlesztés a gyorsítótár két részre osztása, külön cím-, adat- és vezérlőjelsíneket használva a csak utasításokat és a csak adatokat tároló részek eléréséhez. Ebben az esetben 2,5-szörös végrehajtási sebességnövekedés érhető el. Ezekre a rendszerekre egyes leírásokban „szuper” Harvard-architektúraként hivatkoznak (7.4. ábra). Figyeljük meg, hogy ez a kialakítás a Neumann- és Harvard-architektúrák sajátos kombinációját mutatja. Megvan a központi memória, amiben utasítások és adatok tárolódnak, ezekből tölti fel a vezérlőegység a belső utasítás és adatgyorsítótárakat.



7.4. ábra. *Szuper Harvard-architektúra, egy újabb megvalósítás*

- Von Neumann architecture
- Harvard architecture

7.1.2. A 0, 1, 2 vagy több című architektúratípusok

Az architektúrák közötti második megkülönböztetés a központi végrehajtóegység belső felépítésének mentén történhet, annak alapján, hogy hány belső regiszterrel rendelkezik az utasítás-végrehajtó áramkör. A központi végrehajtóegység az utasítás-végrehajtás szempontjából szakaszos (szekvenciális) működési módja okán tárolócellák (regiszterek) közbeiktatását igényli mind az utasítás, mind az adatsínek mentén. Ezek a tárolócellák a központi végrehajtóegység szempontjából lehetnek a külső cím-, adat- és vezérlősínhez illesztettek, vagy a belső cím-, adat- és vezérlősínhez illesztettek.

Figyelmünket most az adatsínre és az egyes utasítások végrehajtására fordítjuk, azt vizsgálva, hogyan kapcsolódik az adott architektúrán belül az aritmetikai-logikai egység be- és kimeneteire tárolócella, illetve szükséges-e adott utasítások esetében (amelyek végrehajtásához az ALU szükséges) memóriacím megadása.

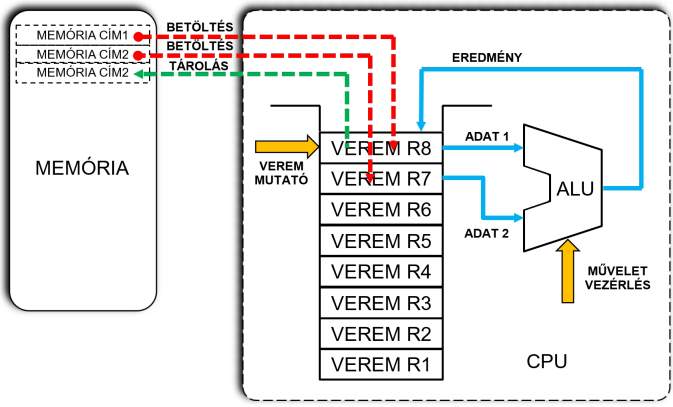
A 0 című architektúra egy belső memóriaegységként működő adatverem (data-stack) celláit használja köztes tárolóként a műveletvégzés során. Általánosan a verem egy olyan memóriarendszer, amelybe az adatok a beérkezés sorrendjében tárolódnak, a legrégebben beírt adat van a legmélyebben a veremben, a legfrissebben beírt adat meg a tetején. A verem tetejének címét az úgynevezett veremmutató (stack pointer) tárolja. A veremben az adatmozgás virtuális, fizikailag a beírt adatok ugyanabban a cellában maradnak a felülírásukig, csak a veremmutató értéke változik, beírás esetén növekszik egyel, kiolvasás esetén csökken egyel.

A vizsgált esetben az ALU két bemenetére az adat-verem két legfelső cellája van hozzákapcsolva (7.5. ábra). Előzetesen az adat-verem celláit adatmozgató utasításokkal feltöltjük a művelet elvégzéséhez szükséges adatokkal. Ezek az utasítások vagy központimemória-címre hivatkoznak (amelyről az adat a verembe kerül), vagy állandó értéket állítanak be az adat-verem tárolócellájába. Ezután a soron következő műveleti utasítás (összeadás, kivonás, szorzás stb.) már operanduscímet nem tartalmaz. Végrehajtásakor az ALU a két bemeneti adaton elvégzi az előírt műveletet. Ezért nevezzük ezt a kapcsolást 0 című architektúrának.

Következő lépésben egy adatmozgató utasítás gondoskodhat az eredménynek a verem tetejéről a központi memória egy adott címére való átmásolásáról, de más esetben egy újabb műveleti utasítás következhet, ami az előzőleg elvégzett művelet eredményét használja bemenő adatként anélkül, hogy adatmozgatást kellene végezni a műveleti utasítások között (pl. sorozatos összeadás vagy szorzás).

Az 1 című architektúra egy nevezetes, belső tárolócellát alkalmaz a köztes adat megtartáshoz, az úgynevezett munkaregisztert vagy más néven akkumulátort (**ACCU**). Ez a tárolócella egyidejűleg képezi az ALU elsődleges adat bemenetét és az eredmény kimenetét.

A vizsgált esetben az ALU egyik bemenetére a munkaregiszter (akkumulátor), a másik bemenetére egy „átlátszó”, belső tárolóregiszter (adatregiszter) van hozzákapcsolva (7.6. ábra). Ez a tárolóregiszter az éppen végrehajtott utasításban



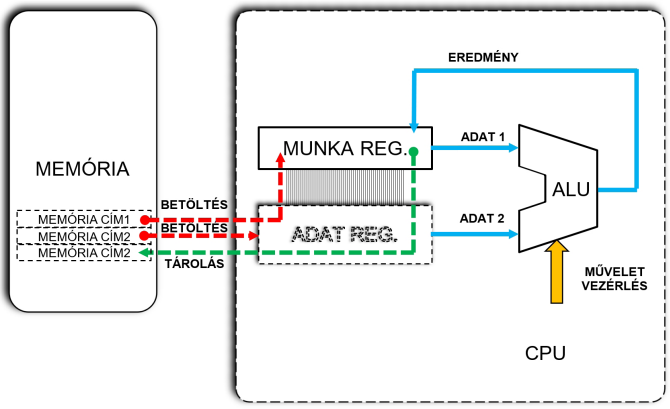
7.5. ábra. *0 című architektúra, belső tárolócellák nélküli kialakítás*

megadott, központimemória-címről beolvasott adatot, vagy az utasításszóba foglalt állandó értéket tárolja ideiglenesen. Az ábrán ez az „átlátszó”, belső tárolóregiszter szaggatott vonalas keretben látható, arra utalva, hogy rajta keresztül közvetlenül az utasításszóban megcímzett központimemória-rekesz tartalmát „látjuk”. Műveletvégzés előtt a munkaregiszter tartalmát töröljük (clear) egy erre hivatott utasítással. Ezután egy műveleti utasítással (pl. összeadás) feltöltjük a művelet elvégzéséhez szükséges adattal (mivel előzőleg töröltük a tartalmát, az összeadás művelet 0-hoz fogja hozzáadni az utasításban megadott központimemória-címről beolvasott adatot vagy az utasításszóba foglalt állandó értéket). A soron következő műveleti utasítás (összeadás, kivonás, szorzás stb.) ismét operandus címet tartalmaz, hiszen a központi memóriából is beemelhet egy változó értéket. A műveleti utasítás végrehajtásakor az ALU a két bemeneti adaton (egyik a munkaregiszterben, másik az „átlátszó” bemeneti adatregiszterben tárolva) elvégzi az előírt műveletet, az eredményt a munkaregiszterbe tárolja. Mivel a műveleti utasításban csak egyetlen cím szerepel, ezt a kapcsolást 1 című architektúraként határozzuk meg.

Következő lépésben egy adatmozgató utasítás gondoskodhat az eredménynek a munkaregiszterből a központi memória egy adott címére való átmásolásáról.

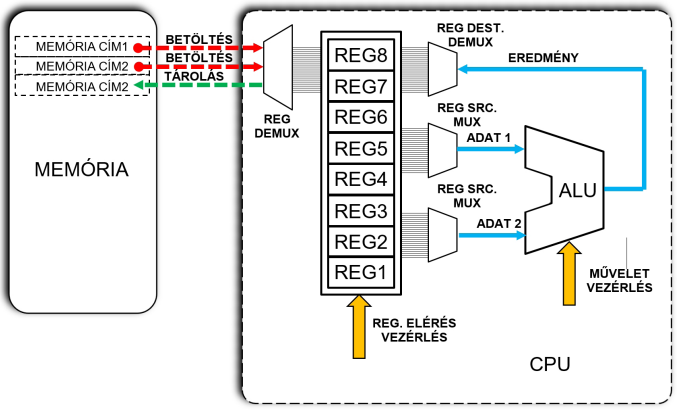
A 2 vagy több című architektúra esetében (7.7. ábra) a nevezetes belső regisztert, az akkumulátort úgynevezett általános rendeltetésű, belső tárolócellák (regiszterek) váltják fel. Bármelyik tárolócella képezheti az ALU be- és kimeneti munkaregiszterét, azaz szolgálhat forrásként (source, ahonnan adatot olvasunk be) és célként (destination, ahova adatot tárolunk el).

A 2 című esetben az ALU egyik bemenetére és kimenetére ugyanaz a kiválasztott (az utasításszóban megjelölt, értsd megcímzett), általános rendeltetésű regiszter kapcsolódik (egyidejűleg forrás és cél szerepben), míg a másik bementére egy ettől különböző, általános rendeltetésű regiszter kapcsolódik (csak forrás szerepben).



7.6. ábra. *1 című architektúra belső, eredménytároló cellával, akkumulátorral kialakított megoldás*

A több című esetben az ALU mindegyik bemenetére (az utasításszóban megjelölt, értsd megcímzett) általános rendeltetésű regiszter kapcsolódik (forrás szerepben), míg a kimenetére egy harmadik, általános rendeltetésű regiszter kapcsolódik (cél szerepben).



7.7. ábra. *2 vagy több című architektúra, belső, általános tárolócellákkal kialakított megoldás*

Műveletvégzés előtt az általános célú, forrásként használt regisztereket feltöltjük a művelet elvégzéséhez szükséges, az utasításban megadott központimemória-címről beolvasott adatokkal vagy az utasításszóba foglalt állandó értékkel. A soron következő műveleti utasításnak (összeadás, kivonás, szorzás stb.) külön-külön

tartalmaznia kell a forrás- és célregiszterek címeit. A műveleti utasítás végrehajtásakor az ALU a két bemeneti adaton elvégzi az előírt műveletet, az eredményt az utasításszóban megadott, általános célú regiszterben tárolja. Mivel a műveleti utasításban 2 vagy több cím szerepel, ezt a kapcsolást 2 vagy több című architektúraként határozzuk meg.

Következő lépésben egy adatmozgató utasítás gondoskodhat az eredménynek az általános célú regiszterből a központi memória egy adott címére való átmásolásáról.

- zero one two address architecture
- instruction format types

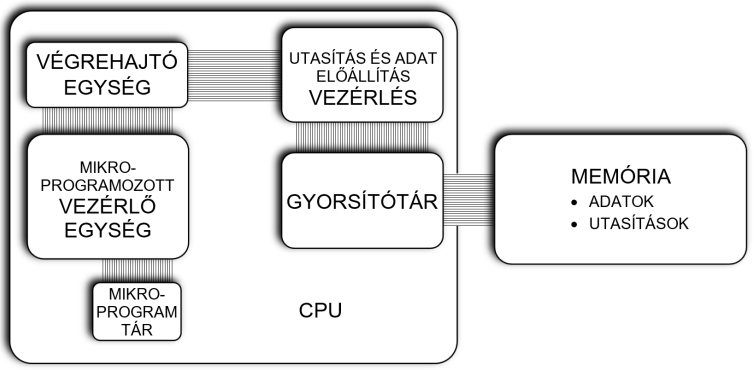
7.1.3. CISC-, MISC- és RISC-architektúrák

Az architektúrák közötti harmadik megkülönböztetés az utasításszavak száma és kialakítása mentén történhet. Több csoportot különböztetünk meg, a kiterjesztett vagy komplex utasításkészletű architektúrákat (CISC – Complex Instruction Set Computer), a nagyon kevés utasítást használó architektúrákat (MISC – Minimal Instruction Set Computer) és a csökkentett vagy egyszerűsített utasításkészletű architektúrákat (RISC – Reduced Instruction Set Computer). A számítástechnika történetének korai szakaszában ez a csoportosítás még nem létezett.

A fejlődéstörténet későbbi szakaszaiban igény mutatkozott az egyre bonyolultabb utasítások hardveres végrehajtási vonalainak kidolgozására, ezt a mikroprogramozott vezérlőegységek megjelenése különösen támogatta, szemben a huzalozott vezérlőegységek előnytelen elbonyolódásával. Ahogy nőtt az utasítások bonyolultsága és száma, úgy nőtt a végrehajtásukhoz szükséges mikroprogram mérete és komplexitása. A CISC-architektúra bonyolult műveletek elvégzéséhez célzott, komplex utasításokat használ, amelyek gyakran több tíz vagy több száz mikroprogram-utasítás vagy -sor végrehajtását feltételezik. Előnyként jelentkezik, hogy a központi memóriában a bonyolult műveletek viszonylag rövid, egyszerű utasításszavakként jelennek meg, kevés tárhelyet foglalva. Hátrányként viszont az egyes utasításszavak végrehajtási idejének elnyúlása jegyezhető fel. Egy CISC-architektúra esetében az utasítás-végrehajtás utasításonként változó, egyes komplex utasítások végrehajtási ideje több tízszerese is lehet más, kevésbé komplex utasításokénál. A CISC-architektúra számos, sajátos feladatot, komplex műveletsorozatot, kész algoritmusokat megvalósító utasítást képes végrehajtani. Az utasításlista ismeretében viszonylag könnyen programozható, a már kész megoldásokból kikombinálható algoritmusok nagyon bonyolult számításokat,

vektorműveleteket, adatmozgatásokat is megoldhatnak. Komplexitásuktól függően viszont az utasításszavak hossza nem azonos.

A CISC-architektúrában (7.8. ábra) kiemelt szerep jut a mikroprogram-vezérlő egységnek és mikroprogramtárnak, amelyek a vezérlőegységhez csatlakozva fogadják és végrehajtják a beolvasott adatokra vagy adatsorokra vonatkozó utasításokat.

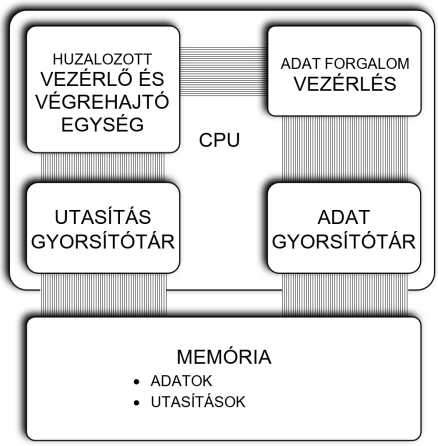


7.8. ábra. *CISC- és MISC-architektúrák tömbvázlata az utasítás-végrehajtási lánc belső egységei közötti adatutak megjelölésével*

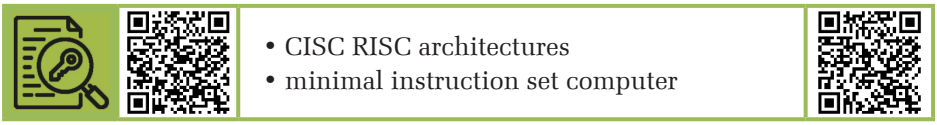
Az egyre komplexebbé váló utasításkészlettel rendelkező processzorok megjelenésével párhuzamosan a számítástechnikai fejlődéstörténet egy újabb ágat hoz létre. Az 1970-ben megjelenő, első mikrovezérlő architektúrák (egy szilíciumlapkára integrált számítástechnikai rendszer), az akkori technológia szintjén megvalósítható memóriaméretek és egyéb integrált hardveres megoldások tükrében nem alkalmazhatják a CISC-architektúrák komplex utasításai nyújtotta előnyöket. A viszonylag kis órajelfrekvencián elérhető maximális végrehajtási sebességre koncentrálva, előnybe helyezik az egyszerű, egyforma szóhosszúságú és főleg azonos számú, nagyon kevés órajelciklus alatt végrehajtható utasításkészleteket, létrehozva ezzel a MISC-architektúratípust.

A RISC-architektúra (7.9. ábra) a lecsökkentett utasításkészlet technikáját a Harvard-architektúraként ismert kapcsolási módból következő utasítás-végrehajtási ciklusidő csökkenésének előnyével kapcsolja össze (külön síneken csatolt utasítás és adatmemóriák, egyidejű utasítás és adatbeolvasás). A RISC-rendszerben az egyszerű, rövid, azonos szóhosszúságú, azonos ciklusidő alatt végrehajtható utasításkészletből következően párhuzamos végrehajtási „vonalak” (pipeline) alakíthatók ki. A „vonalak” külön utasításregiszterből, vezérlőegységből és ALU-ból épülnek fel, és a sorban következő utasítások végrehajtását egymástól, az utasításmemória hozzáférési idejével eltolva kezdik meg és hajtják végre (egyidejűleg csak az egyik „vonal” foglalhatja le a cím- és adatsíneket).

A rendszer hátrányaként a komplex programozási technika, a bonyolultabb algoritmusok megvalósításához szükséges utasítások sokaságának tárhelyigénye róható fel.



7.9. ábra. *Az RISC-architektúra tömbvázlata az utasítás-végrehajtási lánc belső egységei közötti adatutak megjelölésével*



7.2. Mikroarchitektúra-tervezés

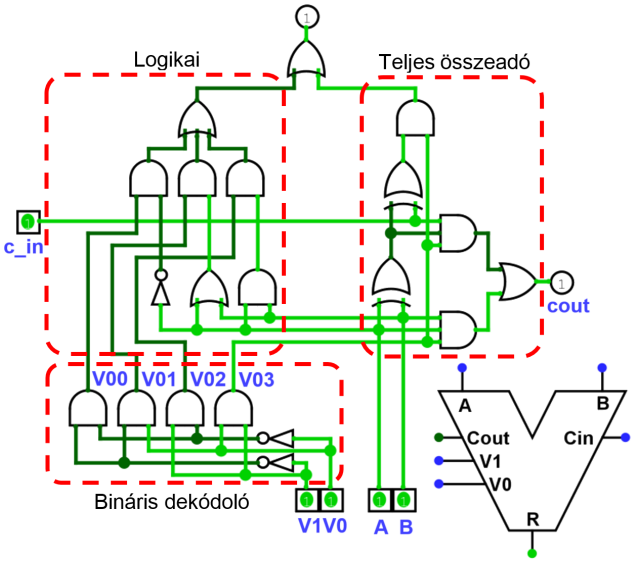
A mikroarchitektúra elnevezés a továbbiakban egy olyan rendszerre vonatkozik, amely több részegységből, aritmetikai-logikai egységből (ALU – Arithmetical Logical Unit), utasításdekódoló és vezérlőegységből (CU – Control Unit), valamint több, különböző rendeltetésű regiszterből (akkumulátor, utasítás-, adat-, címregiszter) áll. A részegységek közötti adatáramlást cím-, adat- és vezérlővonalak biztosítják.

A mikroarchitektúra-tervezés az a folyamat, amelynek során egy számítástechnikai rendszer részegységeit létrehozzuk. Először rendre elképzeljük a működésüket és a közöttük lévő viszonyrendszert, ezután szimbólumokkal jelölve megalkotjuk a logikai, majd elektronikai kapcsolási rajzaikat (a részegységek ki- és bemeneteinek rendeltetésszerű összekapcsolásával), végül egy szimulációs szoftver környezetben ellenőrizzük működésüket.

A cél az, hogy a kialakuló rendszer képes legyen önjáró módon, egymás után következő utasításokat és adatokat előhívni egy memóriából, képes legyen ezeket az utasításokat végrehajtani és az eredményt eltárolni a memóriába.

7.2.1. Az aritmetikai-logikai egység

Az ALU bitszintű matematikai és logikai műveletek elvégzésére alkalmas kombinációs, logikai áramkör. A számítástechnikai rendszer egyik alapegysége, a programokat alkotó utasítások nagy része valamilyen aritmetikai vagy logikai művelet elvégzésére vonatkozik, ezért az ALU a CPU-nak a legtöbbet használt része.



7.10. ábra. *1 bites ALU-cella áramköri rajza és szimbóluma*

Az ALU felépítése (7.10. ábra) abból következik, hogy milyen műveleteket kívánunk elvégeztetni vele. Az összeadás egy olyan aritmetikai alapművelet, amire minden más művelet leképezhető. Ehhez egy teljes összeadócellát használunk fel (6.35. ábra). Az alapvető logikai műveleteket, a TAGADÁST (a komplemens képzés feltétele), az ÉS, valamint VAGY operációkat logikai kapuk segítségével végzi el. Ennek a három logikai műveletnek a kombinációjából bármilyen logikai függvény leképezhető (lásd a Boole-algebra alaptételeit, 6.2. ábra). A műveletvégző részeket ki kell egészítenünk egy kiválasztó logikával, amely meghatározza, hogy melyik művelet eredményét vesszük figyelembe. Ehhez egy bináris dekódoló (6.13. ábra) áramkört alkalmazunk.

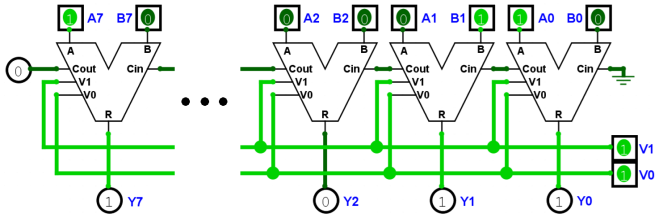
Az **A** és **B**-vel jelzett adatbemenetek eljutnak a teljes összeadó és a logikai műveletvégző alegységekbe. Figyeljük meg, hogy a tagadás művelet csak az **A** bemenetre vonatkozik. A $\mathbf{V_1}$és $\mathbf{V_0}$ műveletkiválasztó bemenetek a bináris dekódoló kimenetein létrehozzák a V_{00}, V_{01}, V_{02} és V_{03} jeleket, amelyek sorban a **TAGADÁS, VAGY, ÉS,** valamint **ÖSSZEADÁS** műveletek eredményeit engedik tovább az **Y** kimenet felé.

C_{IN}	V_1	V_0	A	B	Y	C_{OUT}
-	0	0	0	-	1	0
-	0	0	1	-	0	0
0	-	1	0	0	0	0
-	0	1	-	1	1	0
0	-	1	1	0	1	0
-	1	0	0	-	0	0
-	1	0	1	0	0	0
-	1	0	1	1	1	0
0	1	1	0	1	1	0
0	1	1	1	1	0	1
1	0	1	0	0	0	0
1	0	1	1	0	1	0
1	1	1	0	0	1	0
1	1	1	0	1	0	1
1	1	1	1	0	0	1
1	1	1	1	1	1	1

V_1	V_0	Művelet
0	0	INV A
0	1	A OR B
1	0	A AND B
1	1	A ADD B

7.11. ábra. *1 bites ALU-cella egyszerűsített igazságtáblázata*

A $\mathbf{C_{IN}}$ a teljes összeadókölcsön bemenete, a $\mathbf{C_{OUT}}$ a túlcsordulás kimenet. Az igazságtáblázat (7.11. ábra) egyszerűsítése érdekében összevontuk azokat a sorokat, ahol valamelyik bemeneti érték változása nem befolyásolja a kimenetek állapotát. Az így létrehozott univerzális, 1 bites ALU megsokszorozásával létrehozhatunk egy 8 bites ALU-t (7.12. ábra).

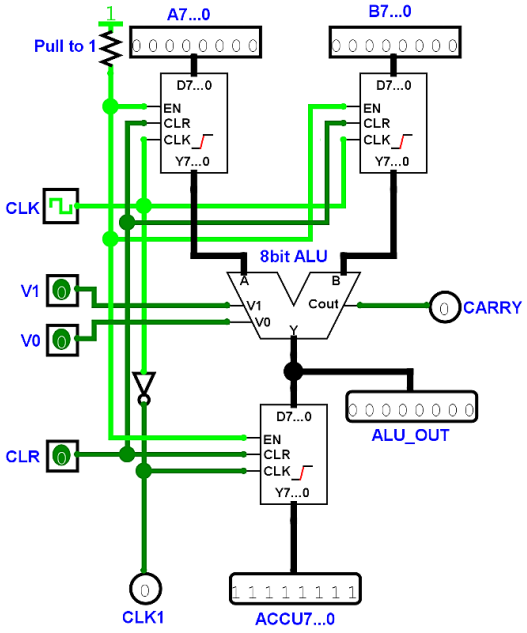


7.12. ábra. *8 bites ALU áramköri rajza*

Arra figyeljünk, hogy a helyértékek növekvő sorrendjében az egyes ALU-cellák túlcsordulás kimenetét a következő cella kölcsön bemenetével összekössük. Az utolsó cella $\mathbf{C}_{\mathbf{OUT}}$ kimenete lesz az általános **TÚLCSORDULÁS** jelzőbit.

Vizsgáljuk meg, hogyan hajtatnánk végre egy kivonás műveletet a létrehozott 8 bites ALU-val. Először a kivonandó számot kellene 2-es komplemens formába alakítanunk. Mivel az ALU csak az A bemenetét tudja tagadni, ezért a kivonandót az $A_{7...0}$ bemenetre helyezzük, és aktiváljuk a $V_0=V_1=0$ beállítást (**INV A**). A kapott $Y_{7...0}$ kimenetet, ami most az $/A_{7...0}$, visszairányítjuk az $A_{7...0}$ bemenetre. A $B_{7...0}$ bemenetre a bináris 1-es értéket állítjuk be, majd aktiváljuk a $V_0=V_1=1$ beállítást (**A ADD B**). Az $Y_{7...0}$ kimeneten megkapjuk a kivonandó érték 2-es komplemensét.

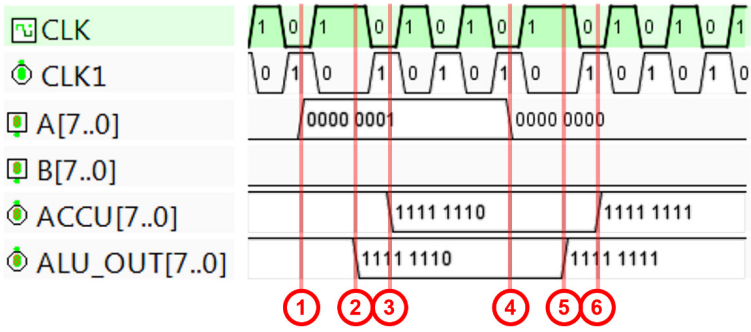
Ezt az értéket ismét visszairányítjuk az $A_{7...0}$ bemenetre. A $B_{7...0}$ bemenetre beállítunk egy számértéket, majd ismét aktiváljuk a $V_0=V_1=1$ beállítást (**A ADD B**). Az $Y_{7...0}$ kimeneten megkapjuk a kivonás eredményét (ami akár negatív, 2-es komplemens formában megjelenő érték is lehet). Az eredmény helyes, 8 bithelyen való értelmezéséhez ügyelnünk kell arra, hogy a bemeneti értékeinket úgy válasszuk meg, hogy 8 biten, 2-es komplemens formában is értelmezhetők legyenek. Látható, hogy az ALU ebben az egyszerű kivitelű formában is képes az alapműveletek elvégzésére, bár kicsit körülményesnek tűnhet a sok adatmozgatás.



7.13. ábra. *8 bites ALU bemeneti és kimeneti regiszterekkel*

- 8bit ALU design
- Intel ALU design

Ahhoz, hogy az adatokat a kívánt sorrendben tudjuk az ALU bemenetein megjeleníteni, szükségünk van egy vagy több másik alapáramkörre: a regiszterre (6.54. ábra). A regiszterek működtetéséhez további vezérlőjelekre van szükségünk: EN – beírásengedélyezés, CLK – órajel, esetleg CLR – aszinkron törlés. A 7.13. ábra az ALU bemeneteire és kimenetére illesztett (kapcsolt) regiszterek kapcsolási rajzát mutatja. A bemeneti értékeket és az elvégzendő művelet kódját az órajel 0 állapotában juttatjuk be a rendszerbe (7.14. ábra 1, a V_0=V_1=0, de a magyarázat szempontjából közömbös). Az $A_{7...0}$ és $B_{7...0}$ regiszterek az órajel felfutó élére mintavételezik a bemeneteiket, és az ALU kimenetén (ALU_OUT) az átfutási idő után megjelenik a beállított művelet eredménye (7.14. ábra 2, és EN mindig 1-es). Mivel a kimeneti regiszter ($ACCU_{7...0}$) az órajel lefutó élére kapja meg a maga felfutó élét (CLK1), az átfutási idő után mintavételezi az ALU kimenetét (7.14. ábra 3). Ekkor a kimeneti regiszterben megkaptuk az elvégzett művelet eredményét. Pár órajelciklus után újra változik a bemenet és a folyamat megismétlődik (7.14. ábra 4, 5, 6).



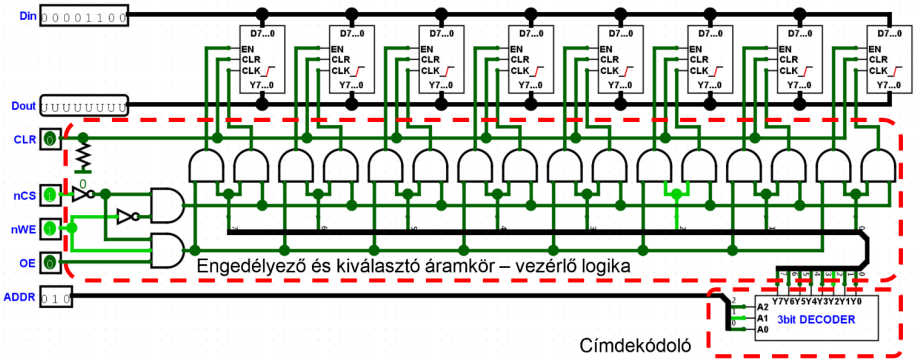
7.14. ábra. *8 bites ALU működésének idődiagramja bemeneti és kimeneti regiszterekkel*

Ezt a folyamatot kell önműködővé tenni, elérni azt, hogy az ALU folyamatosan kapjon bemenő adatokat, és az órajel ütemére elvégezze rajtuk a beállított műveletet. Ehhez az adatokat és a műveletek kódját el kell raktározzuk egy regiszterekből álló tárolóba, vagyis a memóriába. Vizsgáljuk meg, hogyan alakítható ki, a már ismert alkatrészekből, egy írható-olvasható memória.

- reverse engineering of ALU
- CPU simulator

7.2.2. A memória kialakítása és működése

Az adatbitek tárolására alkalmas, már ismert alkatrész a D tároló cellákból kialakított regiszter (6.54. ábra). A regiszterek az órajel és kimenet engedélyező bemeneteik mellett 8 adat be- és kimenettel rendelkeznek. Egy memóriamodul kialakításához több regisztert kapcsolunk ugyanazokra a be- és kimeneti adatvonalakra és vezérlőjelekre. Ahhoz, hogy különbséget tegyünk közöttük, szükség van egy bináris dekódoló áramkörre. A dekódoló a bemenetén beállított bináris érték, a CÍM alapján, csak egyetlen regiszterhez enged hozzáférést. Amíg a cím nem változik, csak erre a regiszterre lesz hatással az órajel (CLK) és a kimenetengedélyezés (EN) bemenete, ezért csak ennek a regiszternek lehet „kiolvasni" (READ) vagy „beírni" (WRITE) a tartalmát. A regiszterek kimenetei háromállapotú (three state) típusúak, így összekapcsolásuk, ameddig csak egy kimenet aktív, megengedett. A memóriamodul vezérlő bemenetei (**nCS**, **nWE**, **OE**) egy engedélyező és kiválasztó áramkörbe futnak be, ahogy azt a 7.15. ábra mutatja.



7.15. ábra. *8 bájtos memóriamodul vezérlő logikával és címdekódolóval*

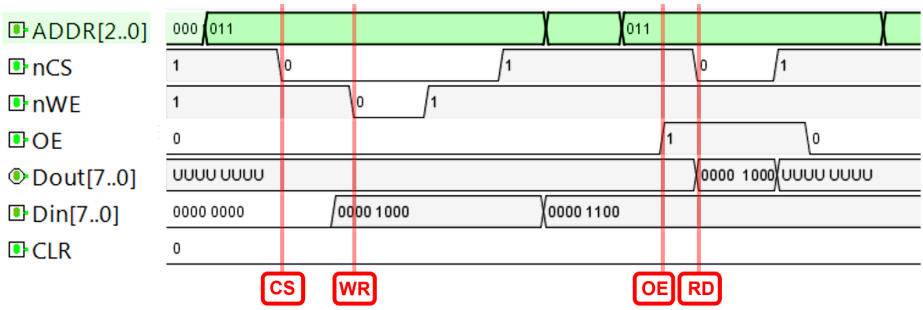
– **nCS** (negative CHIP SELECT) nullában aktív lapkakiválasztó jel. Amíg ez nem aktív (nem 0), a memóriamodul passzívan viselkedik. Abban az esetben, ha több memóriamodult is egymás mellé kapcsolunk ugyanazon cím- és adatvonalakra, az nCS segítségével különbséget tehetünk a különböző modulok között, 2^n-nel kibővítve a címtartományt, ahol n a memóriamodulok száma.
– **nWE** (negative WRITE ENABLE) nullában aktív írásengedélyező jel. Amíg ez a jel nem aktív (nem 0), a memóriamodul az adatbemenetét nem veszi figyelembe.

– **OE** (OUTPUT ENABLE) egyesben aktív kimenetengedélyező jel. Amíg ez a jel nem aktív (nem 1), a memóriamodul kimenete magas impedanciás (high Z) állapotban van.

A vezérlőlogika a címdekódoló kimeneti állapotainak függvényében egyetlen regiszterhez enged hozzáférést. A vezérlőlogika bemeneteit egy regiszter vezérlőbemeneteire vetítve a következő logikai függvények érvényesek:

$$\mathbf{EN} = (\mathbf{OE}\cdot\mathbf{WE}\cdot\overline{\mathbf{CS}})\cdot\boldsymbol{CÍM},\ \mathbf{CLR} = \mathbf{CLR},\ \mathbf{CLK} = (\overline{\mathbf{CS}}\cdot\overline{\mathbf{WE}})\cdot\boldsymbol{CÍM}$$

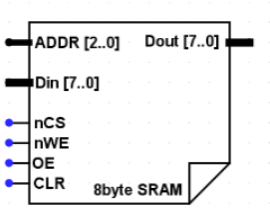
Az alább látható idődiagram (7.16. ábra) mutatja a jelzések kötött sorrendiségét egy memóriaírás (WR) és -olvasás (RD) ciklusban. Mindkét esetben elsőként az érvényes címérték (ADDR[2..0]) áll be. Ezt követi a lapkakiválasztó jelzés (nCS) beállása. Írás esetében a kimenetengedélyezés (OE) inaktív, a kimenetek magas impedanciás állapotban vannak. A bemeneti adatsínen (buszon) megjelenik az érvényes adat. Ezután az írásengedélyező jel (nWE) aktiválásával (0-ba kapcsolásával) megtörténik a bemeneten lévő adat eltárolása a kiválasztott regiszterbe. Olvasás esetén az írásengedélyező jel (nWE) inaktív. A kimenetengedélyezés (OE) aktiválásával (1-be kapcsolásával) a kiválasztott regiszterben eltárolt érték megjelenik ennek kimenetein és a memóriamodul kimenetén is.



7.16. ábra. *A memóriamodul ÍRÁS és OLVASÁS ciklusainak idődiagramja*

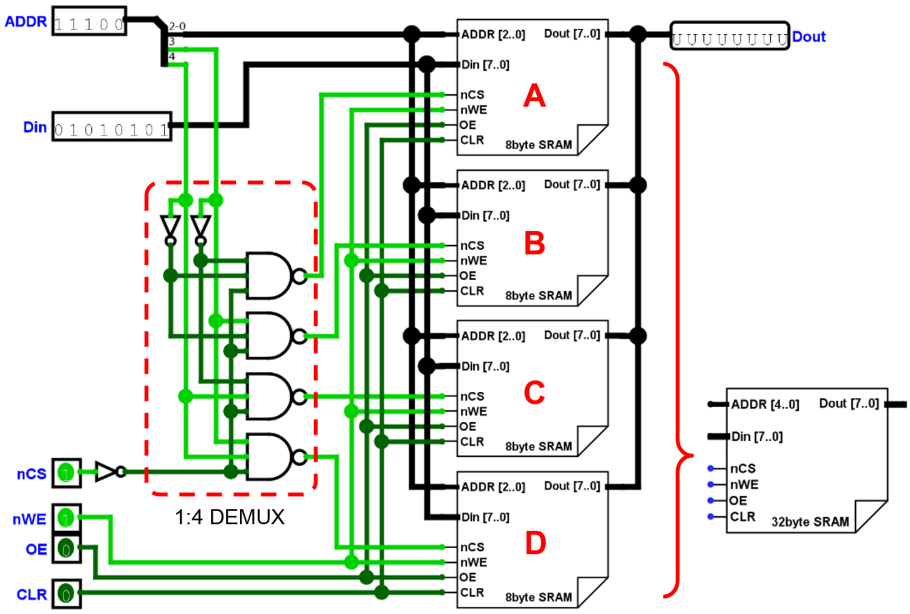
Minden memóriatípusnak megvan a saját időbeni jelsorrendisége (ezt látványosan az idődiagram jeleníti meg), amely a technológia kialakításának függvénye. Egy számítástechnikai rendszerbe integrálva a memóriamodul csak úgy tud helyesen működni, ha a rendszer figyelembe veszi és betartja a jelzések kötött sorrendiségét és időzítéseit.

A további tervezési lépések megkönnyítésére a már ismert módon elvonatkoztatunk és előállítunk egy memóriamodul-szimbólumot (7.17. ábra), amelyet egy még összetettebb, de hasonlóan működő, nagyobb tárolókapacitású memóriamodulba építünk be.



7.17. ábra. *A 8 bájtos memóriamodul szimbóluma*

Az RS-cellákból kialakított memóriákat statikus memóriáknak nevezzük (SRAM), szemben a dinamikus memória (DRAM) típusokkal, amelyeket egy későbbi fejezetben mutatunk be.



7.18. ábra. *32 bájtos memóriamodul és szimbóluma*

A nagyobb kapacitású memóriamodul (7.18. ábra) kialakításánál a címterület növelésével biztosítjuk a több, egyforma memóriamodul azonosítását. Példának a 7.15. ábra 8 bájtos moduljából négyet vonunk össze. Az eddigi 3 bites címsín 2 újabb vonallal 5 bitesre bővül. A bevezetett 2 bit egy demultiplexer áramkörbe (1:4 DEMUX) fut be és a „fő” lapkakiválasztó jel (**ncs**) irányításáról gondoskodik. A demultiplexer megvalósításához használt NAND kapuk biztosítják, hogy kapcsolás

esetén a belső memóriamodulok **n**cs bemenetei helyes jelszintet kapjanak, emiatt a „fő” lapkakiválasztó jelet tagadnunk kell. A címérték 4-es és 3-as helyértékű bitjeinek 0, 0 értékei a lapkakiválasztó jelet az A-val jelzett 8 bájtos, belső memóriamodul felé irányítja, a 0, 1 értékei a B felé, és így tovább.

A már ismert módon elvonatkoztatunk és előállítunk egy újabb memóriamodul-szimbólumot (7.18. ábra), amelyet egy még összetettebb, még nagyobb tárolókapacitású memóriamodulba építünk be.

- how computer memory works
- computer memory types

7.2.3. A memória és az ALU összekapcsolása egy utasítás-végrehajtó rendszerbe

A számítógép-architektúra két fontos részegységének, az ALU-nak és a memóriának a kialakítását és működését áttekintettük. E két alkatrész összekapcsolásához szükségünk van egy további egységre, amely egy ÓRAJEL ütemére **memóriacímeket hoz létre**, majd a létrehozott címeken eltárolt bináris kódoknak megfelelően különböző **működtető, vezérlőjeleket képez** az ALU és a memória számára. A jelek egy csoportja a memória és a regiszterek **WE**, **OE**, **EN**, **CLK** bemeneteihez kapcsolódnak, és létrehoznak közöttük egy **szakaszos adatáramlást** (az egyes lépések között ideiglenesen regiszterben tárolódnak az adatok). A jelek másik csoportja az ALU v_0, v_1 bemeneteihez kapcsolódik és kiválasztanak egy műveletet.

A memóriában tárolt bináris kódok és az őket kiegészítő, további adatbitek az UTASÍTÁSOK. Az utasítások és adatok sorozata alkotja a PROGRAMOT. A program annak függvényében fűzi össze az utasításokat és adatokat, hogy milyen műveletek sorozatát kell az elvárt eredmény érdekében elvégeztetni az ALU-val. A művelet-végzés előfeltétele sok esetben az adatmozgatás, de léteznek adatmozgatás nélküli utasítások is. A PROGRAM egy kiválasztott kódolási rendszer, a programozási „nyelv” logikáját követve hozható létre.

A „PROGRAMFUTTATÁS” során az utasítás-végrehajtó rendszer először létrehoz egy memóriacímet. Majd a címen tárolt és onnan beolvasott bináris kód alapján létrehozza mindazokat a vezérlő- és működtetőjeleket, amelyek az adott utasítás végrehajtásához szükségesek. A jelek létrejöttével teljesülnek azok az előre meghatározott logikai feltételek, amelyek az adott utasítás „végrehajtását” jelentik. Ezután az utasítás-végrehajtó rendszer újabb memóriacímet hoz létre (adott esetben az éppen végrehajtott utasítás eredményének függvényében) és a folyamat megismétlődik.

Az utasítás-végrehajtó rendszer kialakítása a cím- és adatsín szélességének meghatározásával kezdődik. Az ALU 8 bites adatokkal négy műveletet tud elvégezni (7.11. ábra), a memóriamodul 32 bájtos, 5 bites (2^5=32) címsínt igényel (7.18. ábra). Ezekből kiindulva méretezzük az adatsín szélességét 8 bitesre, a címsín szélességét 5 bitesre.

Következő lépésben meghatározzuk a végrehajtandó utasításokat. Az ALU **négy** műveletéből indulunk ki. De mivel az ALU bemeneteire és kimenetére regisztereket kapcsoltunk (7.13. ábra), meg kell oldanunk a memóriából beolvasott adatok beírását a **két** bemeneti regiszterbe, illetve a kimeneti, eredmény regiszter tartalmának átírását legalább az **egyik** bemeneti regiszterbe. Ezen kívül bevezetünk **még egy** olyan utasítást, amelynek végrehajtása során az egyik bemeneti regiszter tartalma a memória egy megadott címére íródik.

Összesen **nyolc** utasítást határoztunk meg, ezeket három biten tudjuk elkódolni. Az ALU-műveletek kódjának legnagyobb helyértéke mindig 0, az adatmozgató műveleteké **1**.

Utasítás neve	Leírás	Bemenet	Kimenet	Utasítás kódja
NOT A	az A regiszter értékének tagadása	A reg.	ACCU	**000**
OR A, B	VAGY művelet A, B regiszterek között	A, B reg.	ACCU	**001**
AND A, B	ÉS művelet A, B regiszterek között	A, B reg.	ACCU	**010**
ADD A, B	Összeadás művelet A, B regiszterek között	A, B reg.	ACCU	**011**
LDA M, A	Konstans érték betöltése az A regiszterbe	Konstans	A reg.	**100**
MVA ACC, A	Akkumulátor regiszter átmásolása az A regiszterbe	ACCU	A reg.	**101**
LDB M, B	Konstans érték betöltése a B regiszterbe	Konstans	B reg.	**110**
STORE A	az A regiszter tartalmának beírása a B regiszterbe beállított memória címre	A, B reg.	Memória	**111**

7.19. ábra. *Az utasítás-végrehajtó rendszer által értelmezett utasítások listája*

A fenti táblázatba (7.19. ábra) foglalt bináris kódok és az őket kiegészítő, további adatbitek képezik az utasításokat. Mivel a bemutatott memóriamodul 8 bites szervezésű, az eltárolt utasítások is 8 bitesek lesznek. A 8 bitből 3 bitet foglalunk el az utasítás kódjának, így 5 bit marad adatok (D_4…D0), címek (A_4…A_0) számára (7.20. ábra). Van olyan sajátos eset is, amikor az adatmező értéke közömbös (x) az utasítás végrehajtása szempontjából.

Utasítás kódja			Adatmező				
U_2	U_1	U_0	D_4	D_3	D_2	D_1	D_0
U_2	U_1	U_0	A_4	A_3	A_2	A_1	A_0
U_2	U_1	U_0	x	x	x	x	x

7.20. ábra. *A 8 bites utasítás szó szerkezete*

Nagyon fontos megjegyezni, hogy az utasítás-végrehajtó rendszer egy ÓRAJEL ütemére működik, az egymás után következő memóriacímeket az órajel ütemére, egy bináris számlálóval hozza létre. A vezérlő- és működtetőjelek is ennek az órajelnek a felfutó és lefutó éleivel szinkronban váltanak logikai állapotot (jönnek létre vagy szűnnek meg).

Példánkban az utasítás-végrehajtó rendszer egy órajelciklus alatt végzi el az utasításokat. Az órajel felfutó élére történik a memória címzése és az utasítás beolvasása, az utasítás kódjának kombinációs áramkörrel való szétbontása vezérlőjelekké, az aritmetikai és logikai műveletek elvégzése. Az órajel lefutó élére történik a belső adatmozgatás, adat vagy cím belső regiszterbe vagy külső memóriába írása. Fontos kiemelni, hogy az órajel lefutó élére az ALU kimeneti értéke beíródik a csatolt regiszterbe (ACCU).

A következő táblázat (7.21. ábra) az utasításkód alapján előállított vezérlőjeleket foglalja össze, megjelölve, hogy az órajel (CLK) felfutó vagy lefutó élére történik a „végrehajtás”, vagy mindkét él jelentőséggel bír (felfutó élre előkészítés: MOVE ACCU, ADDR SEL; lefutó élre végrehajtás: LOAD TO A, MEM WE).

Utasítás neve	Utasítás kódja	Vezérlő jelek							ÓRA JEL CLK
		ALU V0	ALU V1	LOAD to A	MOVE ACCU	LOAD to B	MEM WE	ADDR SEL	
NOT A	000	0	0	0	0	0	0	0	⌐¬
OR A, B	001	1	0	0	0	0	0	0	⌐¬
AND A, B	010	0	1	0	0	0	0	0	⌐¬
ADD A, B	011	1	1	0	0	0	0	0	⌐¬
LDA M, A	100	0	0	1	0	0	0	0	¬⌐
MVA ACC, A	101	0	0	1 ¬⌐	1 ⌐¬	0	0	0	⌐¬⌐
LDB M, B	110	0	0	0	0	1	0	0	¬⌐
STORE A	111	0	0	0	0	0	1 ¬⌐	1 ⌐¬	⌐¬⌐

7.21. ábra. *Az utasítás kódjából fakadó vezérlőjelek*

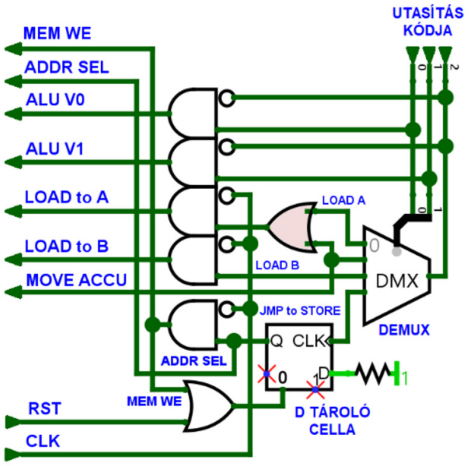
Az ALU-ból, memóriából és utasítás-végrehajtóból kialakuló rendszerben a memória kimenetét engedélyező jelet mindig aktív állapotban (1) tartjuk, így

egy cím beállítása után, ameddig ez nem változik, a memória kimenete mindig a címzett lokáció értékét tartja a kimeneti adatsínen (nincs külön utasításregiszter).

A rendszernek ez a tulajdonsága egyetlen esetben igényel kiigazítást: amikor a megadott memóriacímre adatot kiíró utasítást hajtja végre (**STORE A**). Ebben az esetben a címsínen megjelenik az írásra kijelölt memórialokáció címe, ezáltal az éppen végrehajtás alatt álló utasítás kódja elvész az utasítás-végrehajtó rendszer bemenetéről. Ezért a memóriába eltárolás utasítás (**STORE A**) címsínkiválasztó jelzésének (**ADDR SEL**) aktív állapotát (1), az utasítás-végrehajtás előkészítő fázisában (órajel felfutó éle), egy külön tárolócellába (**D**) írjuk be. A cellát a végrehajtás második fázisában (órajel lefutó éle) töröljük.

A rendszerbe kiegészítésként bevezetünk még egy általános törlés vagy alapállapotba hozó jelzést (**RST**). A jelzés aktív állapotban (1) a regiszterek, a számláló és a tárolócella tartalmát nullába állítja.

A 7.22. ábra az utasítás-végrehajtó rendszer utasításdekódoló részének kapcsolási rajzát mutatja, megjelölve ennek be- és kimeneti jelzéseit. A dekódoló az utasításkód legnagyobb helyértékű bitjét használja a műveletek megkülönböztetésére. Ha ez a bit 0, akkor logikai és aritmetikai művelet, ha 1, akkor adatmozgató művelet érvényes. A 0-ás értéket a tagadott bemenetű **ÉS** kapuk érvényesítik, csak ilyen esetben engedve tovább az utasítás kódjának két, alsóbb helyértékű bitjét, amelyek közvetlenül az ALU v_0 és v_1 bemeneteire csatlakoznak.



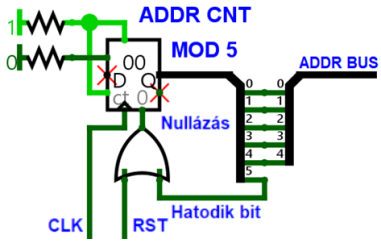
7.22. ábra. *Az utasításdekódoló és vezérlőjeleket létrehozó áramkör kapcsolási rajza*

Az adatmozgató műveletek dekódolását részben egy 1:4 demultiplexer áramkör végzi. Ennek adatbemenetére az utasításkód legnagyobb helyértékű bitjét kapcsoljuk. Így csak akkor kapunk érvényes, 1-es kimeneteket, ha ez a bithely is

1-es értékű. A vezérlőjeleket összefoglaló táblázatból (7.21. ábra) látható, hogy a LOAD TO A vezérlőjel két utasítás végrehajtásában is aktív (LDA és MVA), ezért a demultiplexer 0 és 1-es kimenetei között VAGY összefüggés van (a táblázatban színessel jelölt mezők, illetve a 7.22. ábra színezett VAGY kapuja).

Az adatmozgató műveletek második fázisát (az órajel lefutó élére) a demultiplexer kimeneteire kapcsolt tagadott bemenetű ÉS kapuk érvényesítik. Kivételt képez az előzőekben már tárgyalt, két fázisban végrehajtott STORE A utasításhoz tartozó MOVE ACCU jelzés, amely már a végrehajtás első fázisában (az órajel felfutó élére) is érvényes.

Az utasítás-végrehajtó rendszer memóriacím-előállító egysége (7.23. ábra) egy szinkron, hatbites, bináris számláló, amely az órajel felfutó élére, növekvő irányba számol. A számláló kimenetei a memória címvezetékeire kapcsolódnak. Mivel a rendszerbe kapcsolt memória 32 bájtot tárol, ezek megcímzéséhez 5 bitre van szükségünk. A hatodik kimeneti bitet a számláló nullázására használjuk VAGY összefüggésben a rendszert alapállapotba hozó jelzéssel (RST).



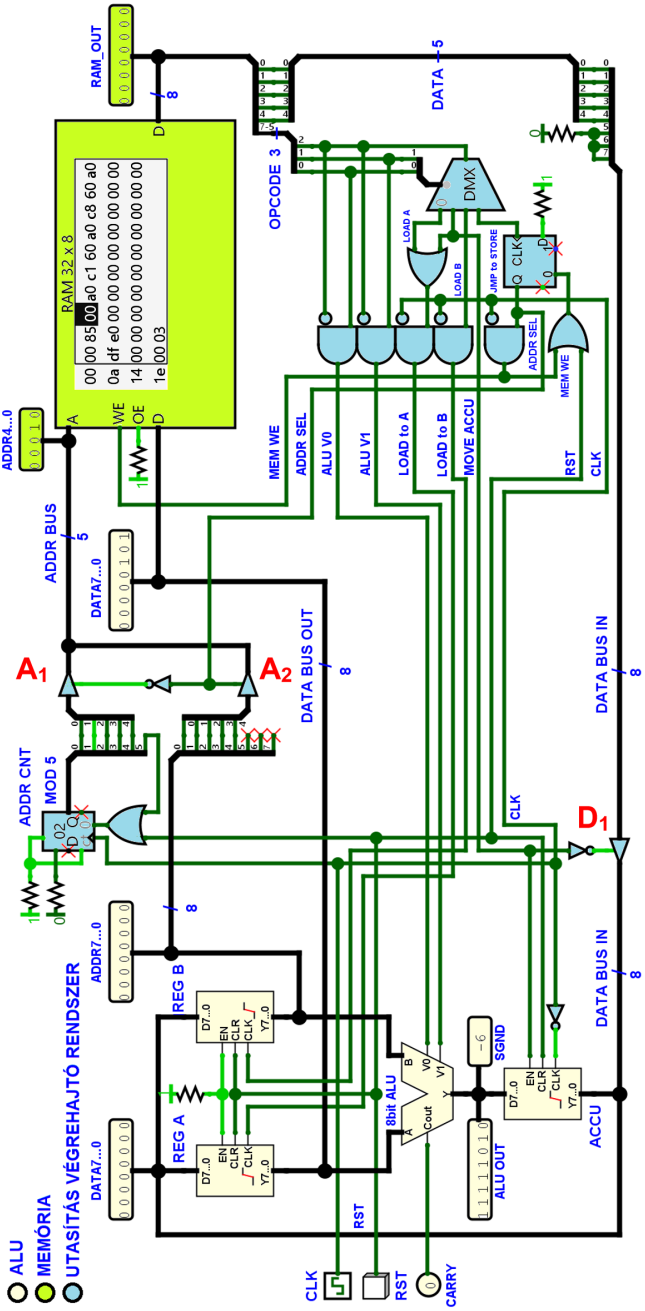
7.23. ábra. *A memóriacím-előállító, bináris számláló kapcsolási rajza*

7.2.4. Egyszerű mikroarchitektúra kialakítása és működése

A számítógép-architektúra három fontos egységének, az ALU-nak, a memóriának és a vezérlőáramkörnek a kialakítását és működését áttekintettük. Kapcsoljuk össze ezt a három egységet adat- és címsínekkel, vezérlőjelekkel, kialakítva ezzel egy mikroarchitektúrát.

A 7.24 ábra a mikroarchitektúra egészének kapcsolási rajzát szemlélteti, azonos színekkel jelölve az adott egységhez tartozó kapukat és komplexebb áramköröket: az ALU-egység sárgás, az utasítás-végrehajtó rendszer kékes, míg a memória zöld színt kapott. Az egységek összekapcsolásakor figyelnünk kell a kimeneti és bemeneti adatbitek számára, ezekhez kell igazítanunk az adat és a címsín szélességét.

A fő adatsín kiindulási pontja a memória adat kimenete (DATA BUS IN). Ebből ágazik le a felső három helyértékű bit, amely az utasítás-végrehajtó rendszer utasításdekódoló áramkörének bemenetére kerül. A továbbhaladó adatvonalakat az utasításkód bitjeinek helyén 0-ásokkal egészítjük ki, így érkeznek meg az ALU-t körülvevő regiszterekhez, ahol az ACCU kimenetére, majd a REG A és REG B



7.24. ábra. *Egyszerű mikroarchitektúra kapcsolási rajza*

bemenetére csatlakoznak. Az adatsín az ACCU regiszter kimeneteinek csatlakozási pontja előtt feltételesen, engedélyező sínmeghajtó buffer közbeiktatásával megszakítható. Tárgyaltuk, hogy a memória kimenete folyamatosan engedélyezett állapotban van (nincs külön bemeneti adatregiszter). Így a leválasztással az ACCU regiszter kimenete ütközés nélkül továbbkapcsolható a REG A regiszter bemenete felé. A REG A regiszter kimenete másodlagos adatsínként (DATA BUS OUT) is működik. A fő és másodlagos adatsínek ellenütemben működnek a MOVE ACCU vezérlőjel hatására.

A REG B regiszter kimenetei egy másodlagos, 5 bites címsínben folytatódnak (ez lesz majd az adatmentésre kiválasztott memórialokáció címe), amely vonalak feltételesen, engedélyező sínmeghajtó buffer közbeiktatásával kapcsolódnak a memória címvonalaira.

A fő címsín a számláló kimenetét kapcsolja össze, szintén feltételesen, a memória címvonalaival. A fő és a másodlagos címsínek ellenütemben működnek az ADDR SEL vezérlőjel hatására.

A REG A és REG B regiszterek kimenetei folyamatosan engedélyezett állapotban vannak, a beírt adatok azonnal megjelennek az ALU bemenetein, valamint a másodlagos adat- és címsíneken.

A mikroarchitektúra működését egy rövid, gépi kódú program végrehajtásának elemzésével mutatjuk be. Végezzünk el egy kivonás műveletet két pozitív, természetes, egész szám között, ahol a kisebbítendő nagyobb, mint a kivonandó (az eredmény pozitív, egész szám lesz), majd mentsük el a művelet eredményét a memória utolsó megcímezhető lokációjába. Mivel az ALU csak összeadás műveletet tud elvégezni, a kivonást a kivonandó 2-es komplemens formába alakításával kezdjük, majd a kapott eredményt hozzáadjuk a kisebbítendő értékhez. A kialakított mikroarchitektúrával összefüggésben értelmezett program a következő lépésekre bontható:

– Töltsük be a memóriából a kivonandó értéket (5) a REG A jelzésű regiszterbe. Ilyenkor a számláló kimenete az **A1** sínmeghajtón keresztül a memória címvonalaira kapcsolódik, az adatsín a **D1** sínmeghajtón keresztül a REG A bemeneteire kapcsolódik. A REG A CLK bemenetére kapcsolódó LOAD TO A vezérlőjel az órajellel szinkronban, annak lefutó élére, a regiszterbe írja az adatsínen érvényes vonalállapotokat.
– Az ALU vezérlő bemeneteit állítsuk a $V_0=V_1=0$ állapotba (ALU **A** bemeneti értékének tagadása – NOT A).
– Az ACCU regiszter tartalmát töltsük át az REG A regiszterbe. Ilyenkor a D1 sínmeghajtó kimenetei magas impedanciás állapotba kapcsolnak a MOVE ACCU vezérlőjel tagadott állapotára, kialakítva ezzel a másodlagos adatsínt. A REG A CLK bemenetére kapcsolódó LOAD TO A vezérlőjel az órajellel szinkronban, annak lefutó élére, a regiszterbe írja az adatsínen érvényes vonalállapotokat.

Kód mező	Adat mező	Assembly utasítások	Gépi kód	Memória cím
000	0 0000	Program kezdete	00h	000h
100	0 0101	LDA 05, A	85h	001h
000	x xxxx	NOT A	00h	002h
101	x xxxx	MVA ACC, A	A0h	003h
110	0 0001	LDB 01, B	C1h	004h
011	x xxxx	ADD A, B	60h	005h
101	x xxxx	MVA ACC, A	A0h	006h
110	0 1000	LDB 08, B	C8h	007h
011	x xxxx	ADD A, B	60h	008h
101	x xxxx	MVA ACC, A	A0h	009h
110	1 1111	LDB 1F, B	DFh	010h
111	x xxxx	STORE A	E0h	011h
000	0 0000	Program vége	00h	012h

7.25. ábra. *Kivonás művelet programja memóriába való eredménykiírással*

– Töltsük be a memóriából az 1-es értékű állandót (konstanst) a REG B jelzésű regiszterbe.
– Az ALU vezérlő bemeneteit állítsuk a $V_0=V_1=1$ állapotba (A összeadása B-vel – ADD A, B), megkaptuk a kivonandó 2-es komplemens formáját (negatív szám).
– Az ACCU regiszter tartalmát töltsük át a REG A regiszterbe.
– Töltsük be a memóriából a kisebbítendőt (8) a REG B regiszterbe.
– Az ALU vezérlő bemeneteit állítsuk a $V_0=V_1=1$ állapotba (A összeadása B-vel – ADD A, B), megkaptuk a kivonás eredményét (8 - 5 = 3).
– Az ACCU regiszter tartalmát töltsük át a REG A regiszterbe, ezzel a másodlagos adatsínen előkészítettük a memóriába kiírandó eredményt.
– Töltsük be a memóriából az eredmény elmentésére szánt memórialokáció címét (1Fh) a REG B regiszterbe, ezzel a másodlagos címsínen előkészítettük az eredmény beírására szánt memórialokáció címét.
– Írjuk be az eredményt a REG B regiszterben beállított memóriacímen lévő lokációba. Ilyenkor a másodlagos címsín aktiválása történik, a REG B kimenete az **A2** sínmeghajtón keresztül a memória címvonalaira kapcsolódik, miközben az **A1** sínmeghajtó kimenetei az ADDR SEL vezérlőjel hatására magas impedanciás állapotba kerülve leválasztják az elsődleges címsínt a memória címvonalairól. Mivel a kísérleti architektúra nem használ külön utasításregisztert, az elsődleges címsín lekapcsolásával az utasításdekódoló és végrehajtó rendszer bemenetéről eltűnik az aktuális utasítás kódja. Ezért a beíró utasítás bináris kódjából származó, elsődleges jelzést egy D tárolócellába mentjük el. Az órajel lefutó élével szinkronban létrejövő MEM WE vezérlőjel

hatására megtörténik a REG A regiszterben tárolt értéknek a REG B regiszterben beállított címen lévő memórialokációba való beírása. Ugyanekkor a D tároló is törlődik, és a memória címsínje újra az elsődleges címsínre kapcsolódik.

A meghatározott utasításkészletből (7.19. ábra) felírható a fent részletezett program és a mikroarchitektúrán futtatható gépi kódú (hexadecimális számjegyekből álló) változata.

7.2.5. Az egyszerű mikroarchitektúra működésének korlátai

A kísérleti mikroarchitektúra-típus besorolását vizsgálva megállapítható, hogy a Neumann-elvnek megfelelően felépített, 2 című gép. A kevés utasításszámot (8), azonos utasításszóhosszat (8bit) és azonos végrehajtási ciklusidőt (1 órajel ciklus) tekintve egy MISC-rendszer sajátosságait mutatja.

A szimulációs modell működését elemezve megállapítható, hogy egyszerű kombinációs és szekvenciális áramkörök összekapcsolásával kialakítható egy „önjáró", az órajellel szinkronban működő logikai áramkör, amely képes egy memóriában tárolt program utasításait két ütemben, sorban végrehajtani.

Továbbá megállapítható, hogy a külön utasítás és adatregiszter hiánya megbonyolítja a kivonás algoritmus kivitelezését, amely így nagyban támaszkodik az egymás után sorjázó utasításokba kódolt vezérlőjelek létrejöttére. Az is látható, hogy bonyolultabb utasítások egy órajelciklus alatt történő végrehajtása nem lenne lehetséges. Elképzelhető olyan utasítás, amelynek végrehajtása során többször is kapcsolna az elsődleges cím- és adatsínek között, vagy olyan, amely előző utasítás végrehajtása során teljesült logikai feltételtől függő végrehajtást igényel.

Az egyszerű megközelítés jelentős hátránya, hogy a rendszer csak állandó értékeket (konstansokat) tud betölteni belső regisztereibe. A tetszőleges memóriacímen elérhető értékek vagy változók betöltése cím- és adatregiszter hiányában nem megoldható, hasonlóan az eltárolásuk sem lehetséges. A több, különböző feladatot ellátó regiszter működtetése több vezérlőjelet igényel, és adott esetben ezek időben is távolabb követik egymást. Ezért a bonyolultabb, értsd: több vezérlőjelet igénylő, végrehajtási láncokat a tárgyalt, egyszerű mikroarchitektúrán nem, vagy csak nagyon nehezen lehetne megvalósítani.

- program counter
- microarchitecture explained

8. Központi végrehajtó egység (CPU) tervezése

8.1. Az utasításkészlet, a fő alkotóelemek és az utasítás szó szerkezete

Az előző fejezetben tárgyalt mikroarchitektúra képes volt önjáró módon, egymás után következő utasításokat és adatokat előhívni egy memóriából, az utasításokat végrehajtani és az eredményt eltárolni a memóriába. De nem volt megoldott benne a memóriacímről történő adatbeolvasás és nem volt általánosan felhasználható részegységekre bontva, ezért nem is volt bővíthető további belső vagy külső egységekkel (pl. veremvezérlő vagy ki- és beviteli eszköz). Ezért a következő fejezetekben egy olyan számítástechnikai rendszert tervezzünk, amely szétválasztja a mikroarchitektúra egyes funkcionalitásait, és különálló egységeket alkot ezekből. Ezek az egységek a későbbiek során, más egységekkel kombinálva, egy paramétereiben és funkcionalitásában bővíthető számítástechnikai rendszert képeznek majd.

A központi végrehajtóegység tervezését a végrehajtani kívánt utasítások csoportjának meghatározásával kezdjük. Elképzeljük és táblázatos formában (8.1. ábra) rögzítjük az utasítások kívánt hatását is.

Utasítás neve	Leírás	Bemenet	Kimenet	Utasítás kódja
LOAD cím	Adat betöltése az ACCU-ba megadott címről	**DR**	**ACCU**	**0000**
STORE cím	ACCU tartalmának kiírása megadott memória címre	**ACCU**	**DR**	**0001**
ADD cím	ACCU-hoz adja a megadott címről a DR-be betöltött értéket	**DR**	**ACCU**	**0010**
AND cím	ACCU **ÉS** a megadott címről a DR-be betöltött érték	**DR**	**ACCU**	**0011**
JMP cím	Feltétel nélküli ugrás megadott címre	**DR**	**AR**	**0100**
JZ cím	Feltételes ugrás megadott címre (ha ACCU = 0)	**DR**	**AR**	**0101**
NOT	ACCU tartalmának bitszintű tagadása	**ACCU**	**ACCU**	**0110**
RSH	ACCU tartalmának jobbra tolása egy pozícióval	**ACCU**	**ACCU**	**0111**

8.1. ábra. *A processzor által végrehajtani kívánt utasítások listája*

Az egyes utasításoknak egymástól eltérő neveket adunk a hatásukat minél tömörebben és pontosabban leíró szavakkal vagy rövidítésekkel, más néven mnemonikokkal (az adott utasítás hatására emlékeztető betűkombinációkkal) jelöljük. Ezeket az elnevezéseket a későbbi programíráshoz meg kell jegyeznünk.

Minden egyes utasításnak egy bináris számsort feleltetünk meg. Eddig 8 utasítást határoztunk meg (3 biten kódolható), de szem előtt tartjuk a későbbi bővítés lehetőségét is, ezért 4 bitet szánunk a kódolásukra (16 lehetséges utasítás). A táblázatban elfoglalt helyük szerint számozzuk őket (léteznek bonyolultabb, több bitet használó, esetleges bithibákat kivédeni képes számozási módszerek is).

Az utasításkészlet összeállítása és a központi végrehajtóegység architektúrájának kialakítása kölcsönösen függ egymástól. Az is fontos szempont, hogy a 7.2.4-es alfejezetben tárgyalt mikroarchitektúránál általánosabb, hatékonyabban működő rendszert hozzunk létre.

Az adatmozgató utasításokat, a betöltő LOAD-ot és az eltároló STORE-t úgy képzeljük el, hogy a rendelkezésre álló teljes memória címterületről képesek legyenek adatot betölteni, illetve eltárolni.

A logikai és aritmetikai műveletek meghatározásánál a Boole-algebra azonossági képleteire gondolva (6.1-es fejezet) választjuk ki az AND (ÉS) és NOT (TAGADÁS) függvényeket. Tudjuk, hogy NAND (NEM-ÉS) függvényből minden más alapfüggvény képezhető. Például a VAGY művelet:

$$Y = \overline{A} \cdot \overline{B} = A + B$$

A NAND utasítás helyett viszont a különálló NOT és AND függvények megvalósítását választjuk. Arra gondolunk, hogy a kivonás művelethez szükséges 2-es komplemens képzéshez szükségünk van egy számérték bitjeinek tagadott formájára is. Ennek a számértéknek a kialakítását és tárolását **egyetlen** regiszterben képzeljük el. Viszont a NOT függvény képzéséhez szükséges, két azonos bemenetű értéket használó NAND megvalósításához **két**, azonos értékkel feltöltött regiszterre lenne szükségünk.

Ezekhez társítjuk az összeadó, ADD utasítást. A kivonás műveletet a kivonandó érték 2-es komplemens formába való átalakításával, majd a kisebbítendő értékhez való hozzáadással végezzük majd el.

A programvezérlő utasítások nélkülözhetetlenek a ciklikus algoritmusok vagy programrészek megvalósításához. Gondoljunk itt egy tetszőleges számérték (változó) ciklikus növelésére vagy csökkentésére. Ilyenkor arra van szükségünk, hogy az a csökkentés vagy növelés művelet egy bizonyos logikai kondíció bekövetkeztéig, például a 0 érték eléréséig ismételjük, utána pedig haladjunk tovább a program következő utasítására. Ehhez a memóriában eltárolt program különböző utasításai között kell „ugrálnunk”, vagyis különböző címekről kell utasításokat végrehajtanunk. A memória-címértékek közötti „ugráláshoz” használjuk a JMP, feltétel nélküli ugrás és a JZ (JUMP IF ZERO), feltételes ugrás (ha egy előző művelet eredménye nulla volt) utasításokat.

A jobbra eltolás, RSH (RIGHT SHIFT) művelet a szorzás algoritmus megvalósításához szükséges, de a 2 hatványaival való, maradék nélküli osztás művelet elvégzésére is alkalmas.

Az utasítások táblázatának további kitöltéséhez szükségünk van arra, hogy elképzeljük és szintén táblázatos formában (8.2. ábra) rögzítsük a központi végrehajtóegység alkotóelemeinek nevét és azok működés közbeni szerepét (az alkotóelemeket a mikroarchitektúra létrehozásakor már megismertük).

Megnevezés	Sín szélesség	Magyarázat
ALU	**8 bit**	**A**rithmetical **L**ogical **U**nit – matematikai és logikai műveleteket végző egység
ACCU	**8 bit**	**ACCU**mulator – munka regiszter, műveletek eredmény regisztere, operandus tároló
PC	**8 bit**	**P**rogram **C**ounter – program számláló, órajellel szinkronban működő, memória cím előállító egység
IR	**4 bit**	**I**nstruction **R**egister – utasítás regiszter, a vezérlő egység bemenete, utasítás kód tároló
AR	**8 bit**	**A**ddress **R**egister – címregiszter a processzorhoz kapcsolt külső elemek kiválasztásához
DR	**12 bit**	**D**ata **R**egister – adatregiszter, ide kerül a megadott címről beolvasott adat
Z	**1 bit**	**Z**ero Flag – Zéró jelzőbit, a munkaregiszter tartalmának kiértékelése: ha ACCU = 0, akkor a jelzőbit 1-es
CU	**-**	**C**ontrol **U**nit – vezérlő egység, az utasítások végrehajtásához szükséges jeleket előállító áramkör

8.2. ábra. *A processzor fő alkotóelemeinek listája*

Ahhoz, hogy az alkotóelemek táblázatát kiegészíthessük az egyes alkotóelemekhez tartozó cím- és adatsínek szélességére vonatkozó adatokkal, meg kell határoznunk az utasításszavak szerkezetét, vagyis azt a bináris számsort, amelyet a továbbiakban a központi végrehajtóegység használni fog. 4 biten kódoltuk az utasításokat. Ha ehhez egy 8 bites cím- vagy adatmezőt kapcsolunk, egy 12 bites sorozatot kapunk.

A 8.3. ábra két 12 bites sorozatot különböztet meg. A táblázat első sorában olyan utasítás általános leírása látható, amely egy változóra utaló címet hordoz, amelyet az adatregiszterbe kell betölteni (pl. LOAD, ADD), vagy az adatregiszterből kell a memóriába kiírni (pl. STORE), vagy amelyet a programszámlálóba kell betölteni (pl. JMP, JZ). A második típusú utasítás nem hordoz értéket az utasítás kódja után következő mezőben, ez a munkaregiszteren (ACCU) végez közvetlen műveletet (pl. NOT, RSH).

Utasítás kódja				Adatmező							
U_3	U_2	U_1	U_0	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
U_3	U_2	U_1	U_0	x	x	x	x	x	x	x	x

8.3. ábra. *A 12 bites utasításszó szerkezete*

Az utasítások kifejtett hatásukat tekintve különböző csoportokba sorolhatók. Vannak **adatmozgató utasítások**, amelyek a processzorhoz kapcsolódó memória írásáért és olvasásáért felelnek. Ezek segítségével töltünk be egy változót a processzor adatregiszterébe vagy írunk ki ugyaninnen a memóriába (pl. LOAD, STORE), vagy csak bithelyes mozgatást végzünk a munkaregiszteren (pl. RSH). Az **aritmetikai-logikai utasítások** (pl. ADD, AND) matematikai vagy logikai műveletet végeznek két, előzőlegesen betöltő utasítással a processzor belső regisztereibe elhelyezett változó között vagy a munkaregiszteren (pl. NOT). A **programvezérlő utasítások** a programszámláló értékét módosítva közvetlenül avatkoznak bele az utasítások végrehajtási sorrendjébe (pl. JMP), vagy csak valamilyen logikai kondíció teljesülése esetén (pl. JZ).

Ezeket tudva visszatérhetünk a processzor fő alkotóelemeinek listájához (8.2. ábra), és kitöltjük az egyes alkotóelemek sínszélességigényét. Az utasításregiszternek (IR) 4 bitre lesz szüksége, az aritmetikai-logikai egységnek (ALU), a hozzá kapcsolódó munkaregiszternek (ACCU), a címregiszternek (AR) és a hozzá kapcsolódó programszámlálónak (PC) egyformán 8 bitre lesz szüksége. Az adatregiszter (DR) a teljes utasításszót kell ideiglenesen tárolja, így ehhez 12 bitre lesz szüksége. A munkaregiszter tartalmának 0 értékbe állását jelző bit (Z) egyetlen helyértéket foglal el.

A továbbiakban visszatérhetünk a processzor által végrehajtani kívánt utasítások listájához (8.1. ábra), és kitölthetjük táblázatot az utasítások végrehajtása során kimenetként és bemenetként használt regiszterek neveivel.

Miután rögzítettük a processzor utasításkészletét, működésének feltételeit és alkotóelemeit, meg kell tervezni az egyik legfontosabb részét, az utasítások végrehajtásához nélkülözhetetlen vezérlőegységet.

- instruction set
- components of a microprocessor

8.2. Az utasítás-végrehajtás menetének felvázolása

A vezérlőegység a processzornak az a része, amely a tulajdonképpeni utasítás-végrehajtás folyamatot lebonyolítja. Idézzük fel, hogy a processzor belső felépítését tekintve regiszterekből és azokat összekötő adatsínekből áll. A regiszterek közötti, szakaszos, ütemezett adatforgalmat irányító, az ALU által elvégzett műveletek eredményét beállító és a memória felé irányító logikai rendszer a vezérlőegység.

Emlékezzünk arra is, hogy általánosan egy utasítás végrehajtási lépései a következők: beolvasás a memóriából, előkészítés (utasításhoz tartozó adat beolvasása a memóriából), utasításértelmezés vagy -végrehajtás, eredmény visszaírása a memóriába.

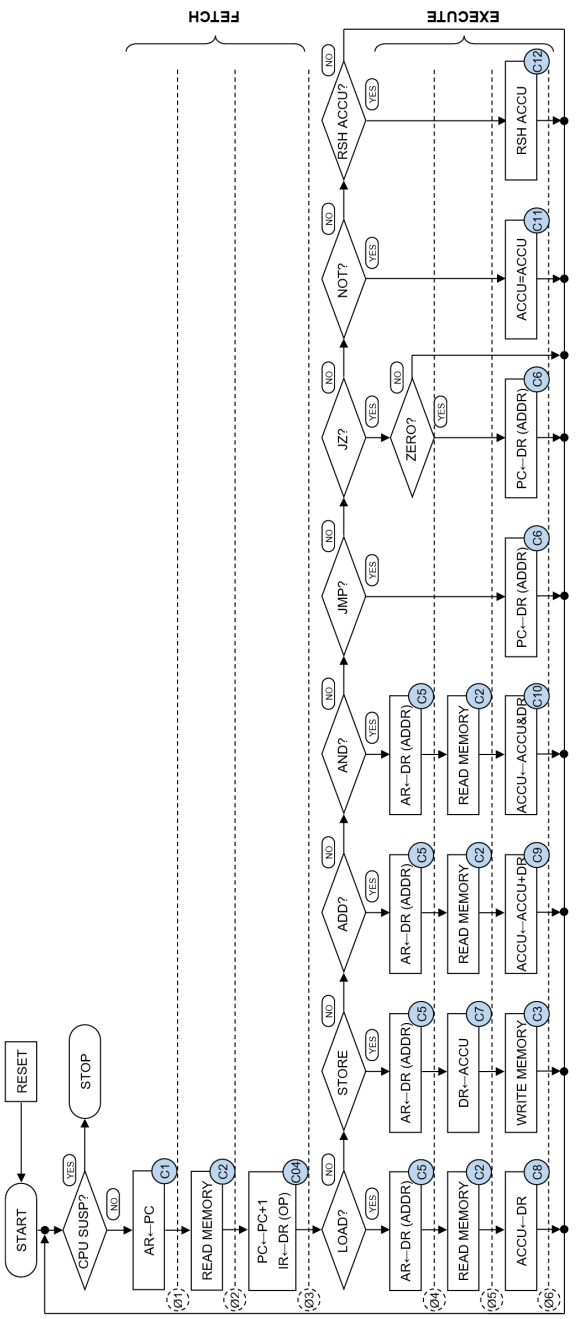
A vezérlőegység a processzor legtöbb alkotóelemével közvetlen jelkapcsolatban áll, vagyis azok felé logikai jeleket tud küldeni (engedélyező, beállító bitek), illetve azoktól logikai jeleket tud fogadni (jelzőbitek) az éppen végrehajtás alatt álló utasítás függvényében.

A vezérlőegység által a processzor alkotóelemei felé kiküldött vagy onnan beérkező jelzések szakaszosan ütemezve, az órajel fel- és lefutó éleihez szinkronizálva jelennek meg és tűnnek el. Ahhoz, hogy ezeket az ütemeket az egyes utasítások végrehajtásával összefüggésbe hozzuk, és a processzor alkotóelemei felé kiküldött vagy onnan beérkező jelzések időbeliségét meghatározzuk, szükségünk lesz egy folyamatábrára.

Ez a folyamatábra (8.4. ábra) az egyes utasítások végrehajtási menetét írja majd le az általunk elképzelt módozatot és sorrendiséget betartva. Kezdetben el kell képzelnünk, hogy milyen módon hajtanánk végre az egyik leggyakoribb utasításunkat, a memóriából a processzor belső adatregiszterébe adatot betöltő LOAD-ot. Gondoljunk arra, hogy a memória egy címalapú tárolóeszköz. Ahhoz, hogy elérjünk és betöltsünk egy benne tárolt adatot, létre kell hoznunk egy címértéket (address), és ezt a memória címvezetékein meg kell tartanunk mindaddig, amíg az adatsínen megjeleníti a beállított címen eltárolt adatértéket (data). A soron következő címértéket a programszámláló (PC) önműködő módon előállítja (a későbbiekben megtudjuk, hogyan teszi ezt). A vezérlőegység ekkor jelzést küld a címregiszternek (AR), amely a programszámláló (PC) kimenetén megjelenő értéket rögzíti (latch). A beolvasáshoz a címérték mellett szükségünk van a memória felé az olvasás szándékát közvetítő jelzésre is (RD – read). A vezérlőegység a megfelelő ütemben erről is gondoskodik.

Ezután nagyon fontos, hogy a vezérlőegység az utasítás-végrehajtás folyamatosságának fenntartásáról is gondoskodjon. Ezt a programszámláló (PC) 1-gyel való növelésével éri el.

Miután az egy ütemben a memóriából beolvasott adat rendelkezésünkre áll az adatregiszterben (DR), gondoljunk arra, hogy mielőtt értelmezni próbáljuk a kapott értéket, ezt át kell helyezzük az utasításregiszterbe (IR), különben előfordulna, hogy a végrehajtáshoz szükséges adatérték a következő ütemben, a memóriából való betöltésekor, az adatregiszterben lévő utasításszót felülírná, így a későbbi értelmezéshez ez már nem állna rendelkezésre.



8.4. ábra. *A tervezett utasítások végrehajtási folyamatábrája*

Az elsődlegesen betöltött adatértéknek az utasításregiszterbe való áthelyezése után, az így kapott („szűrt") utasításkód alapján, a vezérlőegység a végrehajtás következő ütemeiben további vezérlő jeleket hoz létre. Az adatbetöltő (LOAD) utasítást vizsgálva a további ütemekben a vezérlőegység az utasításszóba foglalt címértékről adatot tölt be a memóriából (a már ismertetett módon és jelzések létrehozásával, egy ütemben), majd ezt az értéket a munkaregiszterben (ACCU) helyezi el.

Bejártuk tehát a folyamatábra (8.4. ábra) első oszlopát. Az oszlop tetején látható jelzések (START, STOP) a végrehajtási folyamat indítását és belső hiba (pl. veremtúlcsordulás – a későbbiekben kerül bemutatásra) esetén (CPU SUSP) a leállításának eseménysorát rögzítik. A nyílban végződő vonalak az események bekövetkezési sorrendjét jelzik.

Megfigyelhető, hogy a folyamatábra 6 eseményt ($ø_1$-től $ø_6$-ig jelzett téglalapok) rögzít, az első három között pedig egy döntés (rombusz) jelez. Az előzőekben leírt magyarázat szerint az első három esemény az utasításbeolvasás (FETCH) lépéseinek felel meg.

Ezután eldönthető, hogy milyen utasításról van szó, és következő három esemény során lezajlik a végrehajtása (EXECUTE). Mivel a tárgyalt folyamat alatt a program számlálóértéke eggyel megnőtt, a végrehajtás befejeztével újabb utasítás töltődik be, és a folyamat megismétlődik. A vezérlőegység tehát ezt a folyamatot kell fenntartsa, és mindig 6 ütemben ($ø_1$-től $ø_6$-ig) végrehajt egy-egy soron következő utasítást. Mivel az utasítás-végrehajtás folyamatában az első 3 esemény minden esetben bekövetkezik, a folyamatábra további oszlopai csak a döntés alatt folytatódnak.

8.2.1. A vezérlőjelek meghatározása és számbavétele

Az utasítás-végrehajtás folyamatában az események (áramköri reakciók) kiváltásához jelzésekre van szükség. Ezeket vezérlőjeleknek (control signal) nevezzük, és a vezérlőegység hozza létre őket. A vezérlőjelek az utasítás-végrehajtás folyamatának egyes szakaszait szinkronizálják. A vezérlőjelek mindig egy kötött sorrend szerint jelennek meg és tűnnek el (olykor más jelekkel, például órajel fel- vagy lefutó élével szinkronban). A megjelenés és eltűnés alatt a jelállapot változását értjük, a jelnek az adott áramköri kapcsolásban kifejtett szerepe szerinti aktív és passzív szintértékeit.

A folyamatábrából kiindulva 13 különböző eseményt számolhatunk össze. Ezeket tehát megszámozzuk (c_0-tól c_{12}-ig), és egy táblázatba (8.5. ábra) foglalva, további elemzésre összegyűjtjük. Ezek közül két esemény, a c_0 és c_4 jelzésű párhuzamos módon (azonos időpillanatban) zajlik, ezért a folyamatábrában őket egy téglalap szimbólumban, összevonva ábrázoljuk.

A táblázatban megjelöljük az adott vezérlőjel által kiváltott adatmozgás irányát és mibenlétét, valamint röviden magyarázzuk a jelenséget.

Vezérlőjel szükséges a programszámláló növeléséhez vagy az ebbe való érték elhelyezésekor (c_0, c_6), ugyanúgy a cím, adat és utasítás regiszterekbe való beíráshoz (latch, c_1, c_7, c_4), a memóriának jelezni kell, hogy írás vagy olvasás művelet kezdődik (c_3, c_2). A munkaregiszter (ACCU) feltöltéséhez (latch) az adatregiszterből, vagy az alu műveletvégzés kiválasztása után, szintén több vezérlőjelre van szükség (c_8, c_9, c_{10}, c_{11}, c_{12}). Ugyanígy külön jel aktiválja a munkaregiszter aktuális értékének logikai tagadását (c_{11}).

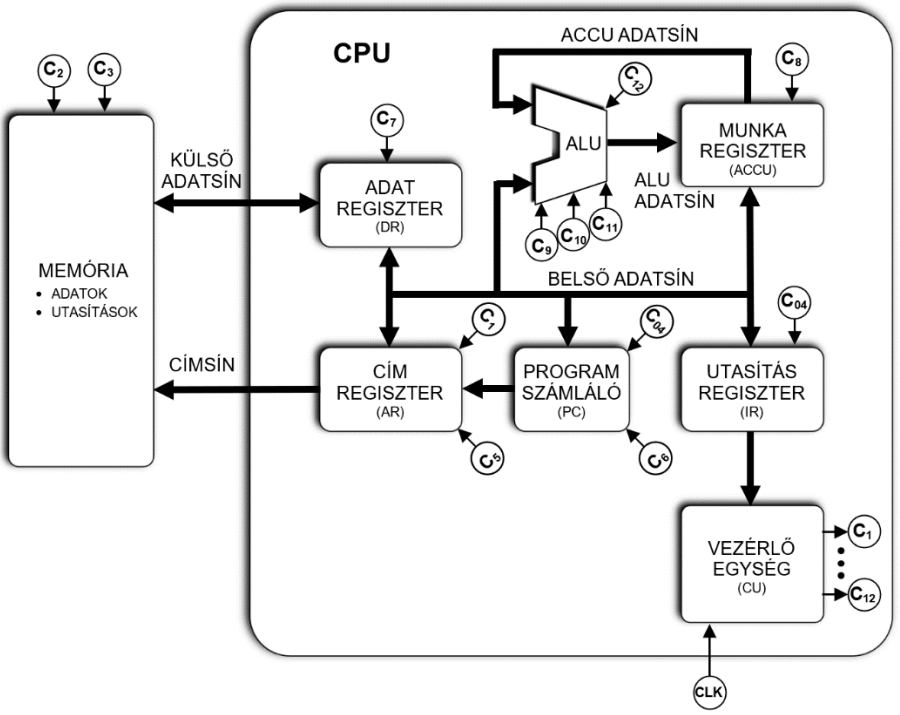
Vezérlő jel	Adat mozgás	Magyarázat
C_0	**PC←PC+1**	Programszámláló (PC) értékének 1-el való növelése
C_1	**AR←PC**	Programszámláló kimenti értékének beírása a címregiszterbe (AR)
C_2	**READ MEMORY**	Memóriaolvasás, a memória kimeneti adatértékének beírása az adatregiszterbe (DR)
C_3	**WRITE MEMORY**	Memóriaírás, az adatregiszterben tárolt érték beírása a memóriába
C_4	**IR←DR (OP)**	Az adatregiszterben tárolt érték egy részének (felső 4 bithely) beírása az utasításregiszterbe (IR)
C_5	**AR←DR (ADDR)**	Az adatregiszterben tárolt érték egy részének (alsó 8 bithely) beírása a címregiszterbe
C_6	**PC←DR (ADDR)**	Az adatregiszterben tárolt érték (alsó 8 bithely) beírása a programszámlálóba
C_7	**DR←ACCU**	A munkaregiszter (ACCU) tartalmának beírása az adatregiszterbe (alsó 8 bithelyre)
C_8	**ACCU←DR**	Az adatregiszterben tárolt érték egy részének (alsó 8 bithely) beírása a munkaregiszterbe
C_9	**ACCU←ACCU+DR**	Az adatregiszterben és a munkaregiszterben tárolt értékek összeadása. Az eredmény a munkaregiszterben tárolódik.
C_{10}	**ACCU←ACCU&DR**	Az adatregiszterben és a munkaregiszterben tárolt értékek közötti **ÉS** függvény képzése. Az eredmény a munkaregiszterben tárolódik.
C_{11}	**ACCU =! ACCU**	A munkaregiszterben tárolt érték tagadása. Az eredmény a munkaregiszterben tárolódik.
C_{12}	**RSH ACCU**	A munkaregiszterben tárolt érték jobbra tolása 1 bithellyel. Az eredmény a munkaregiszterben tárolódik.

8.5. ábra. *Vezérlőjelek táblázata az adatmozgási irányok meghatározásával*

A leírtak alapján felvázolhatjuk az elképzelt processzor belső felépítését. A 8.6. ábra egy olyan tömbvázlatot mutat, amely rögzíti az alkotóelemek helyét és a közöttük fennálló adat- és jelkapcsolatokat. A processzor besorolását tekintve egy Neumann-elvnek megfelelően felépített, 1 című gép. A kevés utasításszámot (8), azonos utasításszóhosszat (12bit) és azonos végrehajtási ciklusidőt (6 ütem) tekintve egy MISC-rendszer sajátosságait mutatja.

A processzorban a belső adatsínre kapcsolódik az ALU egyik bemenete, az adat-, cím-, utasítás- és munkaregiszter és a programszámláló. A programvezérlő utasítások (**JMP**, **JZ** stb.) végrehajtásának feltétele, hogy a programszámláló az 1-gyel való növelési funkció (inkrementálás) mellett be is tudjon tölteni egy értéket.

A munkaregiszter bemenetének kiválasztása utasításfüggő, kapcsolódhat az ALU adatsínen közvetlenül az ALU kimenetére, de fogadhat a belső adatsínen, az adatregisztertől érkező adatot is. A munkaregiszter kimenetének iránya szintén utasításfüggő, vagy az ALU egyik bemenetét képezi, vagy a belső adatsínen az adatregiszterhez kapcsolódik.

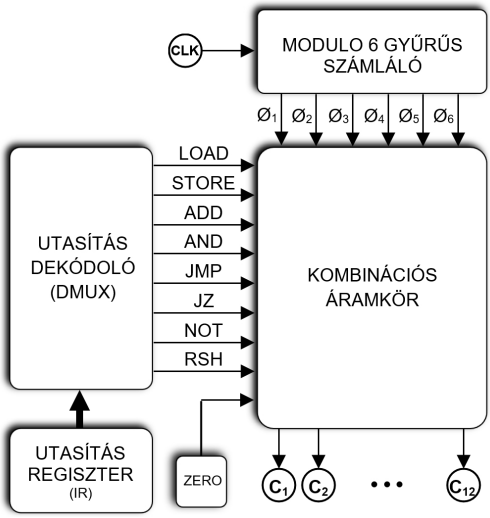


8.6. ábra. *A processzor belső felépítésének vázlatos rajza*

- instruction cycle explained
- control unit of a microprocessor

8.2.2. Huzalozott vezérlőegység tervezése

A 8.7. ábra egy huzalozott vezérlőegység főbb részeit és a közöttük lévő jeleket és azok irányát mutatja. A vezérlőegység három fő részből áll: egy gyűrűs számlálóból, egy demultiplexerből és a logikai kapuk hálózatából létrehozott kombinációs áramkörből. A kombinációs áramkör bemenetei a gyűrűs számláló, az utasításdekódoló és feltételképző (ZERO) kimeneteihez kapcsolódnak. A kombinációs áramkörben a számláló ütemjeleiből, az utasítások kódjából és a feltételképző kimeneti értékéből képezett logikai függvények adják a processzor működéséhez nélkülözhetetlen vezérlőjeleket. Az utasítás dekódoló bemenetéhez az utasításregiszter kapcsolódik.



8.7. ábra. *Huzalozott vezérlőegység tömbvázlata*

A tömbvázlatból kiindulva létrehozzuk a huzalozott vezérlőegység kapcsolási rajzát (8.9. ábra). A tervezéshez alkalmazzuk a szintekre bontás és elvonatkoztatás (absztrakció) módszerét. Elemtáris logikai kapukból bonyolultabb kombinációs és szakaszos áramköröket építünk, ezekből pedig funkcionális egységeket hozunk létre: a ciklusszámlálót (**Johnson CNT MOD 6**), a demultiplexert (DMX) és a logikai kapuk hálózatát.

A vezérlőegység 12 kimenetét (C_1-től C_{12}-ig) a logikai kapuk kombinációs hálózatából nyerjük. 6 logikai bemenete közül 3-at, az órajelet (**CLK**), az alapállapot (**RST**), valamint a feltétel jelzést (ZERO) külső forrásból származtatjuk. Belső forrásnak számít a szintén 3 bites bemeneti regiszter (IR). Ez utóbbi a végrehajtandó utasítás bináris értékét tároló regiszter.

Ahhoz, hogy a vezérlőegység az utasítás-végrehajtási folyamat felvázolt sorrendiséget betartsa, a számláló áramkör 6 ütemet számol ki újra és újra. Egy módosított Johnson-számláló éppen megfelelne a célnak (6.67. ábra), ha úgy állítjuk össze, hogy 6 ütem után, a 7. ütemben, visszatérjen az alapállapotába. A Johnson-számláló kimenete helyértékkódolt (nem binárisan kódolt számérték), minden ütemben eggyel nagyobb helyértékű kimenet aktiválódik, ezért a feladatra dekódolás nélkül alkalmazható.

A szakaszos működést ütemező órajel (CLK) közvetlenül a Johnson-számláló órajel bemenetére kapcsolódik. Az órajel tulajdonságairól előzetesen a 6.5.4 alfejezetben tárgyaltunk bővebben. A jel frekvenciája (üteme) határozza meg az utasítás-végrehajtás sebességét. Egy utasítás-végrehajtási ciklushoz 6 ütemre van szükségünk. Ha a jel frekvenciája f=1MHz, akkor egy utasítást a $t=(1/f)*6$ összefüggéssel fejezhetjük ki, ami $6*10^{-7}$, vagyis 0,000006 szekundum, azaz 6 mikroszekundum. Minél nagyobb az órajel frekvenciája, annál rövidebb ideig tart egy utasítás végrehajtása.

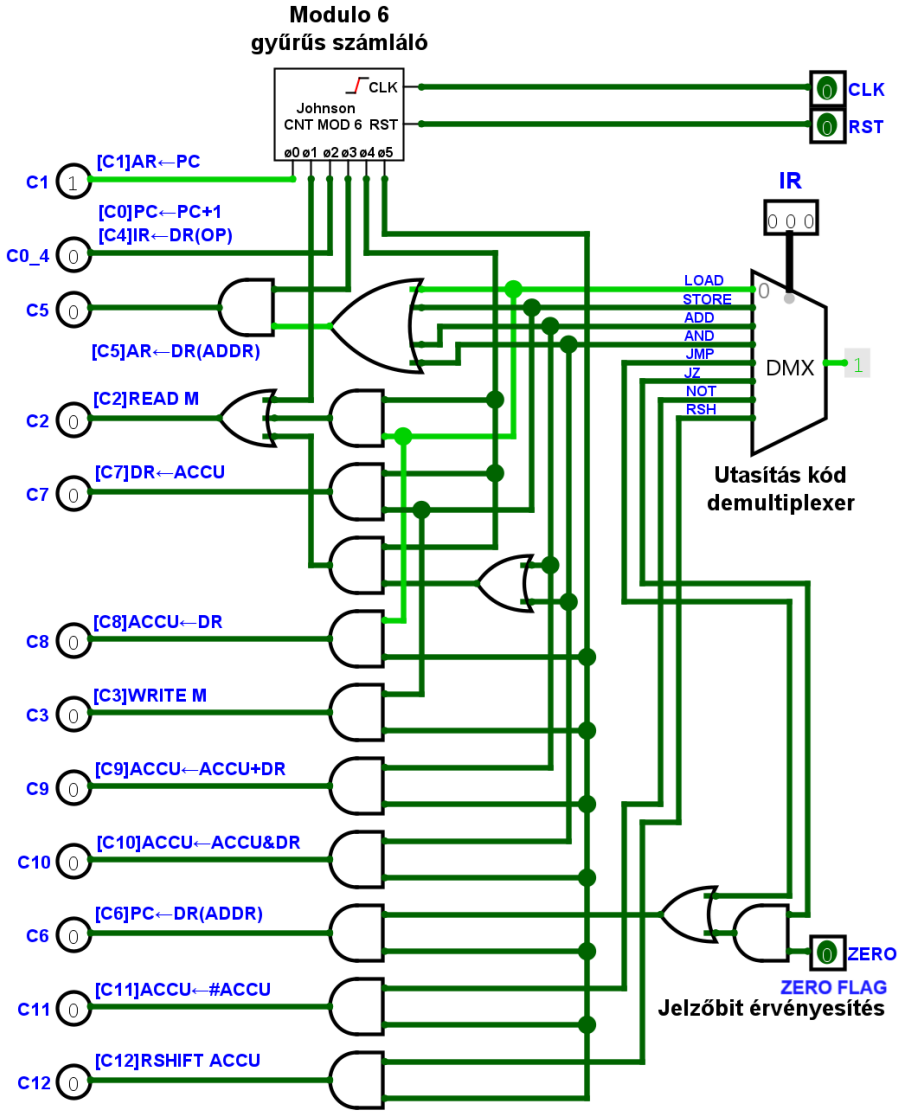
A demultiplexer bemenete a már említett, 3 bites utasításregiszter (IR). A tárolt bináris érték függvényében a multiplexer egyik és csak egyik kimenete aktiválódik. Így a különböző bináris számokkal kódolt utasítások könnyen szétválaszthatók.

A logikai kapuk hálózata egyszerű felépítést mutat, ÉS, valamint VAGY kapuk alkotják. Ezek szerepe a számláló és az utasításokat dekódoló demultiplexer kimenetei közötti logikai összefüggések (függvények) leképezése, amelyeket a következő táblázat foglal össze (8.8. ábra).

Vezérlőjelek logikai függvényei
$C_0=Ø_3$
$C_1=Ø_1$
$C_2=Ø_2+ Ø_5\cdot(LOAD)+ Ø_5\cdot(ADD+AND)$
$C_3=Ø_6\cdot(STORE)$
$C_4=Ø_3$
$C_5=Ø_4\cdot(LOAD+STORE+ADD+AND)$
$C_6=Ø_6\cdot(JMP+(JZ\cdot ZERO))$
$C_7= Ø_5\cdot(STORE)$
$C_8= Ø_6\cdot(LOAD)$
$C_9= Ø_6\cdot(ADD)$
$C_{10}= Ø_6\cdot(AND)$
$C_{11}= Ø_6\cdot(NOT)$
$C_{12}= Ø_6\cdot(RSH)$

8.8. ábra. *A vezérlőjeleket leíró logikai függvények táblázata*

Megfigyelhető a kapcsolási rajzon (8.9. ábra), hogy a központi, függőleges oszlopban elhelyezkedő és kapuk bemenetei egyrészről a számláló valamelyik kimenetét fogadják, másrészről a demultiplexer egyik kimenetére kapcsolódnak. Ebből következően, az adott utasítás végrehajtásához szükséges vezérlőjelek a számláló diktálta sorrendben és ütemben megjelennek az és kapuk kimenetein.



8.9. ábra. *Huzalozott vezérlőegység kapcsolási rajza*

A VAGY kapuk szerepe az egyes utasításokat jelző logikai kondíciók egyesítésében mutatkozik, amilyen például az adatmozgató (LOAD, STORE), valamint aritmetikai-logikai műveletek (ADD, AND) esetében a demultiplexer vonatkozó kimeneteit összefogó, négybemenetű VAGY kapu. A végrehajtási folyamatábra (8.4. ábra) alapján a felsorolt utasítások mindenikének elvégzéséhez szükséges adatbeolvasás, vagyis az utasításszóban megadott címérték beírása a címregiszterbe (AR). Ez a C_5-ös vezérlőjel segítségével történik minden vonatkozó utasítás esetében, amelyek a VAGY kapuban futnak össze.

Hasonló a helyzet a memóriaolvasási műveletet vezérlő vonal esetében, itt a hárombemenetű VAGY kapu azoknak az utasításoknak az esetében fog össze vonatkozó logikai kimeneteket, amelyek elvégzése során egy vagy két alkalommal is történik adatbeolvasás a memóriából (LOAD, ADD, AND).

Vegyük példának az adatbetöltő utasítást, a LOAD-ot, és elemezzük a vezérlőegység működését. A folyamatábra alapján tudjuk, hogy sorban a következő vezérlőjeleket kell létrehozni a végrehajtásához:

$$Ø_1 \rightarrow C_1,\ Ø_2 \rightarrow C_2,\ Ø_3 \rightarrow C_{04},\ Ø_4 \rightarrow C_5,\ Ø_5 \rightarrow C_2,\ Ø_6 \rightarrow C_8$$

Az órajel (CLK) első ütemére ($Ø_1$) a C_1-es vezérlőjel aktiválódik, ennek hatására a programszámlálóban (PC) létrejövő memória-címérték eltárolódik a processzor címregiszterébe (AR). Ennek a vezérlőjelnek még nincsen más logikai kondíciója, ezért logikai kapuk közbeiktatása nélkül halad tovább a címregiszter felé. A címregiszter kimenete rákapcsolódik a címsínre, és ezen megjelenik a tárolt címérték.

Az órajel második ütemére ($Ø_2$) a C_2-es vezérlőjel aktiválódik, ez az olvasás (RD – read) művelet beállítása a memóriának. A vezérlőjel több ütemben is létrejöhet ($Ø_2$, $Ø_5$), több utasítás esetében is (LOAD, ADD, AND), ezért az ezekben az esetekben is létrejövő logikai jeleket egy VAGY kapuban kell összefogni. A memória kimenetén megjelenik a címsínen beállított címen eltárolt adatérték és eltárolódik (latch) az adatregiszterbe (DR).

Az órajel harmadik ütemére ($Ø_3$) egy olyan vezérlőjel jön létre, a C_{04} (vagy csak **C4**), amely egyidejűleg két eseményt is elindít. Ennek sincs logikai kondíciója, ezért logikai kapuk közbeiktatása nélkül halad tovább a programszámláló (PC) és az utasításregiszter (IR) felé. A vezérlőjel a programszámlálóban lévő értéket 1-gyel növeli, ugyanakkor az adatregiszterben lévő adatérték utasításkódot hordozó részét (a legfelső 4 helyértékű bit) beírja az utasításregiszterbe. Ekkor megtörténhet az utasítás kódjának dekódolása a demultiplexer (DMX) segítségével. Mivel a LOAD utasításkódja 0000, ezért a demultiplexer 0-ás kimenete logikai 1-be áll.

Az órajel negyedik ütemére ($Ø_4$) a C_5-ös vezérlőjel aktiválódik. Ez a jel ugyanebben az ütemben több utasítás esetében is létrejöhet (LOAD, STORE, ADD, AND), ezért a demultiplexer vonatkozó kimeneteit egy VAGY kapuban fogjuk össze, majd egy ÉS kapu segítségével kondicionáljuk a létrehozását az órajel negyedik ütemének függvényében. A vezérlőjel hatására a címregiszter eltárolja a memóriából előzetesen beolvasott LOAD utasítás címmezőjének értékét (az alsó 8 helyértékű bit). A címregiszter kimenete rákapcsolódik a címsínre, és ezen megjelenik a tárolt címérték.

Az órajel ötödik ütemére ($ø_5$) ismét a c_2-es vezérlőjel aktiválódik, ez az olvasás (RD) művelet beállítása a memóriának. A memória kimenetén megjelenik a címsínen beállított címen eltárolt adatérték és eltárolódik az adatregiszterbe.

Az órajel utolsó, hatodik ütemére ($ø_6$) a c_8-as vezérlőjel aktiválódik. létrejöttének egyetlen logikai kondíciója, hogy a LOAD utasítás legyen betöltve, ebben az esetben a processzor belső adatsínjén történik adatmozgatás, az adatregiszterben eltárolt érték átíródik (szintén eltárolódik) a munkaregiszterbe (ACCU).

Ezen a ponton a betöltő (LOAD) utasítás végrehajtása befejeződött, de mivel végrehajtása során a programszámláló értékét 1-gyel megnöveltük, a következő, új ütemsorozatban a vezérlőegység újabb utasítást fog beolvasni és végrehajtani.

Egy másik utasítás végrehajtásának elemzése csak az utolsó három ütemre korlátozódhat, mivel az első szakasz minden utasítás esetében azonos módon zajlik le (lásd a 8.2. alfejezet utolsó mondatát). Nézzük például a feltétel nélküli ugrást, a JMP utasítást. A folyamatábra alapján tudjuk, hogy sorban a következő vezérlőjeleket kell létrehozni a végrehajtásához:

$$Ø_1 \rightarrow C_1,\ Ø_2 \rightarrow C_2,\ Ø_3 \rightarrow C_{04},\ Ø_4,\ Ø_5,\ Ø_6 \rightarrow C_6$$

Az órajel negyedik és ötödik ütemére ($ø_4$, $ø_5$) nem jön létre vezérlőjel, üresjárat van.

Az órajel utolsó, hatodik ütemére ($ø_6$) a c_6-os vezérlőjel aktiválódik. Létrejöttének egyetlen logikai kondíciója, hogy a JMP utasítás (0100) legyen betöltve. A vezérlőjel hatására a programszámláló eltárolja a memóriából előzetesen beolvasott JMP utasítás cím mezőjének értékét (az alsó 8 helyértékű bit), ezzel **felülírva** a harmadik ütemben ($ø_3$) 1-gyel megnövelt programszámláló értéket.

Ezen a ponton a feltétel nélküli ugrás (JMP) utasítás végrehajtása befejeződött, de mivel végrehajtása során a programszámlálóban lévő értéket felülírtuk egy újabb értékkel, a következő utasítás ebből következően az így beállított címről kerül végrehajtásra.

Nagyon hasonló a kondicionált ugrás utasítás, a JZ végrehajtása is, azzal a különbséggel, hogy a programszámlálóban lévő érték felülírása a munkaregiszter (ACCU) nullás eltárolt értékétől függ. Ha a munkaregiszter értéke 0 volt, a ZERO jelzőbit logikai 1-be áll mindaddig, amíg a munkaregiszterben tárolt érték 0-tól különböző lesz. A ZERO jelzőbit logikai kondícióját egy ÉS kapuban fűzzük össze a demultiplexerből kilépő JZ utasítás dekódolása esetén megjelenő jellel. Így jön létre ismét a **c6**-os vezérlőjel, amelynek hatására a programszámláló eltárolja a memóriából előzetesen beolvasott JZ utasításcím mezőjének értékét (az alsó 8 helyértékű bit), ezzel **felülírva** a harmadik ütemben ($ø_3$) 1-gyel megnövelt programszámláló értéket.

Ha a ZERO jelzőbit nem aktív, a programszámláló értéke **nem íródik felül**, így az utasítás-végrehajtás a következő ütemsorozatban az 1-gyel növelt programszámláló érték által meghatározott memóriacímről folytatódik. Az elemzés alapján a többi utasítás végrehajtási menete is hasonló módon értelmezhető.

8.2.3. Mikrokódos vezérlőegység tervezése

Az előző alfejezetben láthattuk, hogy a huzalozott vezérlőegység nagy része logikai kapukból és ezek ki- és bemeneteit összekötő vezetékek hálózatából épül fel. Ezért az utasítás-végrehajtás ütemezését biztosító gyűrűs számláló jelzéseire a kimeneti, vezérlőjelek szinte azonnal megjelennek, késleltetést csak az egymás után kapcsolt logikai kapuk átfutási ideje jelent. Ebből következően az utasítás-végrehajtás nagyon gyorsan zajlik.

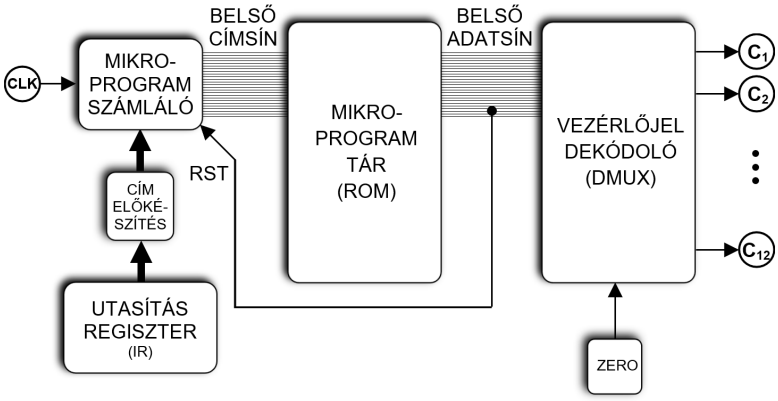
A logikai kapuk típusa, ki- és bemeneteik összekapcsolási módozata az egyes utasítások végrehajtási lépéseit leíró folyamat logikai függvénye. Ha újabb utasítással kívánjuk bővíteni a már meglévő készletet (vagy javítani kell egy logikai, esetleg kapcsolási hibát), az őket leíró logikai függvények megvalósítása újabb áramkörök kialakítását (vagy megváltoztatását) igényli, amelyeket a meglévő hálózatba illesztünk be. Ezt egy szimulációs környezetben (szoftver) könnyen megtehetjük, de ha a 4.4-es alfejezetben ismertetett, összetett gyártástechnológiai folyamatokra gondolunk, belátható, hogy a meglévő áramkörhöz akár csak egy MOSFET tranzisztor hozzáadása vagy egy belső vezeték módosítása is bonyolult tervezési, majd kivitelezési folyamatokat feltételez.

Bár az utasítás-végrehajtás folyamata lassabb lesz, mint a huzalozott vezérlőegység esetében, egyszerűbbé válik az utasításkészlet-bővítés (vagy hibajavítás), ha az egyes utasítások végrehajtási lépéseit leíró folyamat logikai függvényeiből következő vezérlőjeleket binárisan kódolt formában, egy memóriában tárolnánk. Innen, az utasítás-végrehajtás adott ütemében kiolvasva, majd dekódolva, továbbítanánk a megfelelő regiszter felé. Egy ilyen rendszert mikrokódos vezérlőegységnek nevezünk.

Igény esetén a memória tartalmának átírásával újabb utasítások végrehajtásához szükséges vezérlőjelek csoportja adható hozzá a már meglévő rendszerhez, a véletlenül hibásan létrehozott szekvenciák is javíthatók, vagy újabb feltételes végrehajtási szakaszok is képezhetők.

A következő tömbvázlat (8.10. ábra) egy ilyen rendszert mutat be, ahol a kombinációs áramkör szerepét egy memóriaegység veszi át. Minden végrehajtási ciklusban két rövid lépéssorozat fut le, mint egy rövid program. Ennek során rendre létrejönnek az utasítás-végrehajtáshoz szükséges vezérlőjelek. A mikrokódos vezérlőegység főbb egységei a mikroprogram-számláló, a mikroprogramtár és a vezérlőjel-dekódoló. A mikroprogramtár feltétlen olyan memóriatípus, amely tápfeszültség hiányában is megtartja a beírt adatokat (ROM, EEPROM). A mikroprogramszámláló és a mikroprogramtár között a belső címsín, míg a vezérlőjel-dekódoló és a mikroprogramtár között a belső adatsín biztosítja az összeköttetést. A mikroprogram-számláló mikroprogram-vezérelt. A beolvasási szekvencia végén egy „ugrást” hajt végre a mikroprogramtár címterületén belül, a beolvasott utasításkód alapján. A végrehajtási szekvencia végén pedig alapállapotba áll vissza (**RST**).

Az ismert végrehajtási folyamat az utasítást beolvasó mikroprogram-szekvenciával kezdődik, majd ha ennek kódja már az utasításregiszterben (IR) van, lefut egy újabb, rövid lépéssorozat, és egy dekódoló segítségével létrehozza a szükséges vezérlőjeleket.



8.10. ábra. *Mikrokódos vezérlőegység tömbvázlata*

A mikrokódos vezérlőegység tervezését a 8.1. és 8.2.1-es alfejezetekben ismertetett módszerekkel kezdjük, ugyanazt a 8 utasítást és végrehajtási folyamatábrát használjuk fel a mikroprogram kialakításához. Idézzük fel, hogy az utasítás-végrehajtás menetét két szakaszra osztottuk: utasítás-előkészítési (fetch) és utasítás-végrehajtási szakaszra (execute). Ebből következően a mikroprogram egy általános utasítás-előkészítési szekvenciával fog kezdődni, amit az egyes utasítások végrehajtási szekvenciái követnek majd. Azt a szabályt követjük, hogy minden utasítás minden végrehajtási ütemének külön mikroutasítás-szót tartunk fent, és a mikroutasítás-szónak tartalmaznia kell az adott ütem vezérlőjelkódját.

A folyamatábrából tudjuk, hogy a közös, 3 ütemnyi előkészítő szakasz után újabb 3 ütem szükséges egy utasítás végrehajtásához. Az utasítások kódja 4 bit (a további bővítést támogatandó). A 3 ütemnyi jelzés kódolásához 2 bitre van szükségünk. A mikroutasítások címét ezért úgy alakítjuk ki, hogy az alsó 2 bit az ütemek (phase – PH) kódjából, a felső 4 bit pedig a végrehajtandó utasítás kódjából (opcode – OP, 8.11. ábra) következik.

Utasítás kód				Ütem kód	
OP_3	OP_2	OP_1	OP_0	PH_1	PH_0

8.11. ábra. *A 6 bites mikroutasítás-címérték szerkezete*

Ezzel a szerkezettel az egyes utasításhoz tartozó fázisok mikroutasításainak címértékei csak a fázisszámban különböznek egymástól (0, 1 és 2), az utasítás kódjából következő címrész állandó marad (pl. 0001 – LOAD). Ezért ezeket az állandó címrészeket használjuk ugrási címnek az utasítás-végrehajtás második szakaszának megkezdésekor. Mivel a cím felső 4 bitjének értéke (OP_3 … OP_0) az alsó 2 bit (PH_1 … PH_2) minden kombinációjában állandó, egy negyedik, esetünkben szükségtelen fázisszámhoz (a 3-ashoz) is tartozik egy mikrokód-lokáció, aminek értéke azonban 0 (nem használjuk).

6 biten 64 memóriarekeszt címezhetünk meg, ez 16 lehetséges utasítás tárolásához lenne elegendő (16*(3+[1])=64). Mivel a végrehajtás-előkészítő fázis (fetch) ütemeit egy alkalommal, a címterület első négy helyére illesztjük be, a memóriában csak 15 utasításnak marad hely. Ez magyarázza azt is, hogy miért kell, 1-gyel való növeléssel, átkódolni az utasításokat a mikrokódos végrehajtóegység számára (a 8.1. ábra szerint a LOAD utasítás kódja 0000).

A mikroutasítások kódolására többféle módszer is létezik. A legegyszerűbb esetben a mikroutasítás tartalma közvetlenül vezérlőjelekre fordítható, ilyenkor a mikroutasításban nincs hely más jelzéseknek. Olyan esetben, ahol a lehetséges vezérlőjelek száma meghaladja a rendelkezésre álló bithelyek számát, bináris kódokat használnak a vezérlőjelek, -jelcsoportok kódolására. A kódokat a végrehajtás során egy dekódoló áramkör alakítja át vezérlőjelekké.

Esetünkben az utóbbi módszer alkalmazása célszerű, mivel a mikroprogram-memóriát kitöltő mikroutasítások hossza 8 bit, felépítésüket tekintve három mezőből állnak (8.12. ábra), AS_0 (Address Selector) – címkiválasztó, $S_{2\text{-}0}$ (Condition Selector) – feltétel és $D_{3\text{-}0}$ (Control Signals – CTRL) – vezérlőjelmezőből. A 12 vezérlőjel (C_1-től C_{12}-ig) megkülönböztetésére csak 4 bithely áll rendelkezésre. A végrehajtáskor egy demultiplexer bemenetére vezetjük a CTRL mező tartalmát, így kapjuk meg a vonatkozó vezérlőjelet a kimenetén.

AS	Feltétel			Vezérlő jelek			
AS_0	S_2	S_1	S_0	D_3	D_2	D_1	D_0

8.12. ábra. *A 8 bites mikroprogram utasításszó-szerkezete*

Az AS mező a mikroprogram utasításszó legfelső helyértékű bitjeként a mikroprogram-számláló beolvasás üzemmódját vezérlő jel. Ez a bit az utasítás-előkészítési folyamat utolsó ütemében aktív, a mikroprogram-számláló ilyenkor tölti be a végrehajtandó utasítás bináris kódját (amelyet 1-gyel növelni fog a helyes ugrási cím kiszámításának érdekében). A feltétel mező S_0 bitje egyelőre nem használt, az S_1 bit a feltételes ugrás 0 (ZERO) kondíciójának vizsgálatát jelzi, míg az S_2 az adott utasításhoz tartozó mikrokód-szekvencia végén lévő címbetöltést (ugrás a mikroprogramtár kezdőcímére) jelzi a következő táblázatban (8.13. ábra) összefoglalt módon.

AS_0	$S_2S_1S_0$	Magyarázat
0	**000**	Vezérlő jel előállítás
0	**100**	Ugrás a mikroprogram tár kezdőcímére
0	**110**	ZERO jelzőbit által kondicionált vezérlőjel előállítás (C6)
1	**000**	Ugrás a műveleti kód értékkel indexelt mikroprogram tár címre

8.13. ábra. *Mikroprogram-kiválasztó jelzéseinek összefoglaló táblázata*

A 8 alaputasításhoz tartozó mikroprogramtár (ROM típusú memória) tartalmát a következő táblázat (8.14. ábra) foglalja össze. Kitöltése a leírtak szerint történhet: a CTRL mezőkbe beírjuk az adott fázisban szükséges vezérlőjel kódját, ahol feltétel vizsgálata vagy ugrás következik (minden harmadik ütemben), kitöltjük az $s_2s_1s_0$ mezőket. Az AS mező csak az utasítás-előkészítési folyamat utolsó ütemében aktív.

A mikroprogram utasításainak meghatározása után áttérhetünk a mikrokódos vezérlőegység kialakítására (8.15. ábra). A huzalozott vezérlőegységgel ellentétben itt a gyűrűs körszámláló helyett 8 bites, párhuzamos betöltésű, bináris számlálót használunk. Mivel a mikroutasítás-sorozatok végrehajtási ciklusainak vége programvezérelt, mindig pontosan hat ütem után a mikroprogram-számláló alapállapotba áll vissza (reset), és ezzel elölről kezdődik a folyamat.

A mikroprogram végrehajtási folyamatának elején a mikroprogram-számláló tehát 0-ból indulva bejárja az utasítás-előkészítés (fetch) fázisait. A memóriából kiolvasott utasítás kódja az utasítás regiszterből, a belső adatsín 5-2 helyértékeit elfoglalva, először egy inkrementáló áramkörbe kerül. Ez, a már leírtak szerint, 1-gyel növeli az értékét (fetch offset). Erre azért van szükségünk, mert az utasítás-előkészítés fázis mikroutasításai a mikroprogramtár első, a 0x00h címtől kezdődő 4 lokációját foglalják el. Ennek következtében a rendelkezésünkre álló 4 bithelyen az általános FETCH szakasz mellett csak 15 utasításhoz tartozó mikrokódmezőt tudunk létrehozni.

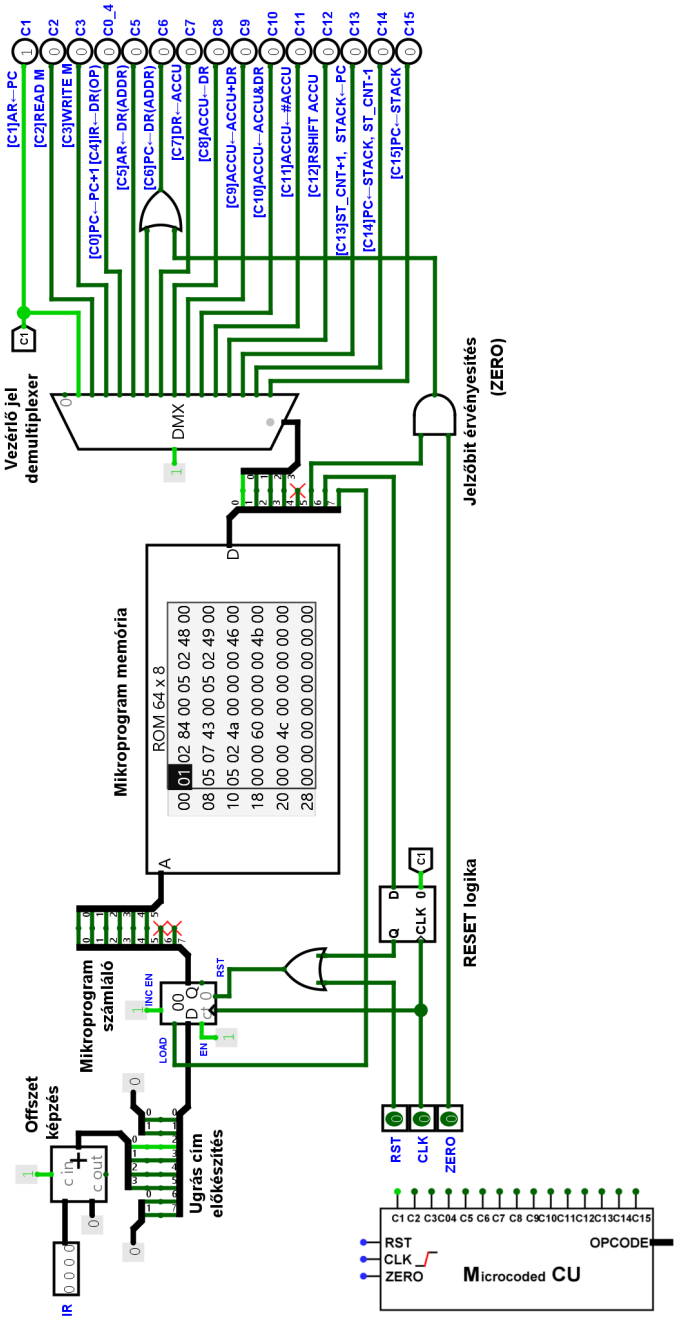
Az utasítás végrehajtási szakaszába (execute) lépve a mikroutasítás értelmében megtörténik a mikroprogram-számláló feltöltése az 1-gyel megnövelt utasításkóddal. Az így kialakuló címről folytatódik a következő három fázisban a mikroutasítások végrehajtása.

A mikroutasítás-végrehajtás egy memóriaolvasásból és a kimeneti adat dekódolásából, illetve a feltétel mező ($s_2s_1s_0$) bitjeinek logikai vizsgálatából áll. A demultiplexer a mikroutasításba kódolt vezérlőjelet állítja elő, míg egy további ÉS kapu a feltételes ugráshoz szükséges ZERO jelzőbit állapotától teszi függővé a vonatkozó vezérlőjelet (JZ utasítás, c_6-os vezérlőjel).

A folyamat végén az utolsó mikroutasítás feltétel mezőjében ($s_2s_1s_0$) beállított 100 érték nullázza a mikroprogram-számlálót, 1-es értéket írva a RESET logika

Tár cím	Memória tartalom			Utasítás	Mikro-művelet	Vezérlő jel
	AS	$S_2S_1S_0$	CTRL			
000000	0	000	0001	FETCH	AR←PC	C1
000001	0	000	0010		READ M	C2
000010	1	000	0100		PC←PC+1 IR←DR(OP)	C04
000011	0	000	0000		-	-
000100	0	000	0101	LOAD	AR←DR(ADDR)	C5
000101	0	000	0010		READ M	C2
000110	0	100	1000		ACCU←DR	C8
000111	0	000	0000		-	-
001000	0	000	0101	STORE	AR←DR(ADDR)	C5
001001	0	000	0111		DR←ACCU	C7
001010	0	100	0011		WRITE M	C3
001011	0	000	0000		-	-
001100	0	000	0101	ADD	AR←DR(ADDR)	C5
001101	0	000	0010		READ M	C2
001110	0	100	1001		ACCU←ACCU+DR	C9
001111	0	000	0000		-	-
010000	0	000	0101	AND	AR←DR(ADDR)	C5
010001	0	000	0010		READ M	C2
010010	0	100	1010		ACCU←ACCU&DR	C10
010011	0	000	0000		-	-
010100	0	000	0000	JMP	-	-
010101	0	000	0000		-	-
010110	0	100	0110		PC←DR(ADDR)	C6
010111	0	000	0000		-	-
011000	0	000	0000	JZ	-	-
011001	0	000	0000		-	-
011010	0	110	0110		PC←DR(ADDR)	C6
011011	0	000	0000		-	-
011100	0	000	0000	NOT	-	-
011101	0	000	0000		-	-
011110	0	100	1011		ACCU←/ACCU	C11
011111	0	000	0000		-	-
100000	0	000	0000	RSH	-	-
100001	0	000	0000		-	-
100010	0	100	1100		RSHIFT ACCU	C12
100011	0	000	0000		-	-

8.14. ábra. *A mikroprogramtár szerkezete és tartalma*



8.15. ábra. *Mikrokódos vezérlőegység kapcsolási rajza és szimbóluma*

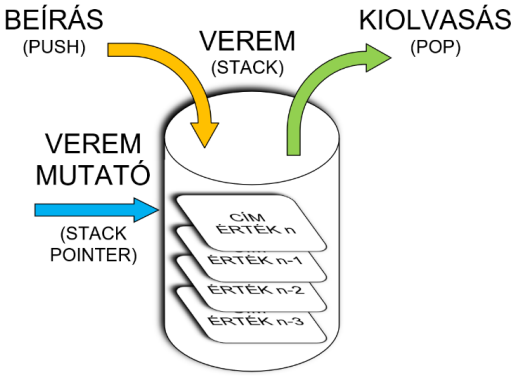
D tárolójába, az órajel (CLK) felfutó élére szinkronizálva. A RST (Reset) jelzés ideiglenes, az órajel lefutó éle utáni megtartását szolgáló D tároló tartalma és kimenete a c_1 vezérlőjel megjelenése pillanatában, aszinkron módon 0 lesz, megszüntetve a mikroprogram-számláló RESET kondícióját. Ezzel elkezdődik egy újabb utasítás előkészítése, majd végrehajtása.

Megjegyzendő, hogy az utólagos utasításkészlet-bővítéshez és a végrehajtásukhoz szükséges, esetleges új vezérlőjelek beillesztéséhez a mikrokódos vezérlőegységet és a teljes processzorarchitektúrát úgy kell megtervezni, hogy kellő számú, általánosan kapcsolható jelvezetéket is tartalmazzon. Másik módszer, hogy a bővítéshez csak a meglévő jelzéseket használják új csoportosításban.

- hardwired control unit of a MCU
- micro-programmed control unit

8.3. A veremvezérlő

A processzor vezérlőegységének megtervezése után egy újabb fontos elem, a veremvezérlő tervezésével folytatjuk. A verem a számítástechnikai fogalomtárban egy olyan memóriaegységet jelöl, amely adatok vagy címértékek ideiglenes eltárolására szolgál. Az éppen elérhető lokáció címét a veremmutató tartja (8.16. ábra).



8.16. ábra. *Verem (stack) működési elve és főbb részei*

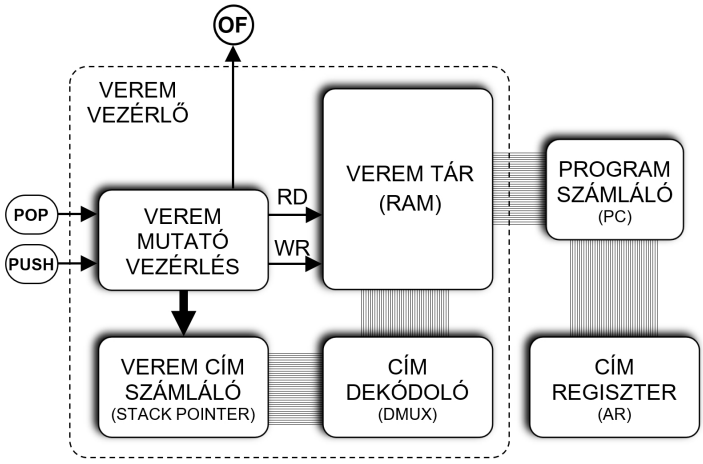
A verem egyik fontos alkalmazási esete a visszatérési címértékek tárolása. Programozástechnikában visszatérési címnek nevezzük azt a memória-címértéket, amelynél valamilyen előre programozott vagy éppen előre nem látott esemény miatt a fő programszál végrehajtása megszakad, és egy másik, nem sorban következő

címértékről beolvasott utasítással, mellékszálon (eljárás – procedure) folytatódik, majd a mellékszálból vissza kívánunk térni a fő végrehajtási szálra. Az ilyen esetek többségében a visszatérési címérték elmentése nélkül a processzor nem tudna visszatérni a fő programszál utasításainak végrehajtásához.

A verem-programvezérlő utasítások végrehajtásakor (eljáráshívás – procedure call) vagy külső esemény hatására (megszakítás – interrupt) először a programszámláló pillanatnyi értékét (mindig a már megnövelt, beállított értéket) a veremmutató által megjelölt „üres” memóriarekeszbe (regiszterbe) másolja (**PUSH**) és növeli a veremmutató értékét. Majd egy újabb programvezérlő utasítás végrehajtásakor (eljárásból való visszatéréskor) innen visszamásolja a programszámlálóba (**POP**). Egymásba ágyazott eljáráshívások esetében minden programszámláló értékmentés után a veremmutató (stack pointer) értéke megnő, és az utolsó, foglalt memóriarekesz címét tartja. Ezzel biztosítja a visszatérési címek helyes sorrendiségét a programszámlálóba való visszamásolás során. Az eljárásból való visszatérések alkalmával a veremmutató értéke rendre, minden címérték programszámlálóba való visszamásolása után csökken, ahogy a memóriarekeszek sorban „felszabadulnak”.

8.3.1. Veremvezérlő tervezése

A veremvezérlő megvalósítását tekintve lehet hardveres vagy szoftveres változat, lehet kötött vagy dinamikusan beállítható memóriaméretű, a veremmutató működhet alapból növekvő vagy alapból csökkenő módon. Működése több megvalósításban is automatikus, de léteznek komplex változatai, amelyeknél a programozónak hozzáférése van a veremmutató értékéhez (megváltoztathatja), illetve dinamikusan választhatja meg a veremnek fenntartott tárhely méretét is.



8.17. ábra. *Verem (stack) vezérlő tömbvázlata*

A veremvezérlő négy fő egységre bontható (8.17. ábra). A veremtár egy írható-olvasható memória (RAM), hozzá kapcsolódik egy címdekódoló áramkör. A címdekódoló bemenetét a veremszámláló (STACK POINTER) képezi. A veremmutatót vezérlő áramkör a beíró és kiolvasó jelzések függvényében beállítja a veremtár ki- és bemeneteit, valamint növeli vagy csökkenti a veremmutató értékét. Ha a veremmutató értéke meghaladja a veremtár maximális címtartományának értékét, egy belső hibajelzést hoz létre, amelyet veremtúlcsordulásnak nevezünk (stack overflow – OF).

A veremvezérlő összeköttetésben van a processzor programszámlálójával, több megvalósításban ez a két egység nem is választható el egymástól.

A processzorba beépített veremvezérlő kapcsolási rajzát a tömbvázlat alapján állítjuk össze. A veremtár ebben az esetben 4 címtároló regiszterből áll. A címtároló regiszterek kimeneteit összefogó sín a processzor programszámlálójának a bemeneteire, míg a bemeneteiket összefogó sín a programszámláló kimenetére kapcsolódik.

Míg a címtároló regiszterek bemenetei egy közös sínre kapcsolódnak, addig a kimeneteik egy-egy sínkapcsolat-engedélyező áramkörön haladnak keresztül. Egy időpillanatban csak egy címtároló regiszter kapcsolódhat a kimeneti sínen keresztül a programszámláló bemenetére.

A címtároló regisztereket egy demultiplexer (DMX) segítségével választjuk ki (**en**). A demultiplexer adat bemenetét mindig logikai 1-ben tartjuk, kiválasztó bemenetei a veremszámlálóra kapcsolódnak. Ennek pillanatnyi értéke határozza meg az éppen aktív címtároló regisztert.

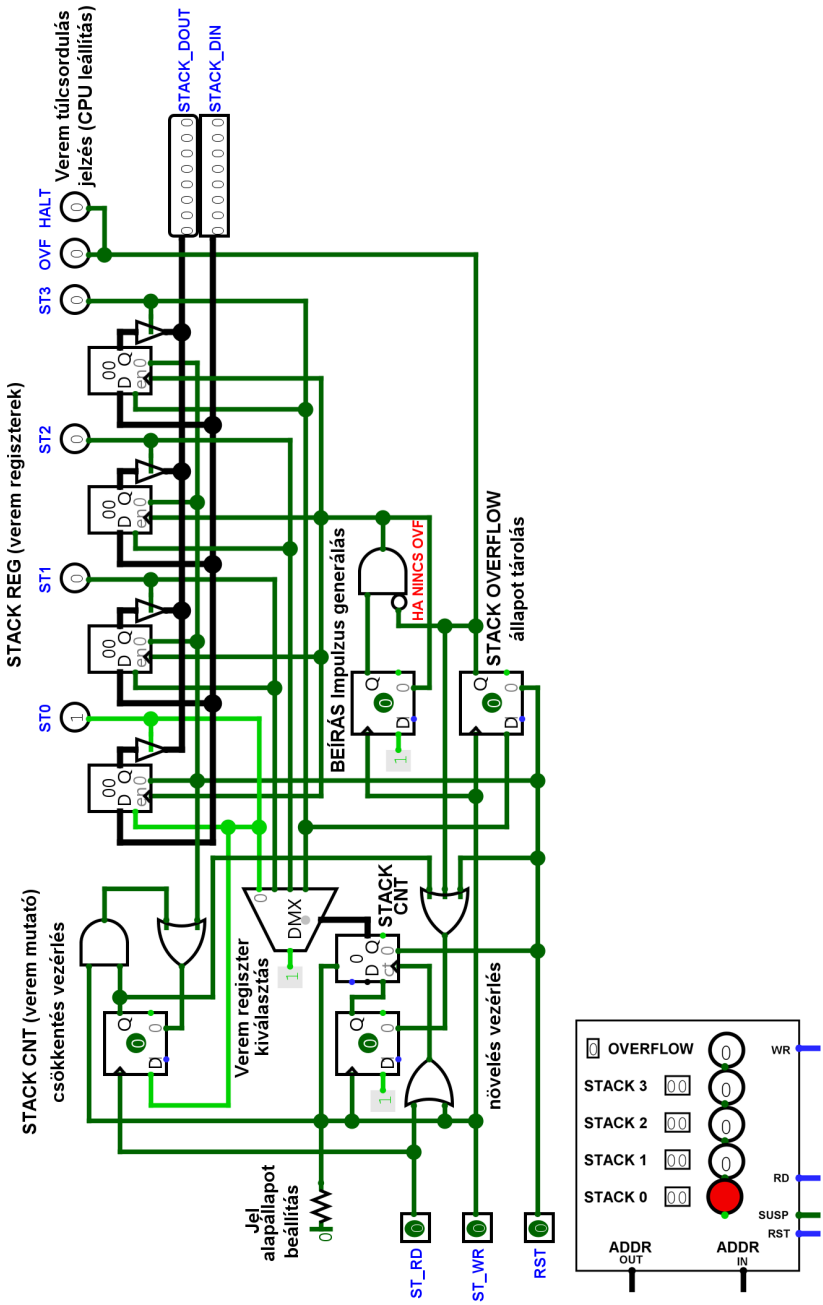
A veremszámláló növelni és csökkenteni is tudja az értékét annak függvényében, hogy beírás (PUSH) vagy kiolvasás művelet (POP) zajlik. Alapállapotban a veremszámláló értéke 0, ilyenkor a 0-ás címtároló regiszter van kiválasztva.

Amikor beírás történik a ST_WR vonal aktiválásával ÉS nincs veremtúlcsordulás, a beírásimpulzus-generálás részegységet alkotó D tároló egy nagyon rövid beírásengedélyező jelzést (clock) hoz létre a kiválasztott (en) címtároló regiszternek. Ekkor a bemeneti sínen lévő címérték a kiválasztott címtároló regiszterbe íródik. Ugyanekkor a veremszámláló értéke is 1-gyel megnő, mivel a veremszámláló növelő vagy csökkentő üzemmódját kiválasztó bemenet 1-ben áll, ebből következően a veremszámláló növelni fogja az értékét.

A veremszámláló növelésvezérlés részegysége egy D tárolóban fenntartja az első beírás jelzés 1-es értékéből következő veremszámláló engedélyezést (Enable) egészen addig, amíg veremtúlcsordulás (stack overflow) nem következik.

Amikor kiolvasás történik a ST_RD vonal aktiválásával, a veremszámláló növelő vagy csökkentő üzemmódját kiválasztó bemenet 0-ba áll, ebből következően a veremszámláló csökkenteni fogja az értékét.

A veremszámláló csökkentésvezérlés részegysége csak akkor engedélyezi a veremszámláló csökkentését, ha ennek értéke nem nulla. Amikor az értéke nulla lesz egy kiolvasást követően, egy D tároló segítségével rövid impulzust képez a



8.18. ábra. *Veremvezérlő-egység kapcsolási rajza és szimbóluma*

veremszámláló működésengedélyezést fenntartó D tároló aszinkron reset bemenetére, így a veremszámláló engedélyezést (Enable) 0-ra állítja, gátolva ezzel a számlálóérték csökkentéséből fakadó esetleges negatív irányú túlcsordulást.

Ha a verembe a maximális számú, egymás után következő beírás után egy újabb beírásjelzés következne, a stack overflow állapottárolás részegység egy D tárolóba beírt 1-es értékkel létrehozza az OVF és HALT jelzéseket. Ennek hatása állandó, nem lehet kiolvasással megszüntetni, a processzor működése ilyen esetben leáll. A számítástechnikai rendszer működése csak az alapállapotba hozással (RESET) indítható újra.

- stack pointer
- stack based CPU

8.4. Belső megszakításvezérlő

A veremvezérlő hozzáadása a processzor architektúrájához előfeltétele a megszakításkérést (IRQ – Interrup ReQuest) kiszolgáló funkció megvalósításának. A megszakításkérés kiszolgálása egy olyan folyamat, amelynek során a számítástechnikai rendszer egy kívülről érkező jelzésre a lehető legrövidebb időn belül reagálni tud egy külön programrész utasításainak végrehajtásával. Ennek lebonyolításához a processzorban szükség van egy belsőmegszakítás-vezérlő áramkörre.

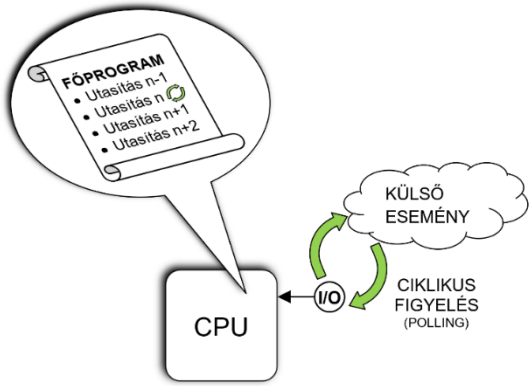
8.4.1. Külső események követése

Egy számítástechnikai rendszer több ki- és bemeneti csatornán keresztül kapcsolódhat a külvilághoz. A bemeneti csatornák közvetítik a számítástechnikai rendszeren kívülről érkező jelzéseket, amelyekre a rendszer valamilyen válaszreakciót ad. A processzor által végrehajtott utasítások sorozata válogatja és elemzi ezeket a jelzéseket, majd egy előre meghatározott módon (egy beprogramozott algoritmus alapján) adatokat módosít vagy jelzéseket hoz létre, amelyeket a rendszeren kívülre is eljuttat.

Ezeknek a külső eseményeknek a megfigyelése legegyszerűbben ciklikus ellenőrzéssel valósítható meg (8.19. ábra). Ez a programfuttatás (utasítások sorozatának végrehajtása) során egy vagy több bemeneti csatorna jelzéseinek ciklikusan ismétlődő megfigyelését és kiértékelését jelenti. A kiértékelés alapján a program a számítástechnikai rendszer kimenetein jelzéseket hozhat létre.

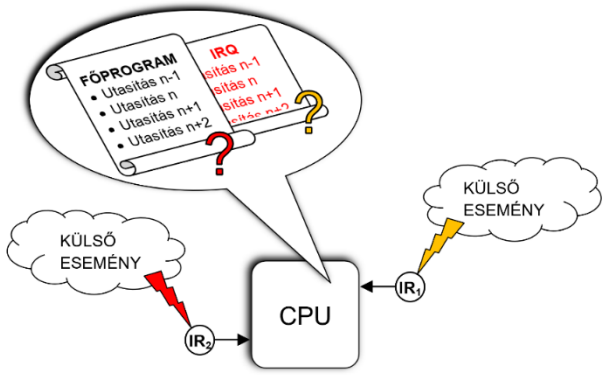
A ciklikus ellenőrzés a futtatott program bizonyos részeiben aktív, feltételezi, hogy a teljes program újra és újra végrehajtódik, vagy legalább azok a részei, amelyek a bemenetek mintavételezését és kiértékelését végzik. A bemenetek

mintavételezésével a mintavétel pillanatában érvényes jelállapotok rögzülnek, semmit sem tudunk meg a jelek állapotváltásainak eseményéről. Csak arra következtethetünk, hogy a változás a két ellenőrzési pillanat között történhetett.



8.19. ábra. *Külső esemény figyelése ciklikus ellenőrzéssel (I/O művelettel)*

A ciklikus megfigyelés során azok a jelzések, amelyek a bemeneti csatorna mintavételezése pillanatában nem aktívak, a kiértékelő algoritmus számára „láthatatlanok” maradnak. A gyorsan változó jelzések megfigyelése érdekében gyorsíthatjuk az ellenőrző programrész végrehajtását, gyakoribbá tehetjük a mintavételi pillanatokat, ezzel jelentősen lefoglalva a processzort. Ebből következően minden más programrész futtatása egyre jobban alárendelődik a mintavételi és kiértékelő algoritmus végrehajtásának. Mivel a bemeneti jelzések állapotváltásainak eseményéről továbbra sincs pontos, kiszámítható információnk, a kiadható válaszreakció időzítése csak hozzávetőleges.



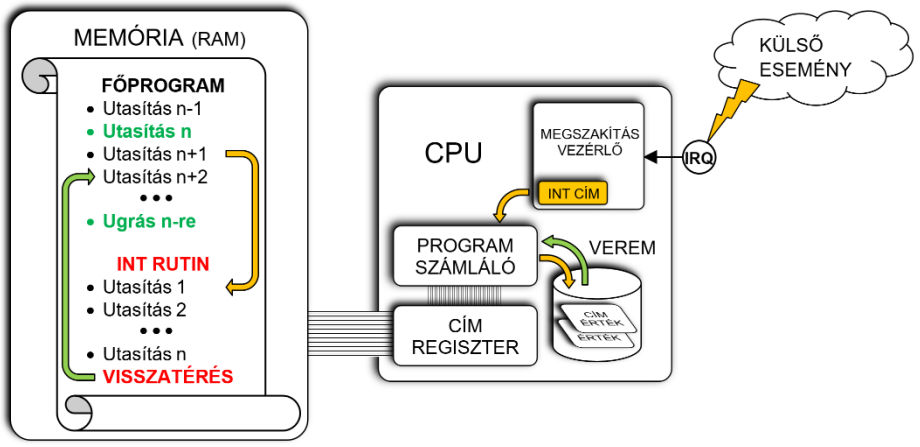
8.20. ábra. *Váratlan, külső események hatása a processzorra: a válaszadáshoz szükséges a főprogram futtatásának megszakítása*

Jobb eredmény érhető el a megszakításrendszer használatával (8.20. ábra). Ez nem igényel a program végrehajtása során következő ciklikus mintavételezéseket és kiértékeléseket, a külső jelzés állapotváltásának eseménye azonnal rögzül. Ilyenkor a főprogram futtatása ideiglenesen megszakad, és a külsőjelzés-kielemzésről és válaszreakció-generálásról egy speciális programrészlet, az úgynevezett megszakítást kiszolgáló eljárás (interrupt routine) gondoskodik.

Ez a módszer a bemeneti jel állapotváltásának időpillanatához sokkal közelebb eső és kiszámíthatóbb válaszreakciót tud létrehozni. Hatékonyabb lehet így a processzor kihasználtsága is.

8.4.2. Megszakításrendszer, megszakításkiszolgálás

A processzoron belül a megszakításrendszert a megszakításvezérlő működteti. Ennek feladata a megszakításjelzés (egy kiválasztott, speciális bemenet – IRQ – Interrupt Request) figyelése és állapotváltásának pillanatában (0-ból 1-be, vagy fordítva) a megszakítást kiszolgáló programrész végrehajtásának kezdeményezése. A folyamatot a 8.21. ábra szemlélteti.



8.21. ábra. *Megszakításkérés (IRQ) kiszolgálásának menete – a narancsárga szín jelzi a kiszolgálás megkezdését, a zöld a befejezését*

A megszakításvezérlő a megszakításjelzés (IRQ) megjelenését követően a programszámláló értékét elmenti a verembe (ez már értelemszerűen a főprogramsorban következő utasításának címértéke). Itt fontos megjegyezni, hogy az utasítás végrehajtási folyamata nem megszakítható. Ez azt jelenti, hogy az éppen végrehajtás alatt álló utasítás közben érkező megszakításkérés kiszolgálása CSAK a végrehajtási folyamat vége után kezdődhet el. Ha arra gondolunk, hogy a tervezett processzorunk 6 órajelciklus alatt hajt végre egy utasítást, akkor

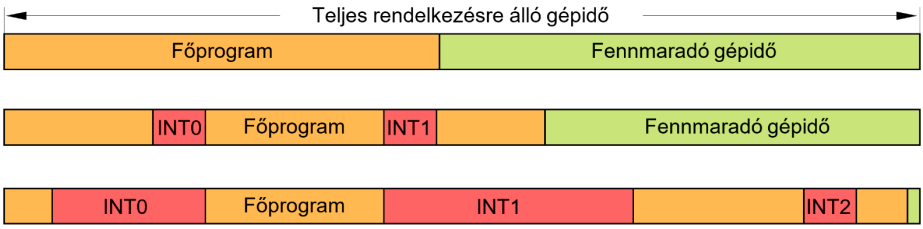
legrosszabb esetben 6 órajelciklus késleltetéssel kezdődhet el egy megszakításkérés kiszolgálása.

Ezután a programszámláló verembe elmentett címértéke a megszakítást kiszolgáló eljárás (programrész) első utasításának címével íródik felül. Az utasításvégrehajtás ettől az új címtől folytatódik. A kiszolgáló eljárás a munkaregiszter (ACCU) tartalmának és ezzel a ZERO jelzőbit állapotának elmentésével kezdődik. Ez biztosítja, hogy a megszakítást kiszolgáló eljárás végén nemcsak a programszámláló, hanem a munkaregiszter és a jelzőbit értéke is visszaállítható.

A kiszolgáló eljárásban a lehető legrövidebb és legoptimálisabb módon kell a kívánt hatást elérni, kimenőjelzést vagy csak értékmentést létrehozni, illetve megvalósítani. A kiszolgáló eljárás összeállításakor lehetőség szerint kerülni kell további eljárások „meghívását". A kiszolgáló eljárás az elmentett munkaregiszter-érték visszaállításával fejeződik be, így a ZERO jelzőbit is a megszakítás előtti állapotba áll be.

A kiszolgáló eljárás végét, kötelező módon, egy speciális utasításszó (VISSZATÉRÉS – RETFIE – REturn From Interrupt [with] Enable) jelöli. Végrehajtása során a megszakítást jelző bit értéke 0-ba áll, a rendszer újabb megszakításkérést fogadhat. A verembe elmentett címérték pedig visszaíródik a programszámlálóba.

Fontos megjegyezni, hogy a szimulációs környezetben létrehozott processzorban a megszakításeljárás végrehajtása közben a megszakításvezérlő nem képes újabb megszakításkérést fogadni és kiszolgálni. Erre csak az éppen futó kiszolgáló eljárás befejezése után kerülhet sor.



8.22. ábra. *A szabad gépidő és a processzorterheltség alakulása több megszakítás esetén*

A megszakításkérést kiszolgáló eljárás összeállítása előtt gondos tervezést igényel. Végrehajtása beágyazódik a főprogram futási idejébe, meghosszabbítva azt. A processzor terheltsége ennek függvényében megnő, a fennmaradó gépidő csökken. Ezt a jelenséget a 8.22. ábra szemlélteti. Az ábra első sorában a főprogram-megszakítás kiszolgáló eljárás nélkül fut le, sok szabad gépidőt hagyva. Ezt a főprogram bővítésére, változó hosszúságú ciklusai végrehajtásának időfedezetére, vagy energiatakarékos üzemmódba kapcsolásra lehet fordítani. A második sorban

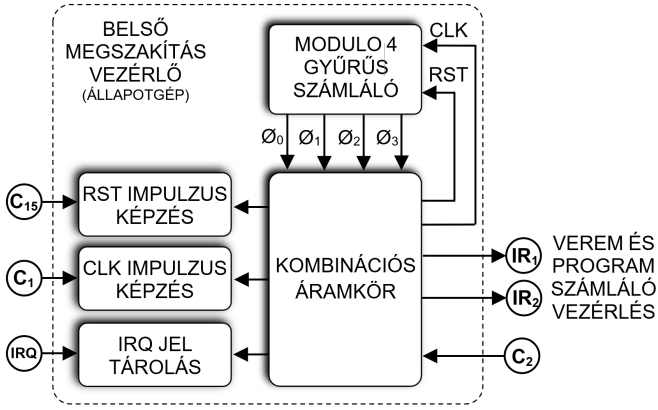
a főprogramot két, rövid idő alatt végrehajtható megszakításkérést kiszolgáló eljárás tagolja. A fennmaradó gépidő itt még optimális mértékű. A harmadik sorban egy olyan eset áll elő, amikor a főprogramot például három, változó hosszúságú végrehajtási időt igénylő megszakításkiszolgáló eljárás tagolja. Így a főprogram végrehajtási ideje annyira megnő, hogy alig marad gépidő ennek befejezéséhez. A legrosszabb eset akkor állhat elő, ha a főprogram a teljes rendelkezésre álló gépidő alatt nem tud legalább egyszer, teljesen lefutni.

A kiszolgáló eljárást alkotó utasításokat és ezzel a teljes végrehajtási idejét úgy kell meghatározni, hogy a főprogramfutás idején túl fennmaradó gépidő biztosan fedezze a főprogram részét képező, esetlegesen kitolódó várakozási vagy egyéb ciklusok végrehajtási idejét is. A megszakításkiszolgáló eljárásban törekednünk kell, hogy csak a megszakítást kiváltó eseménnyel kapcsolatos legfontosabb jelzést hozzuk létre, vagy csak azokat a változókat mentsük el vagy módosítsuk, amelyek ehhez feltétlen szükségesek lehetnek, minden más kapcsolódó részletfeladatot csak jelöljünk meg különböző jelzőbitekkel és fejezzük be a kiszolgáló eljárást. A részfeladatok végrehajtására egy későbbi időpontban, a főprogram végrehajtásának részeként térünk majd vissza.

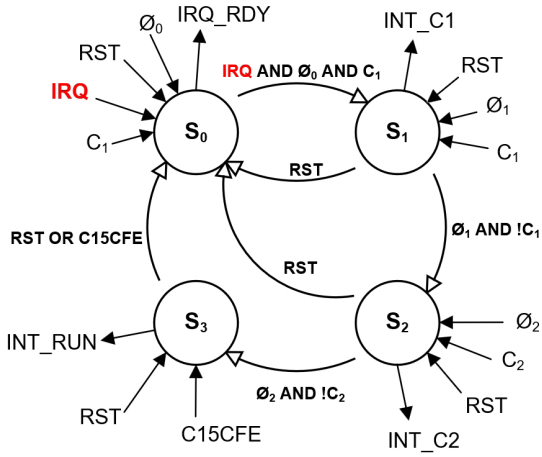
8.4.3. Belső megszakításvezérlő tervezése

A belső megszakításvezérlő feladata a processzormegszakítás bemenetének állapotkövetése és a kiszolgáló eljárás végrehajtásának kezdeményezése. Felépítését tekintve a 8.23. ábra szerint több funkcionális részre osztható. A D tárolókból kialakított IRQ-jel-figyelő és impulzusképző blokkokat egy kombinációs áramkör kapcsolja össze egy gyűrűs számlálóval. A vezérlő a megszakításbemenet (**IRQ**) állapotváltozásának (0-ról 1-re) eseményét egy tárolócellában (**IRQ JEL TÁROLÁS**) rögzíti, majd megvárja az éppen végrehajtás alatt álló utasítás befejezését. Az állapotgépként működő gyűrűs számláló a soron következő utasítás-végrehajtási ciklusban megjelenő, az utasítás-előkészítéshez szükséges vezérlőjelek hatására (C_1, C_2) a kombinációs áramkör közbeiktatásával speciális, a verem- és a programszámláló működtetéséhez szükséges vezérlőjeleket (IR_1, IR_2) állít elő.

Az állapotgép bemeneteit a megszakításjelzés (**IRQ**), az utasítás-előkészítő vezérlőjelek (C_1, C_2), a megszakításkérést kiszolgáló eljárás utolsó, speciális utasításának (**RETFIE**) végrehajtásához szükséges vezérlőjel (C_{15}) és a **RESET** (**RST**) jel képezik. Kimenetein a vermet és programszámlálót vezérlő jelzések (**IR1** – **INT_C1**, **IR2** – **INT_C2**), valamint az áramkör belső állapotait megjelenítő jelzések (**INT_RUN**, **IRQ_RDY**) állnak. Az állapotgép logikai sémáját az állapotokat jelző körök és az állapotok közötti átmeneteket jelző nyílvonalak alkotják (8.24. ábra). Az állapotkörökbe befele tartó nyílvonalak a bemeneteket, a kifele tartó vonalak pedig a kimeneteket jelölik. Az átmeneteket jelző nyílvonalak mellett mindig megjelenik az állapotváltást kiváltó logikai függvény.



8.23. ábra. *Belső megszakításvezérlő egység tömbvázlata*



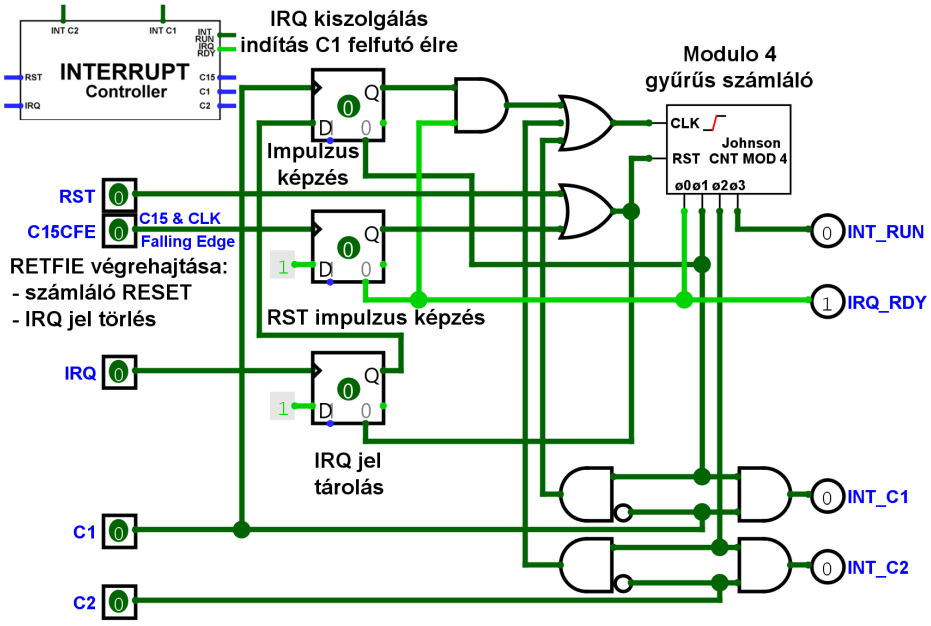
8.24. ábra. *Megszakításvezérlő állapotdiagramja*

A kimeneti jelek alakulása egy igazságtáblázatban (8.25. ábra) foglalható össze. A színessel kiemelt mezőkben látható bemeneti jel állapotok hatására az állapotgép egyik állapotból egy következőbe lép. A táblázat bemeneti oszlopai között ÉS, sorai között VAGY összefüggés van. A VAGY összefüggés az áramköri rajzban (8.26. ábra) az állapotgépet működtető gyűrűs számláló órajel (CLK) bemenete felé tartó jeleket gyűjti egybe (3 sor – 3 bemenet).

Aktuális állapot	Bemenetek IRQ	C_1	C_2	C_{15} CFE	RST	$Ø_3$	$Ø_2$	$Ø_1$	$Ø_0$	Következő állapot	Kimenetek IRQ RDY	INT_C1	INT_C2	IRQ_RUN	$Ø_3$	$Ø_2$	$Ø_1$	$Ø_0$
00	X	X	X	X	1	0	0	0	1	00	1	0	0	0	0	0	0	1
00	0	X	X	X	0	0	0	0	1	00	1	0	0	0	0	0	0	1
00	1	0	X	X	0	0	0	0	1	00	1	0	0	0	0	0	0	1
00	1	1	0	0	0	0	0	0	1	01	0	1	0	0	0	0	1	0
01	X	X	X	X	1	0	0	1	0	00	1	0	0	0	0	0	0	1
01	X	0	X	X	0	0	0	1	0	10	0	0	1	0	0	1	0	0
10	X	X	X	X	1	0	1	0	0	00	1	0	0	0	0	0	0	1
10	X	X	0	X	0	0	1	0	0	11	0	0	0	1	1	0	0	0
11	X	X	X	X	1	1	0	0	0	00	1	0	0	0	0	0	0	1
11	X	X	X	1	0	1	0	0	0	00	1	0	0	0	0	0	0	1

8.25. ábra. *Az állapotgép egyszerűsített igazságtáblázata*

A táblázatból észrevehető, hogy a gyűrűs számláló alapállapot (RST) bemenetét befolyásoló jel is a külső RST és a megszakításkiszolgáló eljárás végét jelző C_{15}CFE jelek között képezett, második VAGY függvény eredménye (2 sor – 2 bemenet). A kapcsolási rajzban szereplő D tárolók az órajelbemenetük felfutó élére állítják be a kimeneti értékeiket, ezért a kialakuló állapotok eseményfüggősége biztosított.



8.26. ábra. *Megszakításvezérlő egység kapcsolási rajza és szimbóluma*

Az így kialakított belső megszakításvezérlő lehetővé teszi egy megszakítás-kérés-kiszolgálás automatikus megkezdését, lebonyolítását (ez már a processzor utasítás-végrehajtó részének a feladata), majd a rendszer alapállapotba hozásával a megszakított főprogram folytatását.

- interrupt system
- function of the interrupt controller

8.5. Programvezérlő utasítások kialakítása mikrokódos vezérlőegységben

Az előző két alfejezetben leírt processzorrészegységek, a verem- és a belsőmegszakítás-vezérlő megteremtik a lehetőségét a programvezérlő utasítások végrehajtásának. Továbbá kihasználhatjuk a mikrokódos vezérlőegység könnyű bővíthetőségét, és a mikroprogramtárba elhelyezett újabb mikroutasításokkal úgynevezett programvezérlő utasítások hozhatók létre. Ezek végrehajtása újabb vezérlőjeleket igényel. A már ismert módszer szerint meghatározzuk ezeket az utasításokat, és táblázatos formában (8.27. ábra) felírjuk az elképzelt funkcióikat, valamint a végrehajtásuk során érintett ki- és bemeneti egységeket.

Utasítás neve	Leírás	Bemenet	Kimenet	Utasítás kódja
CALL cím	Eljárás hívás, megadott címre ugrással és STACK←PC+1	**DR, AR**	**AR, STACK**	**1100**
RETURN	Visszatérés eljárásból verembe elmentett címre PC←STACK	**STACK**	**AR**	**1101**
RETFIE	Visszatérés megszakítás eljárásból verembe elmentett címre PC←STACK	**STACK**	**AR**	**1110**

8.27. ábra. *Függvényhíváshoz és megszakításrendszerhez kapcsolódó utasítások táblázata*

Az eljáráshívás vagy CALL utasítás végrehajtásával a főprogramba foglalt eljárás kezdőcíme betöltődik a címregiszterbe (AR), ugyanakkor a programszámláló (PC) értéke a verembe (STACK) másolódik. Az eljáráshívás végét jelző visszatérés vagy RETURN utasítás végrehajtásakor a veremben tárolt címérték visszamásolódik a címregiszterbe. Ugyanez történik a megszakításkérést kiszolgáló eljárás végét jelző RETFIE (Return From Interrupt with Enable) utasítás végrehajtásakor, de

ennél a címérték visszaállítása után a megszakítást jelző bemenet tárolt állapota is törlődik.

A processzor architektúráját kiegészítő belsőmegszakítás-vezérlőt az IC rövidítéssel (Interrupt Controller), a veremvezérlőt SC rövidítéssel (Stack Controller) jelöljük (8.28. ábra). Sínszélesség tekintetében a veremvezérlő a 8 bites címsín méretéhez alkalmazkodik. A belsőmegszakítás-vezérlőnek nincs kapcsolata az architektúra sínrendszereivel, csak bizonyos vezérlőjelek kialakításáért felel.

Megnevezés	Sín szélesség	Magyarázat
IC	**-**	**I**nterrupt **C**ontroller – megszakítás vezérlő, a megszakítás eljárás automatikus meghívására szolgáló áramkör
SC	**8 bit**	**S**tack **C**ontroller – verem vezérlő, a programszámláló pillanatnyi értékének eltárolását és visszaállítását biztosító áramkör

8.28. ábra. *Függvényhíváshoz és megszakításrendszerhez kapcsolódó alkotóelemek listája*

Az újabb utasítások végrehajtásához a már meglévő vezérlőjelek csoportját továbbiakkal bővítjük. Ezek a jelzések a veremvezérlő és a belsőmegszakítás-vezérlő működését szinkronizálják az utasítás-végrehajtás folyamatában. Ezeket is táblázatos formában rögzítjük (8.29. ábra).

Vezérlő jel	Adat mozgás	Magyarázat
C_{13}	**ST_CNT+1** **STACK←PC**	Veremszámláló értékének növelése. Programszámláló értékének beírása a verembe (STACK)
C_{14}	**PC←STACK** **ST_CNT-1**	A veremben tárolt címérték beírása a programszámlálóba. Veremszámláló értékének csökkentése.
C_{15}	**PC←STACK** **ST_CNT-1** **IRQ CLR**	A veremben tárolt címérték beírása a programszámlálóba. A megszakításkérést (IRQ) jelző bit törlése.

8.29. ábra. *Függvényhíváshoz és megszakításrendszerhez kapcsolódó vezérlőjelek táblázata az adatmozgási irányok meghatározásával*

A c_{13}-as jelzés a CALL utasítás végrehajtásakor aktiválódik, és a veremvezérlő mutatójának (vagy számlálójának) növelését (**ST_CNT+1**), valamint a programszámláló értékének veremregiszterbe való elmentését (**STACK←PC**) kezdeményezi. A c_{14}-es jelzés a RETURN utasítás végrehajtásakor aktiválódik, és a fordított folyamatot, a veremvezérlő mutatójának csökkentését (**ST_CNT-1**), valamint a programszámláló értékének veremregiszterből való visszaállítását (**PC←STACK**) kezdeményezi.

A c_{15}-ös jelzés a megszakításkiszolgáló eljárás végét jelző RETFIE utasítás végrehajtásakor aktiválódik. A c_{14}-es jelzéshez hasonlóan, a veremvezérlő mutatójának csökkentését (**ST_CNT-1**), valamint a programszámláló értékének veremregiszterből való visszaállítását (**PC←STACK**) kezdeményezi, kiegészítve a megszakításkérést jelző bit (irq) törlésével (ezt egy úgynevezett képzett jelzés, a c_{15} jelből a rendszer órajel lefutó élével szinkronizálva kapott **c15cfe** – c_{15} clock Falling Edge hatása váltja ki).

A kívánt jelzések létrehozásához a mikroprogramtár tartalmát a **CALL**, **RETURN** és **RETFIE** utasítások mikrolépéseinek vagy mikroutasításainak bináris formájával kell kibővíteni. Ezt a már tárgyalt módon (8.2.3. alfejezet), táblázatos formában rögzítjük (8.30. ábra). A **CALL** utasítás végrehajtása két vezérlőjelet (c_{13}, c_6), míg a **RETURN** és **RETFIE** utasítások egy-egy vezérlőjelet (c_{14} és c_{15}) igényelnek. Az így kapott mikroutasításokat hozzáfűzzük a mikroprogramtár tartalmához. A mikrokódos vezérlőegységben kialakított jelzéseket elvezetjük a megfelelő processzoralegységekig.

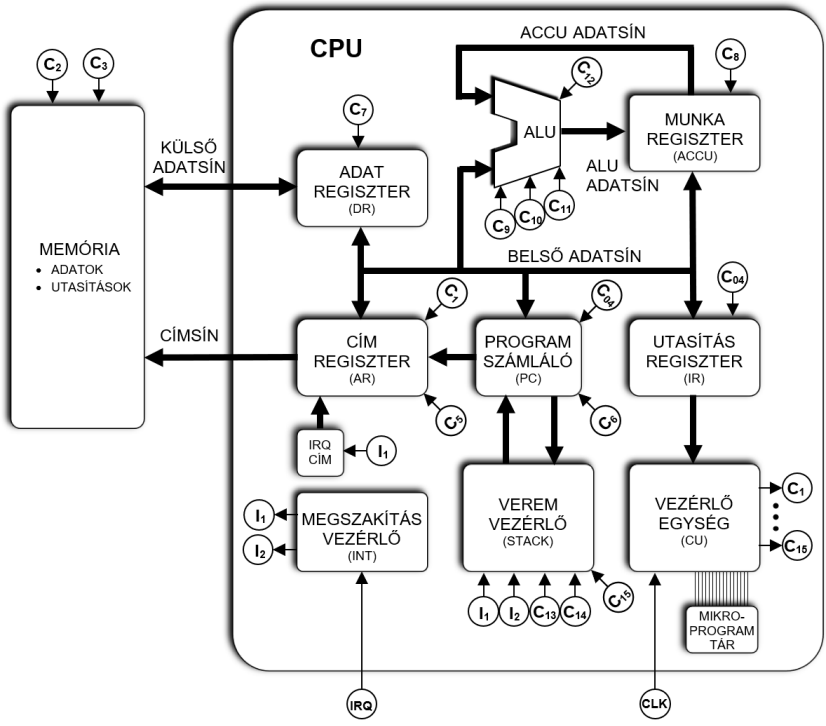
Tár cím	Memória tartalom			Utasítás	Mikro-művelet	Vezérlő jel
	AS	$S_2S_1S_0$	CTRL			
011000	0	000	0000	CALL	-	-
011001	0	000	1100		ST_CNT+1 STACK←PC	C13
011010	0	100	0110		PC←DR(ADDR)	C6
011011	0	000	0000		-	-
011100	0	000	0000	RETURN	-	-
011101	0	000	0000		-	-
011110	0	100	1110		PC←STACK ST_CNT-1	C14
011111	0	000	0000		-	-
100000	0	000	0000	RETFIE	-	-
100001	0	000	0000		-	-
100010	0	100	1111		PC←STACK ST_CNT-1 IRQ CLR (C15CFE)	C15
100011	0	000	0000		-	-

8.30. ábra. *Eljáráshívás és megszakításkérés-kezelő utasításokkal bővített mikroprogramtár szerkezete és tartalma*

8.6. 8 bites, mikrokódos központi végrehajtó egység (CPU) összeállítása

Az eddig megtervezett és létrehozott alegységekkel, a kibővített utasításkészlettel rendelkező mikrokódos vezérlőegységgel, veremvezérlővel és belsőmegszakítás-vezérlővel kiegészítjük a 8.6. ábra alapján elképzelt processzorarchitektúránkat. Az így kapott áramkör (8.31. ábra) az előző változathoz hasonló szélességű (8 bit) belső sínrendszerrel és regiszterpalettával (cím, adat, utasítás és munkaregiszter) rendelkezik.

A mikrokódos vezérlőegység (CU) saját, belső cím- és adatsínnel rendelkezik, amely nincs kapcsolatban a processzor saját, belső sínrendszerével. Az egység kimenetei közvetlenül a vezérlő jelvezetékekre csatlakoznak. A veremvezérlő (STACK) funkcionálisan a programszámlálóhoz kapcsolódik, fizikai kapcsolatrendszerüket a későbbiek során tárgyaljuk. A megszakításvezérlő (INT) nem csatlakozik a belső cím- és adatsínrendszerhez, csak vezérlőjeleket szolgáltat, kimenetei közvetlenül a vezérlő jelvezetékekre csatlakoznak.

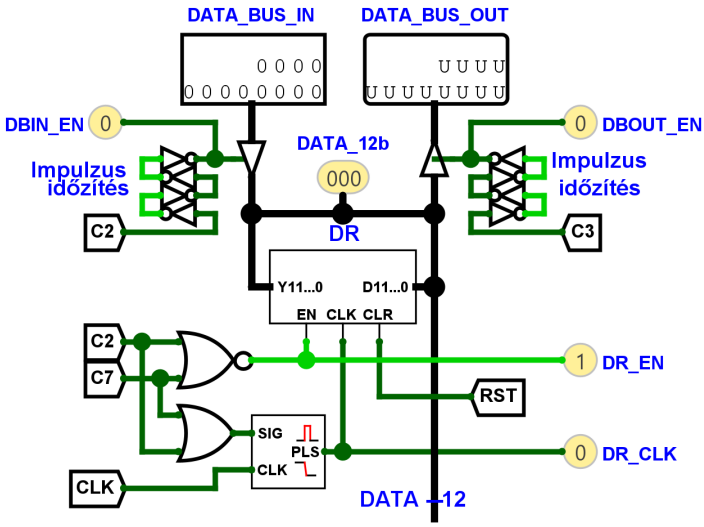


8.31. ábra. *Mikrokódos vezérlőegységgel, verem- és megszakításvezérlővel rendelkező processzor vázlatos rajza*

A felvázoltak alapján, szimulációs környezetben összekapcsoljuk a létrehozott alegységeket. A processzor besorolását tekintve egy Neumann-elvnek megfelelően felépített, 1 című gép. A kevés utasításszámot (15), azonos utasításszóhosszat (12bit) és azonos végrehajtási ciklusidőt (6 ütem) tekintve egy MISC-rendszer sajátosságait mutatja.

A továbbiakban elemezzük az így kialakított processzor architekturális sajátosságait. A szimulációs modell segít megérteni azokat a technikai részleteket, amelyek alapján a processzor működni fog. Láthatjuk a sínek elhelyezkedését, kapcsolatát a különböző alegységekkel, követhetjük a különböző jelzések útját (címkékkel egyszerűsített ábrázolás), megfigyelhetjük a processzor részegységeinek órajellel való belső szinkronizációját elősegítő áramköri megoldásokat.

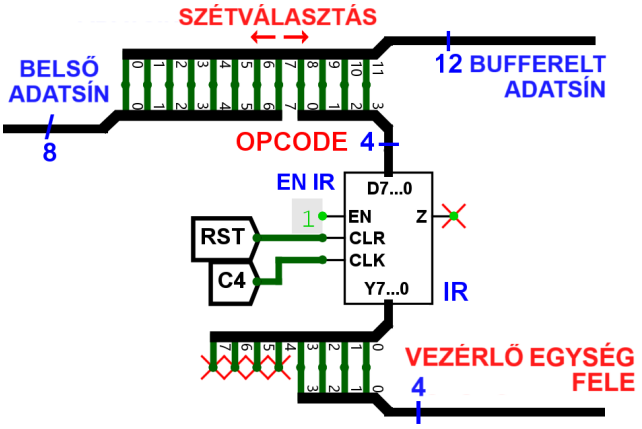
Elsőként a szimulációs környezetben létrehozott **adatregisztert** (DR) és a hozzá tartozó áramköröket (8.32. ábra) vizsgáljuk. Az adatregiszter egy külön adat ki- (D_{11}-D_0) és bemenetekkel (Y_{11}-Y_0) rendelkező tárolócella. A kapcsolási rajzban az adatregiszter kimenetei és bemenetei össze vannak kötve egymással, és így kapcsolódnak a belső, kétirányú adatsínre. Az adatregiszter a külső adatsínre két ellentétes irányú, 12 bites sínmeghajtón keresztül csatlakozik. A sínmeghajtók engedélyező jelei több, egymással sorba kapcsolt tagadókapun haladnak keresztül. A kapuk átfutási ideje összeadódik, és így a sínmeghajtók engedélyező jeleinek késleltetése finomhangolható (Impulzus időzítés). Erre az időzítés finomhangolásra a számítógéprendszer összeállításánál lesz majd szükség.



8.32. ábra. *A mikroprocesszor adatregiszterének (DR) és vezérlőjeleinek kapcsolási rajza*

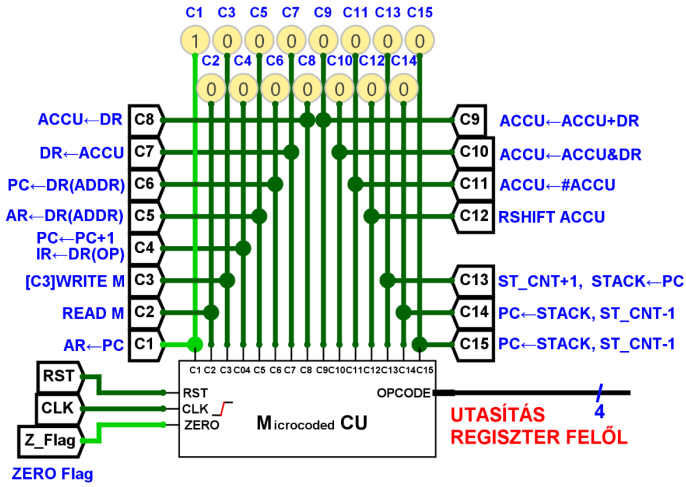
Az adatregiszter kimeneteinek a belső adatsínre (DATA **12**) kapcsolódását engedélyező EN bemenet mindig aktív, kivéve, ha a c_2 (READ MEMORY) VAGY c_7 (DR←ACCU) vezérlőjelek valamelyike aktív. Ilyenkor az adatregiszter kimenetei lekapcsolódnak a belső adatsínről, de az adatsínre kapcsolt bemenetein megjelenő bináris kód a c_2 VAGY c_7 vezérlőjelek hatására, mindig az órajel (CLK) lefutó élével szinkronban beíródik. az adatregiszter kimenete a c_3 jel hatására a külső adatsínre kapcsolódhat. A RST általános hatású, a teljes rendszert érintő, azt alapállapotba hozó jelzés.

A processzor **belső adatsínje** 12 bit széles. Szétszálazása (splitting) a szimulációs környezetben a 8 bites munkaregiszter (ACCU), illetve a 8 bites utasításregiszter (IR) felé a megfelelő méretűre választott sín kibontó „fésűk" segítségével történik. Az utasításregiszter kimenete mindig aktív (EN=1) és a vezérlőegység bemenetére kapcsolódik. A belső adatsínről az utasítás-végrehajtás előkészítő szakaszának végén kiadott c_4-es jelzés hatására vesz be adatot, esetében az utasítás kódját. A kapcsolási rajz részleten megfigyelhető a 0-tól 11-ig számozott belső adatsín (DATA **12**) kibontó, amelynek alsó 8 bitje (DATA **8**) a munkaregiszter felé halad tovább, a felső 4 bitje (OPCODE **4**), növekvő helyértéki sorrendben az utasításregiszter bemeneti adatsínjét képezi.



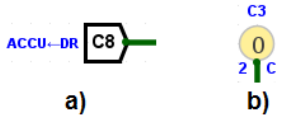
8.33. ábra. *A mikroprocesszor utasításregiszterének (IR) és vezérlőjeleinek kapcsolási rajza, valamint az adatsín vonalainak csoportosítása*

Az utasításregiszter kimenetei rácsatlakoznak a **mikrokódos vezérlőegység** (**Microcoded CU** 8.34. ábra) bemenetére (OPCODE). A vezérlőegység további három bemenete az alapállapotba hozó jelzés (RST), az órajel (CLK) és a zéró jelzőbit (Z_FLAG).



8.34. ábra. *A mikroprocesszor vezérlőegységének (CU) és vezérlőjeleinek kapcsolási rajza*

A vezérlőegység kimenetei a tervezési folyamatban megállapított jelek. A címkézett ábrázolásmód előnye, hogy a kapcsolási rajzot nem hálózzák be a követhetetlen útvesztővé gubancolódó vezetékvonalak. Minden vezérlőjel egy címkét kap (8.35. ábra), ennek alapján azonosítható a kapcsolási rajzban. Ahol ez a címke felbukkan, oda kapcsolódik az adott jelvezeték. Egy címke több helyen is megjelenhet, ebben az esetben a jelvezeték többfelé is elágazik. Minden vezérlőjelhez egy állapotjelző mező is tartozik (8.35. ábra). A kis, sárga körökből az adott jelvezeték pillanatnyi állapotának bináris értéke olvasható ki.



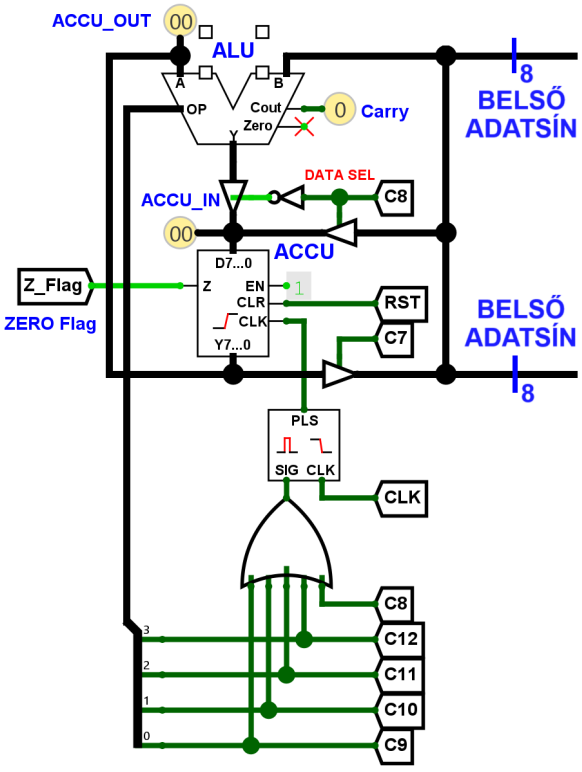
8.35. ábra. *A kapcsolási rajz olvashatóságát elősegítő címke és állapotjelző mező szimbólumai*

A **munkaregisztert** (ACCU) és **aritmetikai-logikai egységet** (ALU) összefüggő rendszerként tárgyaljuk (8.36. ábra). Az aritmetikai és logikai műveletek egyik operandusa mindig a munkaregiszterből kerül az ALU egyik bemenetére, az eredmény szintén a munkaregiszterbe kerül vissza. Az ALU másik bemenete a belső adatsínre kapcsolódik, így műveletvégzéskor a másik operandus a bemeneti adatregiszterben (DR) tárolt érték lesz. Az ALU a C_9, C_{10}, C_{11}, C_{12} vezérlőjelek hatására az

ADD (összeadás), AND (és művelet), NOT (ACCU érték tagadása) és RSH (jobbra tolás) műveleteket tudja elvégezni.

A munkaregiszter bemenete egy-egy vezérelhető sínleválasztó áramkörön keresztül kapcsolódik az ALU kimenetére, illetve a belső adatsínre. A sínmeghajtókat a c_8 jel ellenütemben vezérli. Amikor a c_8 1-es, a munkaregiszter bemenete a belső adatsínre kapcsolódik, és az órajelütem lefutó élére szinkronizálva adatot vesz át a bemeneti adatregisztertől. Amikor a c_8 0-ás, az órajelütem lefutó élére szinkronizálva, valamelyik művelet kiválasztó jel hatására a munkaregiszter eltárolja az ALU kimenetén megjelenő értéket.

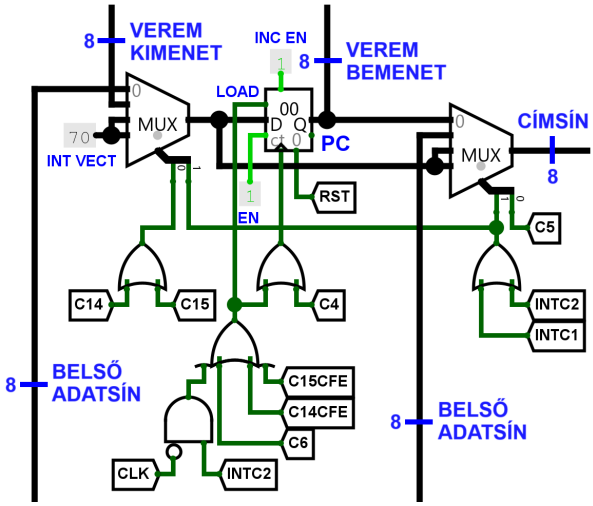
A munkaregiszter kimenete szintén sínleválasztó áramkörön keresztül kapcsolódik a belső adatsínre, amelyet a c_7 jel vezérel. Amikor c_7 1-es, a munkaregiszter kimenete a belső adatsínre kapcsolódik és beíródhat az adatregiszterbe. Amikor c_7 0-ás, a munkaregiszter kimenete nem kapcsolódik a belső adatsínhez.



8.36. ábra. *A mikroprocesszor munkaregiszterének (ACCU), aritmetikai-logikai egységének (ALU) és vezérlőjeleinek kapcsolási rajza*

A munkaregiszterben egy 8 bemenetű NEM-VAGY kapu figyeli a cellákban eltárolt értékeket, és a z kimenete azonnal 1-be vált, ha minden cella értéke egyidejűleg 0. ez képezi a processzor Z-FLAG (nulla eredmény) állapotjelző bitjét.

A **programszámláló** (PC) egy 8 bites, bináris, kétirányú számláló, párhuzamos adatbetöltési lehetőséggel (8.37. ábra). A számlálás iránya rögzített (INC EN=1), minden órajelütemre 1-gyel nő a számlálóban lévő érték, és a számláló mindig aktív (EN=1). Alapállapotba hozás (RST) esetén a tartalma 0 lesz. A programszámláló állapotait táblázatos formába rendezve a 8.38. ábra a) része mutatja. A számláló be- és kimenetéhez egy-egy multiplexer áramkör (MUX) csatlakozik. A bemeneti multiplexer három irányból érkező érték közül választ az éppen aktív vezérlőjelek függvényében. Továbbíthatja a veremből érkező, a belső adatsín felől érkező vagy, megszakítás kérés esetén, a 0x70 hexadecimális formában megadott, állandó címértéket.



8.37. ábra. *A mikroprocesszor programszámlálójának (PC) és vezérlőjeleinek kapcsolási rajza*

A kimeneti multiplexer szintén három irányból érkező érték közül választ az éppen aktív vezérlőjelek függvényében. A címsín értéke lehet a programszámláló kimeneti, a belső adatsín, vagy, megszakításkérés esetén, a 0x70 hexadecimális formában megadott, állandó bemenet irányából kapcsolt címérték. A multiplexerek kapcsolási kombinációit táblázatos formába rendezve a 8.38. ábra b) és c) része mutatja.

Aktuális állapot	Bemenetek RST	CLK	C_4	C_6	C_{14} CFE	C_{15} CFE	INT_C2
RESET	1	X	X	X	X	X	X
LOAD	0	0	X	0	0	1	0
LOAD	0	0	X	0	1	0	0
LOAD	0	0	X	1	0	0	0
LOAD	0	1→0	X	0	0	0	1
INC	0	0	1	0	0	0	0
INC	0	0	X	0	0	1	0
INC	0	0	X	0	1	0	0
INC	0	0	X	1	0	0	0
INC	0	1→0	X	0	0	0	1

a)

Kiválasztás INT_C1	INT_C2	C_{14}	C_{15}	MUX Kimenet
0	0	0	0	BELSŐ ADATSÍN
0	0	0	1	VEREM KIMENET
0	0	1	0	VEREM KIMENET
0	1	X	X	0x70
1	0	X	X	0x70

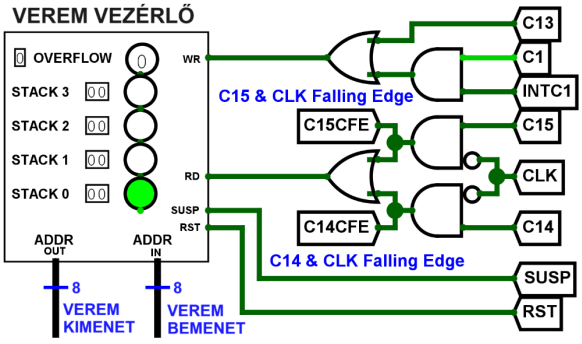
b)

Kiválasztás INT_C1	INT_C2	C_5	MUX Kimenet
0	0	0	PC KIMENET
0	0	1	BELSŐ ADATSÍN
0	1	X	0x70
1	0	X	0x70

c)

8.38. ábra. *A programszámláló (a), valamint be (b) és kimeneti (c) multiplexereinek kapcsolási táblázatai a vezérlőjelek függvényében*

A **veremvezérlő** (STACK **Controller**) bemenete közvetlenül a programszámláló kimenetére csatlakozik (8.39. ábra). A kimenete a programszámláló bemeneti multiplexerének 1-es (01 kiválasztó kombinációval elérhető) bemenetére csatlakozik. A verembe való címértékírás (WR) VAGY megszakításkérés (IRQ) esetén a belső megszakításvezérlő INTC1 ÉS a mikrokódos vezérlőegység c_1-es jeleire, VAGY a CALL utasítás végrehajtása során aktiválódó c_{13}-as jel hatására következik be.

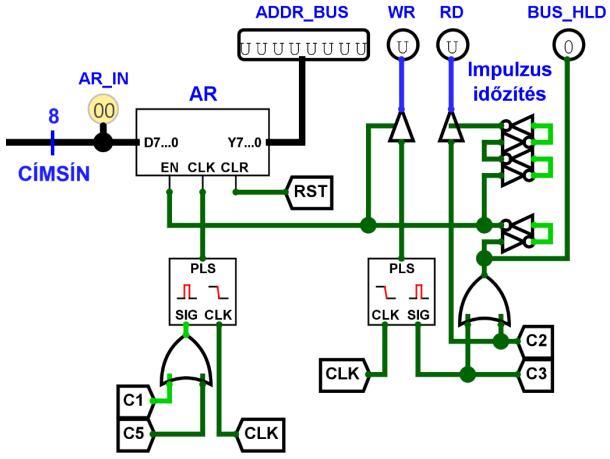


8.39. ábra. *A mikroprocesszor veremkezelő áramkörének (SC) és vezérlőjeleinek kapcsolási rajza*

A veremből történő címérték-kiolvasás az órajel (CLK) lefutó élével szinkronban történik a C_{14} vagy C_{15}-ös jelek hatására, a RETURN vagy RETFIE utasítások végrehajtása során. Ennek a három jelnek (CLK, C_{14}, C_{15}) a kombinációja hozza létre a programszámláló-beíráshoz (LOAD) és a belsőmegszakítás-vezérlő állapotgépének alapállapotba hozásához szükséges C14CFE és C15CFE jelzéseket.

A veremvezérlő SUSP kimenete a verem túlcsordulásállapotát jelzi és rögzíti (több mint négy egymásba ágyazott eljáráshívás vagy megszakításkérés gyűlt össze). Ez a jel csak a teljes számítógéprendszer alapállapotba hozásával (RST) oldható fel.

A **címregiszter** a belső és a külső címsínt kapcsolja össze (8.40. ábra). Bemenete a programszámláló kimeneti multiplexeréhez, míg kimenete a külső címsínhez kapcsolódik. A belső címsínen megjelenő értéket az órajel (CLK) lefutó élével szinkronizált C_1 és C_5 jelek hatására tárolja el az utasítás-végrehajtási folyamat előkészítési fázisában. Kimenetét a C_2 és C_3 jelek hatására kapcsolja a külső címsínre az utasítás-végrehajtási folyamat végrehajtási fázisában.



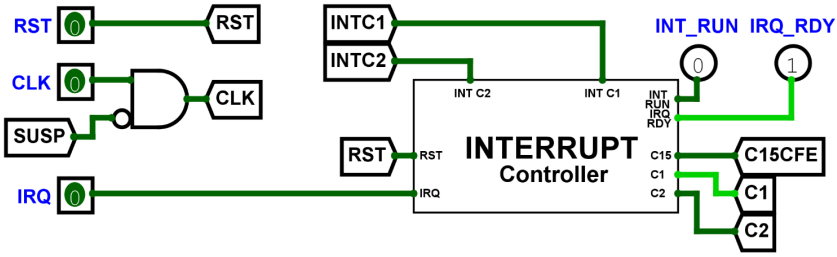
8.40. ábra. *A mikroprocesszor címregiszterének (AR) és vezérlőjeleinek kapcsolási rajza*

Ez a két jel (a C_2 és C_3) további jeleket hoz létre, amelyek a számítógéprendszer külső vezérlőjeleit alkotják. A WR a külső adatsín irányát jelzi, a rajta megjelenő adat beírandó a megcímzett perifériába. A RD esetében a külső adatsínen megjelenő értéket a processzor az adatregiszterébe tárolja el. Mindkét jel (a WR és RD) a számítógéprendszer processzorához kapcsolt perifériáknak szól. A BUS_HLD jelzés a külső cím- és adatsín foglaltságát jelzi a későbbiekben létrehozandó közvetlen memóriahozzáférés-vezérlő áramkörnek (DMAC). A jelzés értéke alapján (0 vagy 1) állapítja meg, hogy mikor bonyolíthat le adatátvitelt két periféria között, a processzor közbeavatkozása nélkül.

A WR és RD jelzések sínmeghajtó áramkörök közbeiktatásával haladnak a számítógépes rendszer sínrendszere felé. Amikor a BUS_HLD aktív, ezek a jelzések is aktívak. Amikor a BUS_HLD nem aktív, a WR és RD jelzések lekapcsolódnak a külső vezérlősínről, így a DMA-vezérlő szabadon használhatja a külső cím-, adat- és vezérlősíneket.

A címregiszter kimenete, illetve a RD jelzés kimenetvezérlése a teljes számítógép sínrendszer-összehangolásához szükséges késleltetések (Impulzus időzítés) után lesz aktív. Az időzítés finomhangolására használt tagadó kapuk sorozatának átfutási ideje összeadódik, ez biztosítja, hogy a számítógéprendszerben a jelzések, illetve kimenetengedélyezések időrendi sorrendben egymás után, eltolva következzenek. A felhasznált tagadó kapuk száma mindig páros, legkevesebb kettőt kapcsolunk sorba, így a rajtuk áthaladó jelzés megtartja eredeti logikai értékét.

A **belsőmegszakítás-vezérlő** nem kapcsolódik közvetlenül a cím- és adatsínekhez. A processzor belső működéséhez szükséges jelzéseket figyel és állít elő (8.41. ábra).

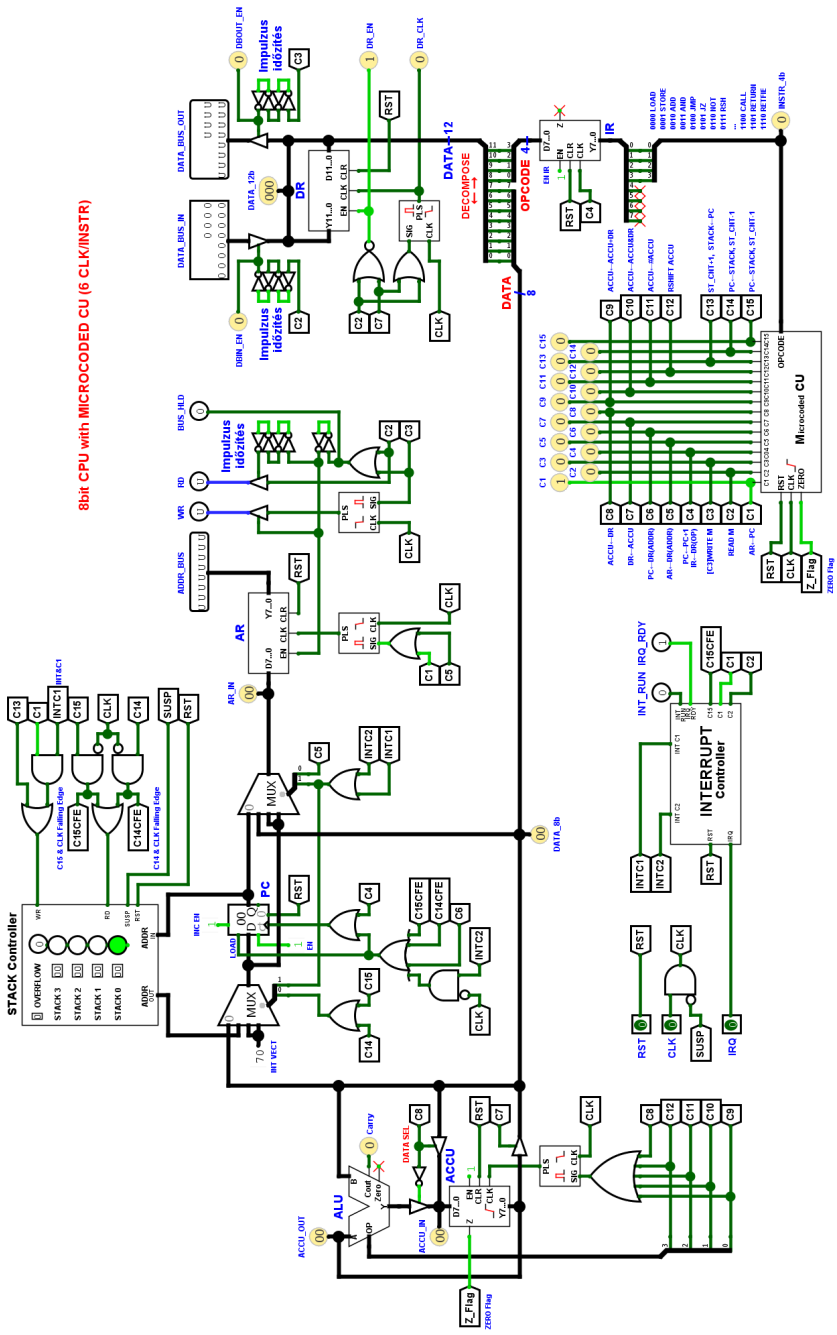


8.41. ábra. *A mikroprocesszor megszakításkezelő áramkörének (IC), órajelbemenetének (CLK) és vezérlőjeleinek kapcsolási rajza*

Elsődleges az IRQ bemenet, az áramkör mindaddig nem fejti ki hatását, amíg ez a bemenet nem aktiválódik (0-ból 1-be való átmenettel). Ekkor az áramkör megvárja az éppen végrehajtás alatt álló utasítás elvégzését, majd a következő végrehajtási ciklussal elkezdi a megszakításkérés kiszolgálását a C_1 és C_2 jelek megfigyelésével és az INTC1, majd az INTC2 jelzések aktiválásával. Ezt követően kivárja a megszakításkérést kiszolgáló eljárás utolsó utasítását (RETFIE), amelynek a végrehajtása során aktiválódó C15CFE jelzés hatására törli a megszakításkérést jelző bitet tároló celláját, és alapállapotba hozza a belső állapotgépét.

Ugyanebben a kapcsolási rajz részletben láthatjuk **a processzor órajelét** befolyásoló SUSP-jelzés szerepét: a verem túlcsordulása esetén tiltja az órajel továbbhaladását a processzor belső áramkörei felé, a processzor működése leáll.

Működése során a belső megszakításvezérlő áramkör két szemléltető jelzést is létrehoz, az IRQ_RDY-t, amikor alapállapotban van és megszakításkérés-jelzésre vár, és az INT_RUN-t, ami a megszakításkérést kiszolgáló eljárás futását jelzi és ennek befejezéséig aktív állapotban marad.

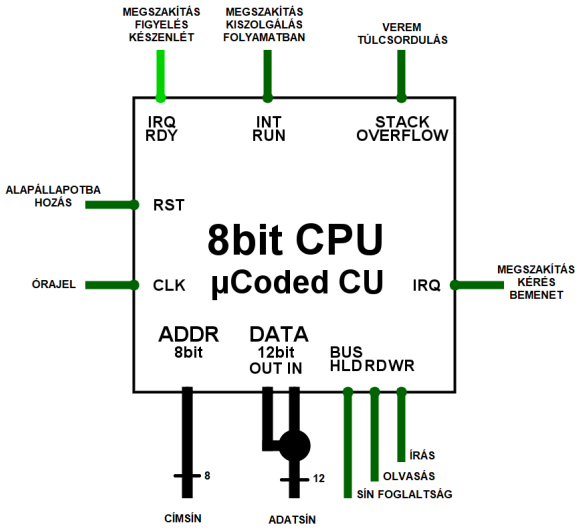


8.42. ábra. *A 8 bites mikroprocesszor kapcsolási rajza*

Az architekturális sajátosságok elemzése után tanulmányozható az összeállított processzormodell kapcsolási rajza. Az ábrát terjedelme miatt 90 fokkal jobbra fordítva szemléljük. A jobb felső résztől indulva bal felé, elsőként az adatsín csatlakozik az adatregiszterhez, és szétszálazva tovább halad lefele az utasításregiszter irányába. Az ábra közepétől balra tovább folytatódik az adatsín, melynek függőleges leágazásai elérik a programszámláló be- és kimeneti multiplexereit, majd egészen bal oldalon az ALU-hoz és a munkaregiszter (ACCU) sínmeghajtójához kapcsolódik. Az ábra jobb alsó felében a mikrokódos vezérlőegység, középen a belsőmegszakítás-vezérlő, míg a bal alsó sarokban az ALU és a munkaregiszter-vezérlő jeleinek csokrai láthatók.

Megfigyelhető, hogy az adatsínhez képest az ábra felső és középső részében látható, a programszámláló kimeneti multiplexere és a címregiszter között húzódó címsín hossza sokkal rövidebb, ez a sín nem ágazik el a processzor belsejében, és akár az adatsín, egy sajátos funkciójú leágazásának is tekinthető.

A tervezési fázis utolsó mozzanataként a kialakított processzorhoz, a többi belső áramköréhez hasonlóan, egy absztrakt szimbólumot rendelünk (8.43. ábra). Ez a szimbólum a ki- és bemenő sínek, valamint jelek funkcióját és elhelyezkedését rögzíti, és lehetővé teszi, hogy a tovább bővülő számítógéprendszer kapcsolási rajzát egy magasabb értelmezési szinten a fölösleges részletek elhagyásával tanulmányozhassuk.



8.43. ábra. *A 8 bites processzor szimbóluma*

- 8bit CPU design
- registers of a CPU

8.7. Gépi kódú és assembler programozás. A fordítóprogram működése

A processzor elképzelt működését programok futtatásával ellenőrizzük. Ha jól építettük fel a rendszert, minden utasítást helyesen hajt végre. Ebben az esetben az utasítások végtelen kombinációjából álló programokat is helyesen futtatja.

Egy processzor programozásának több szintje lehetséges, függetlenül attól, hogy az adott processzort egy szimulációs környezetben vagy fizikailag megvalósított alkatrészként kezeljük. A legalacsonyabb szint a gépi kódú programozás. Ilyenkor a programozó egy táblázat (például 8.1. ábra) segítségével azonosítja az egyes utasításokat jelentő, szimbolikus értelmű szavaknak (mnemonikoknak) megfelelő, hexadecimális számokat, és ezekkel tölti fel a központi memóriát. A feltöltési forma értelmezés kérdése, a számítógéprendszerben az utasítások, adatok bináris formában terjednek. A feltöltés után a processzor végrehajtja ezeket az utasításokat. A 8.44. ábra egy gépi kódú programrészletpéldát mutat be. A táblázat első két oszlopa a memóriacímet és a gépi kódú program utasításait rögzíti. A további oszlopok az értelmezést segítik, megjelölve az adott utasítás kódját, az adat értékét és az operandus címét.

Memória cím	Gépi kód	Utasítás kód	Adat	Cím
0x00h	0x009h	0x00h	-	0x09h
0x01h	0x20Ah	0x02h	-	0x0Ah
0x02h	0x10Bh	0x01h	-	0x0Bh
0x03h	0x30Ch	0x03h	-	0x0Ch
0x04h	0x10Bh	0x01h	-	0x0Bh
0x05h	0x412h	0x04h	-	0x12h
…	…	…	…	…
0x09h	0x055h	-	0x55h	-
0x0Ah	0x011h	-	0x11h	-
0x0Bh	0x000h	-	0x00h	-
0x0Ch	0x00Fh	-	0x0Fh	-

8.44. ábra. *Gépi kódban írt programpélda*

A programrészlet a 0x09h címről betölt (LOAD) egy értéket (0x55h) a munkaregiszterbe (ACCU). Ehhez hozzáadja (ADD) a 0x0Ah címen lévő értéket (0x11h). Az eredményt (0x66h) kiírja (STORE) a 0x0Bh címre (amely lokáció eredeti értéke 0x00h). Mivel az eredmény továbbra is a munkaregiszterben marad, ezen további műveletek végezhetők. Így a következő utasítás ÉS műveletet (AND) végez a munkaregiszterben tárolt érték (0x66h) és a 0x0Ch címen található érték

(0x0Fh) között. Az eredményt (0x06h) kiírja (STORE) a 0x0bh címre, felülírva ezzel az előzetesen ott tárolt értéket (0x66h), végül egy ugró utasítást (JMP) hajt végre a 0x12h címre.

Egy magasabb szinthez szükség van egy fordítóprogramra. Ilyen esetben a programozó egy szövegszerkesztő szoftver segítségével létrehoz egy szöveges állományt, amelynek tartalmaznia kell azokat a szimbolikus jelentésű szavakat (assembler menemonikok vagy szimbólumok), amelyek az egyes hexadecimális számoknak megfelelnek. Ezeknek a szavaknak a végtelen számú kombinációjából alakítható ki egy adott feladat elvégzését leíró eljárás vagy algoritmus. A 8.45. ábra a fenti, gépi kódban írt programpélda assemblerben írt változatát mutatja.

Memória cím	Assembly utasítások
0x00h	**LOAD** from **0x09h**
0x01h	**ADD** with **0x0Ah**
0x02h	**STORE** to **0x0Bh**
0x03h	**AND** with **0x0Ch**
0x04h	**STORE** to **0x0Bh**
0x05h	**JMP** to **0x12h**

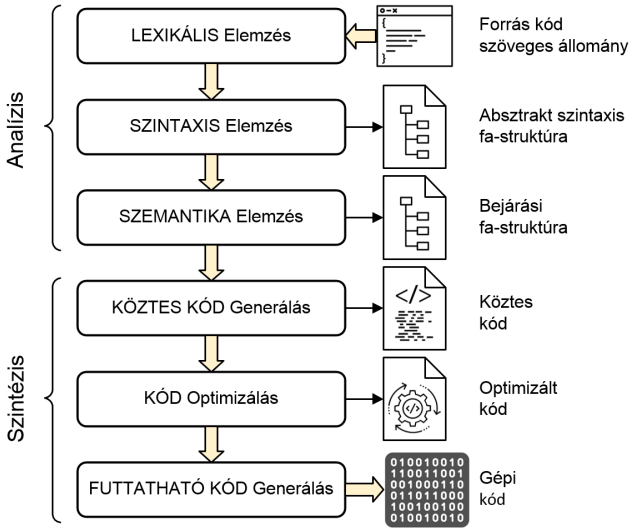
8.45. ábra. *Assemblerben írt programpélda*

A szimbolikus szavak sorozatait tartalmazó szöveges állományt a fordítóprogram feldolgozza, azaz ellenőrzi a helyességüket, értelmezi a sorrendiségüket, majd ezek után megkeresi az egyes szimbolikus szavakhoz és jelekhez tartozó hexadecimális számokat és behelyettesíti őket, létrehozva egy gépkódú állományt.

A programozási nyelvek „helyesírási” és „mondattani” szempontból nagyon kötöttek, egyetlen szintaktikai vagy sorrendiségi hiba sem lehet bennük, ellenkező esetben a fordítóprogram nem tudja lefordítani a bemeneti, szöveges állományt.

A 8.46. ábra egy fordítóprogram működését mutatja a köztes, származékállományok felsorolásával. A program két nagy feldolgozási fázis során jut el a kimeneti, gépi kódú állomány létrehozásáig. Az analizáló fázis a bemeneti szöveges állomány lexikális elemzésével kezdi a feldolgozást, meghatározva a bemeneti állományban fellelhető szimbólumok szövegét és típusát. A lexikális elemzés karaktersorozatokat olvas be, és ezekben keresi az előre meghatározott kulcsszavakat, szimbólumneveket. Például a „LOAD from 0x09h” sorban azonosítja a „LOAD”, a „from” szavakat, valamint a 0xNNh szót, ahol NN a címérték két karaktere. Amennyiben „LAOD form 0h09x” sort olvasna be, nem tudná azonosítani a kulcsszavakat és hibaüzenetet generálna. A köztes eredményt a szintaktikai elemző rész veszi át, és megvizsgálja a sorokban található szimbólumok helyes sorrendjét. Amennyiben a sor a „From 0x09h LOAD” sorrendben tartalmazná a szimbólumokat,

bár ezek lexikális szempontból mind helyesek, lehetséges sorrendiségüket tekintve helytelen kombinációt mutatnak. A szintaxiselemző egy fastruktúrában rögzíti a kulcsszavakhoz, szimbólumokhoz tartozó további jelöléseket (például „LOAD” alatt a „from” szó, majd az alatt a 0x09h címérték).

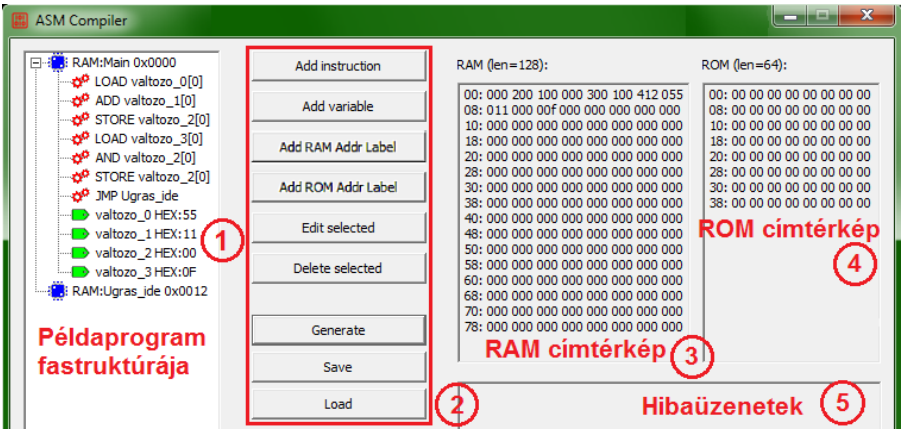


8.46. ábra. *Fordítóprogram működésének vázlatos ábrázolása a bemeneti és kimeneti állományok megjelölésével*

A szemantikai elemzés a változók értékét, címértékét, az ugráscímek helyességét vizsgálja. Minél magasabb szintű nyelven írt bemeneti állományról van szó, annál több részletet kell itt megvizsgálni (változók deklarációja vagy annak hiánya, elérhetősége, operátorok és operandusaik közötti típuskompatibilitás, eljárások címe és kiterjedése, memóriaterület foglaltsága stb.). Az elemzés eredménye egy bejárási fastruktúrában kerül a következő feldolgozási fázis, a szintetizálás első szakaszába, ahol egy köztes kódállomány jön létre. Ezt a kódállományt különböző programozástechnikai megfontolások alapján optimizálni lehet (például kihasználva azt, hogy egy aritmetikai művelet eredménye a munkaregiszterben marad, több olyan utasítás is egymás után rendezhető, amelyek a munkaregiszterben tárolt értéket annak megváltoztatása nélkül használják, kiiktatva ezzel egy pár betöltő utasítást stb.).

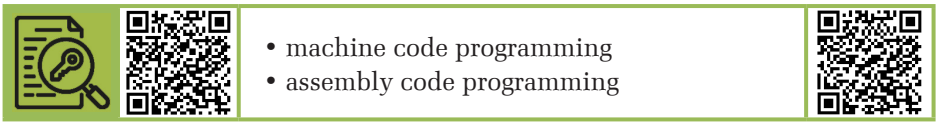
Az optimizálási szakasz után a fordítóprogram behelyettesíti a szimbólumokat, változóneveket, címértékeket azok hexadecimális formában ábrázolt értékeivel, majd ezek sorozatát, a kívánt bináris formában, egy kimeneti állományba írja, létrehozva a processzor által futtatható kódot.

A fordítóprogram analízist végző része kiváltható egy sajátos megközelítési módszerrel: a szövegbeviteli rész grafikus szerkesztői felületre cserélhető (8.47. ábra). Ebben az esetben az utasításokat, változókat, címeket egy menürendszer segítségével, legördülő listák alkalmazásával visszük be. A legördülő listák (2) tartalma kötött, utólagos elemzést nem igényel, így elmarad a lexikális, szintaktikai és szemantikai ellenőrzés és strukturálás. A bevitt utasítások, változók helye a memóriacímtérben attól függ, hogy grafikusan hol helyezzük el őket a programot megtestesítő fastruktúrában (1).



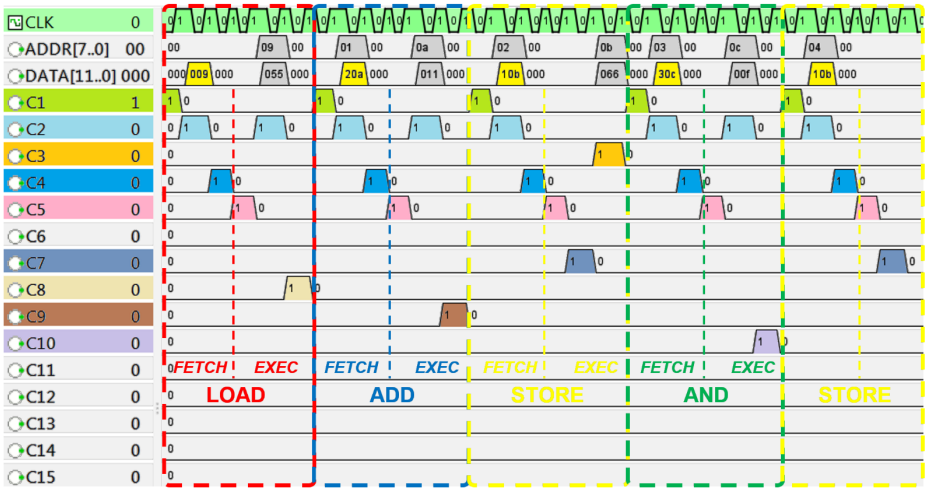
8.47. ábra. *Grafikus beviteli felülettel rendelkező assembler programfejlesztő környezet és fordítóprogram*

A programszerkesztés befejezését követően a fordítás rögtön a második, szintézis fázisba léphet. Ez három szakaszban zajlik, először a kialakított fastruktúra bejárásával kialakul egy köztes kód, amely minden programban szereplő változót, függvényhívást, ugráscímet képviselő címkét ellát a megfelelően kiszámított címértékkel. A második szakaszban történik a dinamikus bemenetekkel rendelkező utasítások (**LOAD**, **STORE**, **CALL** stb.) feltöltése a kiszámított címértékekkel, majd harmadikként következik a gépi kódok sorozatának kimeneti, memória-tükör állományokban (RAM – 3, ROM – 4) való rögzítése. Ezeket az állományokat töltjük be a szimulációs környezet memóriamoduljaiba. A könyvben bemutatott számítógépszimulációs-modellre írt programok egy ilyen szerkesztő-fordító környezetben készültek.



8.8. Az utasítás-végrehajtás szakaszai és időDiagramja

A megtervezett mikrokódos vezérlőegységgel rendelkező processzor szimulációs modelljén egy kísérleti programot futtatva elemezhetjük az egyes utasítások végrehajtási menetének időDiagramját. A 8.48. ábra egy tömbösített időDiagramot jelenít meg, felső sorában az órajellel. A processzorban minden jelzés az órajel (CLK) valamelyik fel- vagy lefutó élével szinkronban változik, jelenik meg vagy tűnik el (alapállapotba kerül), ezért az elemzést ennek függvényében végezzük. Az órajel alatt a címsín (ADDR**[7..0]**), majd adatsín (DATA**[11..0]**) pillanatnyi értékei sorakoznak. Ez után következnek a vezérlőjelek (C**1-C15**) állapotait rögzítő sorok.



8.48. ábra. *Utasítás-végrehajtási ciklusok a vezérlőjelek megfigyelésével LOAD, ADD, STORE és AND esetében*

Az előző fejezetekből tudjuk, hogy az utasítás-végrehajtás 6 órajelciklus alatt történik, az utasítás-előkészítő (FETCH), majd -végrehajtó (EXEC) szakaszok bejárásával. Az időDiagram adott részeinek függőleges vonali összeolvasásával vizsgáljuk az utasítás-végrehajtás menetét. Piros szaggatott vonalas keretben látható a „LOAD from 0x09h” utasítás. Az utasítás-végrehajtás a 0x00h címről kezdődik, ezt a címsínen látható érték (**00**) is tükrözi. Az utasítás-előkészítő szakaszban a C_1, C_2 és $C_{(0)4}$ jelzések (AR←PC, READ MEMORY, PC←PC+**1** és IR←DR (OP)) követik egymást. Az adatsín értéke kezdetben **000**, ez az első C_2 vezérlőjel megjelenése (felfutó éle) után **009** értékre változik, ez lesz a végrehajtandó utasítás kódja és a betöltendő változó címe (0x009h). Az előkészítő szakasz végén az utasítás kódja az utasításregiszterbe (IR), az operandus címe az adatregiszterbe (DR) kerül. A végrehajtási szakaszban a C_5-ös vezérlőjellel (AR←DR (ADDR)) szinkronban az operanduscím a címregiszterbe (AR) kerül. A címsínen a második C_2-es jelzés felfutó élére megjelenik a **09** (0x09h)

érték, majd a c_2-es jelzés lefutó élével szinkronban az adatsínen megjelenik a 0x09h címről kiolvasott **055** (0x055h) érték. Végül a c_8 jelzés (ACCU←DR) hatására a beolvasott adat a munkaregiszterbe kerül.

Hasonló folyamat játszódik le a kék szaggatott vonalas kerettel kiemelt ADD utasítás végrehajtása során is. Az utasítás-előkészítő szakaszban a c_1, c_2 és $c_{(0)4}$ jelzések (AR←PC, READ MEMORY, PC←PC+1 és IR←DR (OP)) követik egymást. Az adatsín értéke kezdetben **000**, ez az első c_2 vezérlőjel megjelenése (felfutó éle) után **20a** értékre változik, ez lesz a végrehajtandó utasítás kódja és a betöltendő változó címe (0x20Ah). Az előkészítő szakasz végén az utasítás kódja az utasításregiszterbe (IR), az operandus címe az adatregiszterbe (DR) kerül. A végrehajtási szakaszban a c_5-ös vezérlőjellel (AR←DR (ADDR)) szinkronban az operanduscím a címregiszterbe (AR) kerül. A címsínen a második c_2-es jelzés felfutó élére megjelenik a **0a** érték, majd a c_2-es jelzés lefutó élével szinkronban az adatsínen megjelenik a 0x0ah címről kiolvasott **011** (0x011h) érték. Végül a c_9 jelzés (ACCU←ACCU+DR) hatására a beolvasott adat (0x11h) a munkaregiszterben tárolt értékhez (0x55h) hozzáadódik, és az eredmény (0x66h) a munkaregiszterbe kerül.

A sárga szaggatott vonalas kerettel kiemelt STORE utasítás előkészítő szakasza is az eddig vázoltak szerint zajlik. Az előkészítő szakasz végén az utasítás kódja az utasításregiszterbe (IR), a tárolásra kerülő érték címe (0x0bh) az adatregiszterbe (DR) kerül. Az utasítás végrehajtási szakaszában a c_5-ös vezérlőjellel (AR←DR (ADDR)) szinkronban a tárolási cím a címregiszterbe (AR) kerül. Ezután a c_7-es vezérlőjel (DR←ACCU) hatására a munkaregiszter (ACCU) tartalma átíródik az adatregiszterbe (DR). Majd a c_3-as jelzés (WRITE MEMORY) felfutó élével szinkronban a címsínen megjelenik a **0b** (0x0bh) érték, az adatsínen pedig a 0x0bh címre beírandó **066** (0x066h) érték. Végül a hatodik órajel felfutó élével szinkronban megtörténik a memóriaírás, az összeadás eredményét eltároltuk.

A leírt értelmezési modell alapján próbáljuk meg követni az idődiagramon zöld szaggatott vonalas kerettel kiemelt AND, majd ismét egy sárga szaggatott vonalas kerettel kiemelt STORE utasítás végrehajtásának menetét.

- instruction execution cycle
- machine cycles timing diagram

9. Számítógéparchitektúra-tervezés

A processzor modelljének létrehozása, működésének ellenőrzése, valamint az utasítás-végrehajtás folyamatának elemzése után elkezdhetjük a processzor köré épülő számítógép-architektúra megtervezését. Vázoljuk a rendszer címtérképét, létrehozzuk a külső cím-, adat- és vezérlősíneket, kidolgozzuk a címdekódoló áramköröket, majd ezeken keresztül különböző alkatrészeket (memória, kijelző, billentyűzet, közvetlen memóriahozzáférés-vezérlő) csatlakoztatunk a sínekre. Alapvetés, hogy a számítástechnikai rendszer címek alapján azonosítja a csatolt elemeket, azok belső regisztereit. Általános elvként egy számítástechnikai rendszerben minden kezdeményezett adatátvitel a címsínen érvényes címérték megjelenítésével kezdődik, ezt követi az adatsín és a megfelelő, külső vezérlőjelek beállítása.

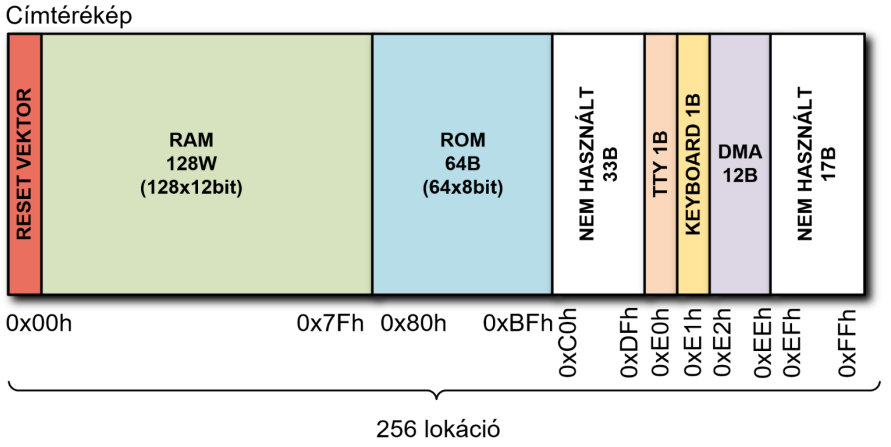
9.1. Címtérkép, címkiosztás, címdekódolás

A címtérkép felvázolása az architektúratervezés alapját képezi. Ebből kiindulva határozzuk meg a későbbiekben a címdekódoló áramkörök típusát és a dekódolás természetét (teljes vagy részleges dekódolás), ebből vezetjük le az egyes illesztett alkatrészek viszonyát a processzorral és a teljes rendszerrel összefüggésben. A címtérkép (9.1. ábra) a kialakítani kívánt rendszer sajátosságai szerint alakulhat, és a címterület nagysága a címsín szélességének függvénye. Tartalmaznia kell az egyes címterületek kezdeti és végső címét, valamint a szabadon maradt címterületek jegyzékét. Az egyes területek címértékei befoglaló (inkluzív) jellegűek, azaz az adott alkatrész első és utolsó elérhető címértékeit jelölik. A szabad területek jegyzéke azokat a címeket jelenti, amelyeket nem foglal le egyetlen alkatrész sem. A szabad címterület a rendszer későbbi bővíthetőségének meghatározó feltétele. Minél több szabad címterület áll rendelkezésre, annál alakíthatóbb (flexibilisebb) a számítástechnikai rendszer.

A címsín szélessége nem minden esetben azonos a processzor külső címsínjének szélességével, ez multiplexelési technikákkal a többszörösére bővíthető. Az eddig végigvezetett tervezési folyamat során létrehozott processzormodell 8 bites külső címsínnel rendelkezik, ennek megfelelően a maximális kialakítható címérték a 0xFFh. A kereskedelemben kapható számítástechnikai rendszerek címsínszélessége és a kialakítható címérték nagysága ennek többezerszerese is lehet.

A tervezendő architektúra címtérképének első szakaszát a RAM memória címterülete adja, és 128 szónyi (word – szó, a 8 többszöröseitől különböző számú bitet számláló adatmező-kiterjedés) helyet foglal le a 0x00h kezdőcímtől a 0x7Fh címig. Fontos megjegyezni, hogy ez a címterület magába foglalja a **RESET VEKTOR** elnevezésű,

kiemelt fontosságú címértéket (0x00h) is. A RESET VEKTOR azon a címértéken tartalmaz utasítást, amelyet betáplálás (fizikailag létező rendszer), szimulációindítás vagy alapállapotba hozás után a processzor elsőként beolvas és végrehajt. Ez rendszerenként változó lehet, nem megkötés, hogy a processzor a 0x00h címértéket használja. A RAM-terület a rendszerben alapvetően azt a központimemória-területet jelöli, ahol a végrehajtandó utasítások és adatok foglalnak helyet.



9.1. ábra. *Számítástechnikai rendszer címtérképének vázlata*

A címtérkép második szakaszát a ROM memóriának fenntartott terület jelenti, és 64 bájtnyi helyet foglal le a 0x80h címtől a 0xBF címig. A ROM-címterület alapvetően az állandó értékeknek (konstansoknak) van fenntartva, például a kijelzőn megjelenített szövegek karakterkódjainak tárolására használható.

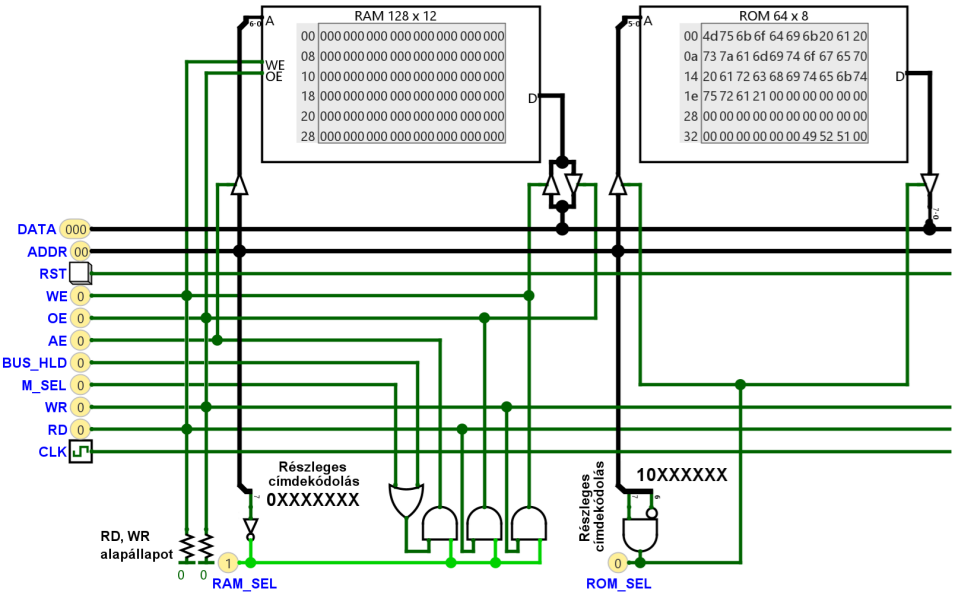
A harmadik szakasz 38 bájtnyi szabad területet jelöl a 0xC0h címtől a 0xDF címig. Ide a későbbiekben a számítástechnikai rendszert kiegészítő alkatrészeket lehet befűzni.

A negyedik és ötödik szakasz egy-egy címértéket fed le, a 0xE0h címen található a rendszerhez kapcsolt kijelző (TTY – Tele Type Terminal) adatregisztere, a 0xE1h címen érhető el a billentyűzetperiféria kimeneti regisztere.

A hatodik, 12 bájtnyi szakaszt a közvetlenmemóriahozzáférés-vezérlő (DMAC – Direct Memory Access Controller) belső regiszterei foglalják le a 0xE2h címtől a 0xEEh címig terjedően. A 0xEFh címtől a 0xFFh címig fennmaradó 17 bájtnyi címterület a számítástechnikai rendszer későbbi bővítésekor szabadon használható.

A rendszerbe kapcsolt alkatrészek általában logikai kapukból felépített összehasonlító áramkörök, bináris komparátorok segítségével azonosítják a címsínen megjelenő, nekik szóló címértékeket. Ha a megjelenő címérték eltér az adott alkatrész által felismerhető értéktől, az alkatrész nem vesz részt az adatsínen zajló adatátvitelben (kommunikációban).

A címdekódolás lehet részleges vagy teljes. Részleges címdekódolás (9.2. ábra) ott valósulhat meg, ahol a címértéknek elegendő egyetlen vagy több jellegzetes bit helyértékét vizsgálni, és ebből már megállapítható, hogy az adott címérték az elérni kívánt alkatrésznek szól. Példánkban a RAM és ROM memóriamodulok esetét vizsgáljuk, mindkettő részleges címdekódolást használ a nekik szóló címértékek előzetes szűrésére.

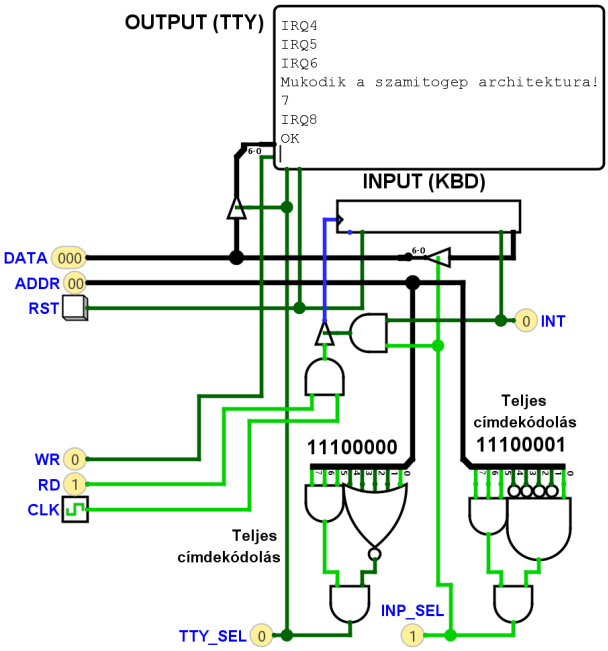


9.2. ábra. *RAM és ROM memóriamodulok rákapcsolása a számítógép cím-, adat- és vezérlőjelsínjeire. Példa részleges címdekódolásra*

A RAM memória esetében a szűrés a címsínen megjelenő érték legfelsőbb bithelyértékét vizsgálja. Ha ez 0, akkor a címsínen megjelenített érték a RAM memória címterületét jelöli. Ilyenkor a RAM memória cím- és adatvezetékei rákapcsolódhatnak a számítástechnikai rendszer cím- és adatsínjeire. A címtérképből is következik, hogy a 0x7Fh (**0**1111111) és ennél kisebb értékek a RAM címterületének részei. A címértéktartomány szűréséhez ebben az esetben egyetlen, a címsín legnagyobb helyértéket hordozó vonalára kapcsolt tagadó kapura van szükség (9.2. ábra bal oldali része). Ennek kimenete engedélyezi a **RD** és **WR** jelzések, valamint a címsín és adatsín vonalainak továbbhaladását a RAM-modul felé.

Hasonló megoldás látható a ROM-modul esetében is, itt a címsínen megjelenő érték két legfelső bithelyértékének vizsgálatát vonja egybe az egy tagadott bemenettel rendelkező **ÉS** kapu (9.2. ábra jobb oldali része). Ennek kimenete engedélyezi a címsín és adatsín vonalainak továbbhaladását a ROM modul felé. A címtérképből

is következik, hogy a 0x80h-tól (**10**000000) és ennél nagyobb értékek egészen a 0xBFh-ig (**10**111111) a ROM címterületének részei.



9.3. ábra. *Kijelző- és bemeneti egység (billentyűzet) rákapcsolása a számítógép cím-, adat- és vezérlőjelsínjeire. Példa teljes címdekódolásra*

Teljes címdekódolást (9.3. ábra) használunk ott, ahol egyetlen címértékre kell szűrni, például a kijelző- és a billentyűzetperifériák esetében.

A **kijelző** a 0xE0h (**111**00000) címen fogadja az adatokat (karakterkódokat). A címdekódolás egy hárombemenetű **és** meg egy ötbemenetű **tagadott vagy** (NOR) kapu, valamint ennek a két kapunak a kimenetét összefogó kétbemenetű **és** kapu felhasználásával valósítható meg. A **billentyűzet** kimeneti regiszterének címét, az 0xE1h (**111**00001) értéket egy hárombemenetű meg egy ötbemenetű **és** kapuból, valamint ennek a két kapunak a kimenetét összefogó kétbemenetű **és** kapuból kialakított áramkör figyeli. Az ötbemenetű **és** kapu négy bemenete, a cím nullás értékeinek megfelelő helyeken, tagadott. A két periféria csak akkor elérhető, ha az általuk dekódolt címérték jelenik meg a címsínen.

- memory map of a computer
- memory address decoding

9.2. Közvetlen memóriahozzáférés-vezérlő (DMAC) tervezése

Egy számítógéprendszer számítási teljesítményét nagymértékben befolyásolja a rendszeren belüli adatmozgatás sebessége. Minél gyorsabb egy perifériából vagy háttértárolóból az elérni kívánt adatoknak a központi memóriába másolása vagy onnan való kiírása, annál több utasítást tud adott idő alatt a processzor ezeken az adatokon elvégezni.

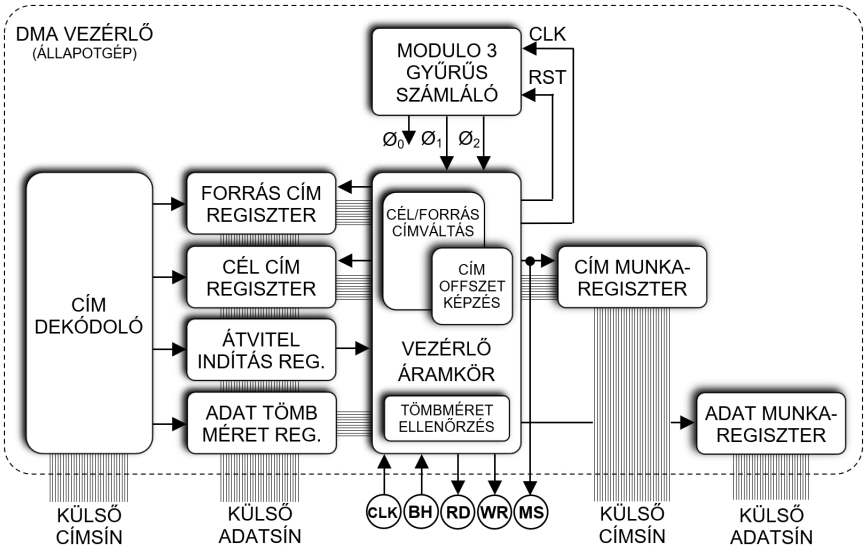
Ha az adatmozgatást csak a processzor végzi az erre alkalmas utasítások végrehajtásával, minden adatnak a forrásból a célba való eljuttatásához (átírásához) két utasítást kell végrehajtania. A szimulációs környezetben létrehozott processzornak 6 órajel szükséges egy utasítás végrehajtásához, így egy adat átmásolásához 12 órajelre lenne szükség. Kiszámolható, hogy ha az órajel például 100 MHz frekvenciájú, akkor periódusideje 10 ns (nanoszekundum), egy adatátviteli ciklus időtartama ennek 12-szerese, azaz 120 ns. Egy 32 bájtos tömb átvitele 3,84 μs-ba (mikroszekundum) telne. Nem elhanyagolandó részlet, hogy az adatmásolási ciklus alatt a processzor nem tud más programrészletet végrehajtani, csak kizárólag a másolási ciklus utasításait.

Az adatátvitel felgyorsítására egy olyan perifériát hozhatunk létre, amely átveszi az adatmozgatás feladatát a processzortól, és az átvitelt akkor bonyolítja le, amikor a processzor nem használja a számítógéprendszer cím-, adat- és vezérlősínjeit, és közben egy utasítást hajt végre. Ez a legtöbb utasítás (**LOAD**, **STORE**, **ADD**, **AND** stb.) végrehajtásában legalább egy átvitelt, de bizonyos utasítások (**JMP**, **JZ** stb.) esetében két átvitelt is jelenthet.

A közvetlenmemóriahozzáférés-vezérlő (DMAC) áramkört úgy képzelhetjük el, mint egy olyan automatát, amely a **LOAD** és **STORE** utasítások hatásmechanizmusát kombinációs és sorrendi alapáramkörökből kialakított, a rendszer órajelével szinkronban működő állapotgép segítségével valósítja meg. Működési elvét tekintve egy olyan áramkörről van szó, amelynek megadhatjuk az átvitelre szánt adattömbnagyságát (bájtok, szavak számát), a forrás és a cél címét, majd egy jelzéssel utasítjuk az adatátvitel megkezdésére. Az áramkör önműködően, a processzor közbelépése nélkül elkezdi az adatátvitelt, amelyet addig folytat, amíg a meghatározott számú adat átvitelét be nem fejezi. Fontos kiemelni, hogy adatátvitelt csak akkor bonyolít le, amikor a processzor éppen nem használja a cím-, adat- és vezérlősín jeleit.

A közvetlenmemóriahozzáférés-vezérlő felépítését tekintve több részegységből áll, amelyek a 9.4. ábra szerint kapcsolódnak egymáshoz. Az ábrán a részegységek funkcióik szerint csoportosítva a központban lévő vezérlő áramkör két oldalán helyezkednek el. Az ábra érthetőségének fokozása érdekében mindkét oldalon megjelenik a külső cím- és adatsín, a valóságban az áramkör csak egyetlen pontban kapcsolódik ezekhez a sínekhez. Bal oldalra kerültek a DMAC áramkör működési paramétereit rögzítő beállító vagy konfigurációs regiszterek,

jobb oldalon a tulajdonképpeni adatátvitelt lebonyolító cím- és adat-munkaregiszterek láthatók.



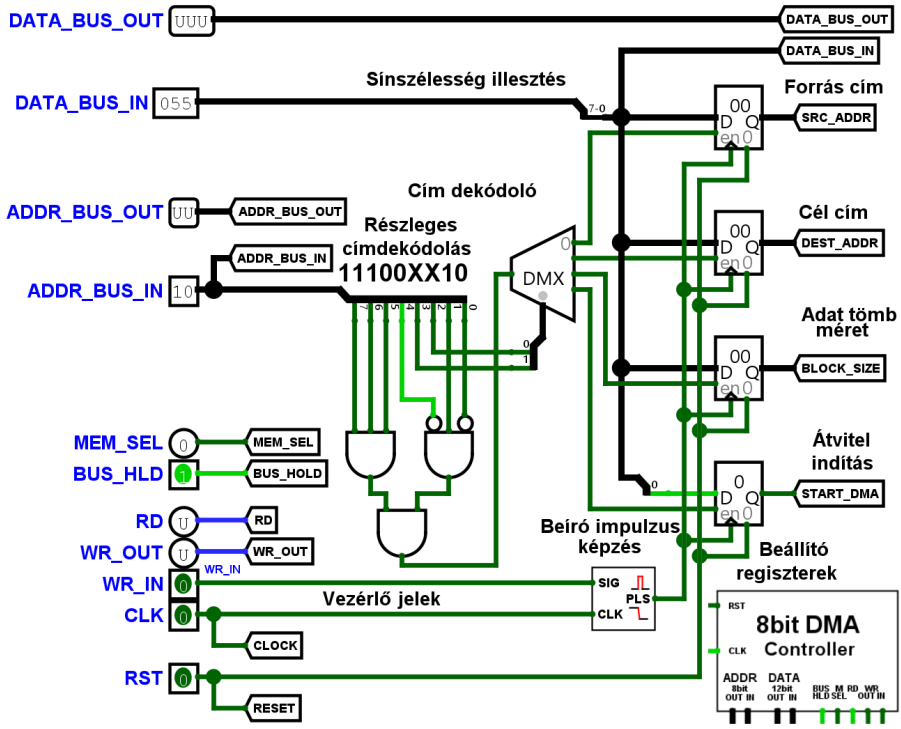
9.4. ábra. *Közvetlenmemóriahozzáférés- (DMAC) vezérlő tömbvázlata*

A címdekódoló a számítógéprendszer címsínjén megjelenő értékek függvényében engedélyezi a DMAC valamelyik belső regiszterének az adatsínre való csatlakozását. A **forrás cím** regiszterben adjuk meg az átmásolni kívánt adattömb kezdőcímét, a **cél cím** regiszterben pedig azt a címet, ahova át akarjuk másolni az adattömböt. Az **adattömbméret** regiszterbe kerül az átvinni kívánt bájtok vagy szavak száma. Miután ezeket a regisztereket feltöltöttük a megfelelő értékekkel, az **átvitelindítás** regiszterbe írt értékkel, megindíthatjuk a folyamatot.

A DMAC áramkör működés közben folyamatosan figyeli a cím-, adat- és vezérlősínek processzor általi foglaltságát (**bh** – **bus hold** jelzés), és csak akkor kezdeményez saját adatátvitelt, ha a jelzés szabad utat (0) mutat.

Ilyenkor egy belső állapotgép a beállító regiszterekben lévő értékekből létrehozza az adatátvitelhez szükséges címértéket (**cím offszet képzés**) és vezérlőjeleket. Először beolvas a forráscímről (**rd**), majd írást kezdeményez a célcímre (**wr**). A műveletek alatt saját memóriakiválasztó jelzést (**ms** – **memory select**) hoz létre, amely a processzor sínfoglaló jelzésének (**bh**) megfelelője. Ezt a lépéssorozatot addig folytatja, amíg a belső lépésszámlálója eléri az adattömb méretét (**tömbméret-ellenőrzés**). Ez a folyamat értelemszerűen minden esetben megszakad, amikor a processzor használja a cím-, adat- és vezérlősíneket (**bh**=1), majd folytatódik, amikor „elengedi” (**bh**=0) ezeket.

A 9.5. ábra a DMA-vezérlőegység regisztereinek kapcsolási rajzát mutatja. A négy darab, 8 bites regiszter bemenete az adatsínre kapcsolódik. Az adatsínre való kapcsolódásuk engedélyezése a címdekódoló (részleges címdekódolás) kimeneti jelzésétől és a csatolt demultiplexer kimeneteinek állapotától függ. Ezeken a kimeneteken csak akkor jelenik meg 1-es logikai érték, ha a dekódolt cím helyes. A címdekódolás három ÉS kapuból kialakított kombinációs áramkörrel valósítható meg. Két hárombemenetű ÉS kapu a cím állandó bitértékeit szűri, kimeneteiket egy ÉS kapu fogja össze. A demultiplexer kiválasztó bemenetei a cím változó bitértékeit fogadják, adatbemenete az ÉS kapu kimenetére csatlakozik. A regiszterek írása a WR vezérlőjellel történik (STORE utasítás) az órajel (CLK) lefutó élével szinkronban (**Beíró impulzus képzés**).

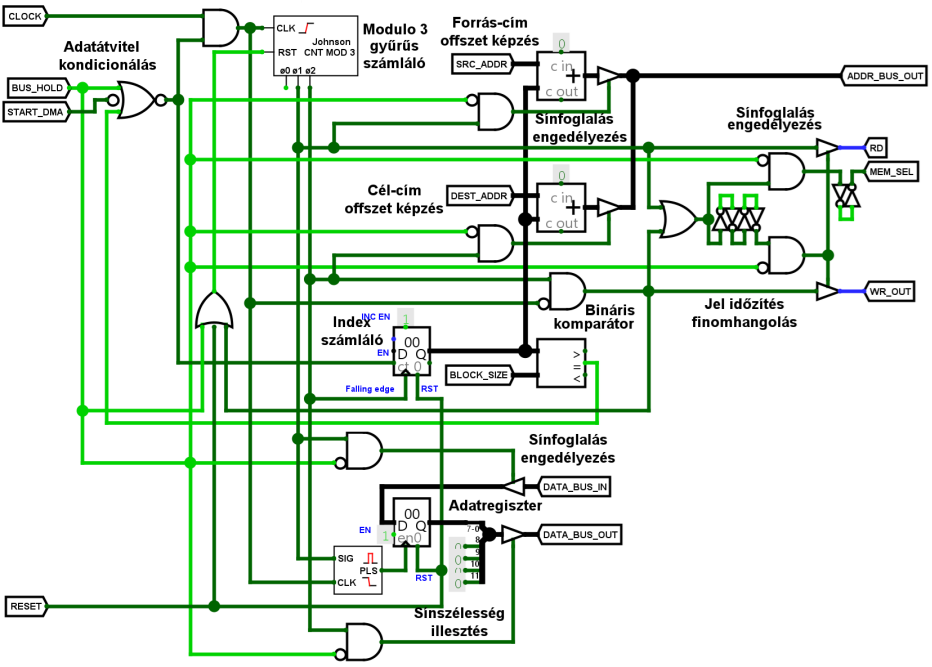


9.5. ábra. *DMA-vezérlő egység regisztereinek kapcsolási rajza és szimbóluma*

A kapcsolásban különbséget teszünk a DMA-vezérlő által fogadott WR jelzés (WR_IN) és a kiküldött WR jelzés (WR_OUT) között, viszont a számítógéprendszer vezérlősínjének WR jelzésére mindkettő rácsatlakozik. A DMA-vezérlő belső adatsínje 8 bit szélességű. Következésképpen ebben a kiépítésben csak adatok átvitelére alkalmas. A WR vonalhoz hasonlóan a kapcsolásban különbséget teszünk a

dma-vezérlő által fogadott cím- és adatértéket hordozó sínek (ADDR_BUS_IN, DATA_BUS_IN) és a kiküldött adatértékeket hordozó sínek (ADDR_BUS_OUT, DATA_BUS_OUT) között, viszont a számítógéprendszer cím- és adatsínjeire mindkét irányt jelölő sín rácsatlakozik.

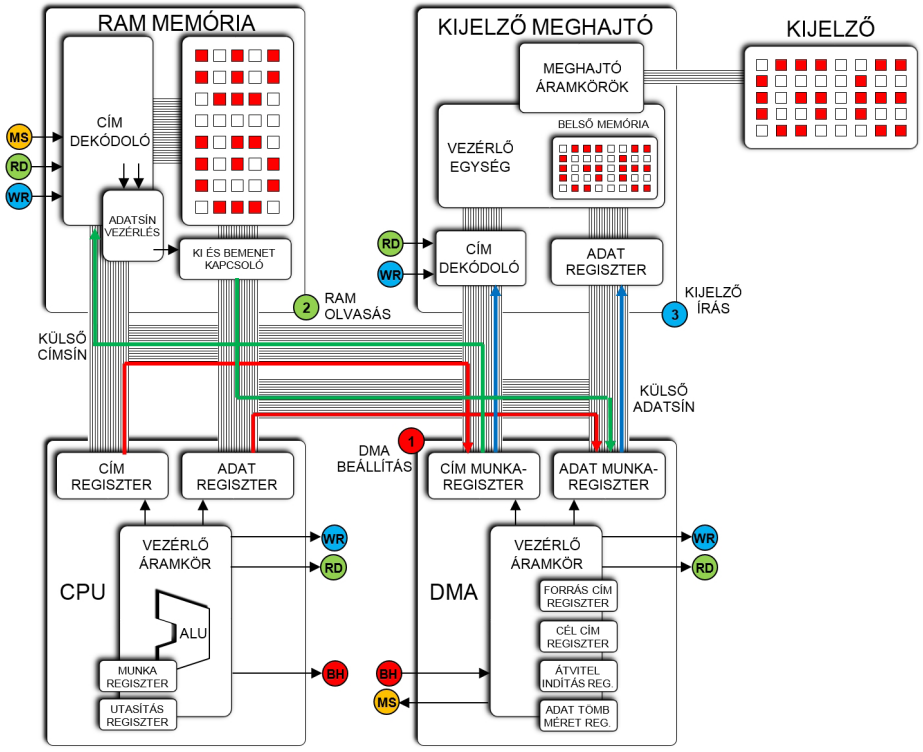
A 9.6. ábra a DMA-vezérlő állapotgépének kapcsolási rajzát mutatja. Az állapotgép mindaddig alapállapotban marad, amíg BUS_HOLD jel 1-es, a START_DMA jel 0-ás, valamint a tömbméret-ellenőrző bináris komparátor kimenete 1-es. Ilyenkor a gyűrűs számláló $ø_0$ kimenete aktív, de ez nem kapcsolódik más áramkörök bemeneteire. Amikor a BUS_HOLD jel 0-ás és a START_DMA jel 1-es lesz, és a tömbméret-ellenőrző bináris komparátor kimenete 0-ás, az állapotgép két további lépésben lebonyolítja egy adatbájt átvitelét. A $ø_1$ fázisban történik a forráscímofszet- (eltolás) képzés, a RD jelzés aktiválása (1) és az adat beolvasása az adatregiszterbe. A $ø_2$ fázisban történik a célcímofszetképzés, az adatregiszter rákapcsolása a külső adatsínre, végül a WR jelzés aktiválása (1) és az indexszámláló növelése. Ekkor az előzőlegesen beolvasott adatérték beíródik a megcímzett lokációba.



9.6. ábra. *DMA-vezérlőegység állapotgépe a címképző áramkörökkel, indexszámlálóval, komparátorral és adatregiszterrel*

Ez a folyamat addig ismétlődik, amíg a bináris komparátor nem észlel egyezést a tömbméretregiszter (BLOCK SIZE) és az indexszámláló kimeneti értékei között, vagy

a processzor meg nem szakítja az adatátvitelt a BUS_HOLD jel aktiválásával. Mindkét esetben az állapotgép visszatér a kiindulási állapotába. Amennyiben a BUS_HOLD jel inaktív lesz (0), és a bináris komparátor kimenete nem aktív, azt jelenti, hogy még van átvitelre váró adatérték. Az adatátviteli folyamat újra elkezdődik és mindaddig tart, amíg a bináris komparátor egyezést észlel a tömbméretregiszter (BLOCK SIZE) és az indexszámláló kimeneti értékei között, ekkor azonnal alapállapotba hozza az állapotgépet. Itt a DMA-átvitel befejeződik.



9.7. ábra. *Közvetlenmemória-hozzáféréses (DMA) adatátvitel lebonyolításának tömbvázlata memória és periféria között*

A közvetlenmemória-hozzáférés adatátvitel folyamata legjobban egy gyakorlati példán keresztül érthető meg. Tegyük fel, hogy négy fő alkatrészből álló számítógéprendszert működtetünk. A processzor, a RAM memória, a kijelzőmeghajtó és a DMA-vezérlő a cím- és adatsíneken kapcsolódik egymáshoz. A processzor egy olyan programot hajt végre, amely grafikus képeket hoz létre. A képeknek megfelelő adattömböket a RAM memóriában hozza létre egy grafikai funkciókat ellátó algoritmus segítségével. Amikor egy-egy adattömb elkészül, ezt a lehető

leggyorsabban el kell juttatni a kijelzőmeghajtó belső memóriájába. Ennek alapján a meghajtó egy kijelzőn megjeleníti a beírt képet. Figyeljük meg a 9.7. ábra színes korongjait és a bennük szereplő jelzések nevét, valamint a cím- és adatsíneken haladó nyilazott vonalak irányát. Az azonos színű korongok és vonalak kiemelik a közvetlenmemória-hozzáférés adatátvitel egyes szakaszaihoz tartozó jelzéseket.

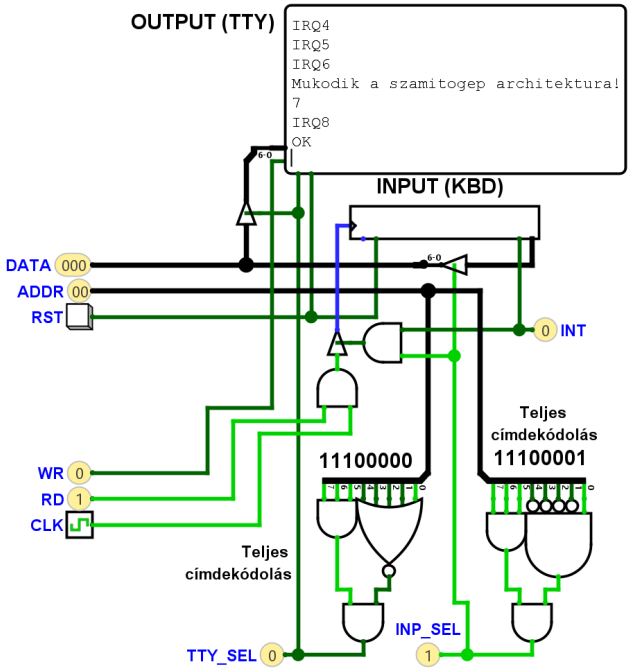
Az első szakaszban (piros) a DMA-vezérlő beállítása történik. A processzor által végrehajtott programrészlet feltölti a DMA-vezérlő beállító regisztereit. A DMA-vezérlő figyeli a sínek foglaltságát tükröző **BH** jelzést. Amint a sínek felszabadulnak, aktiválja a saját **MS** jelzését, és az átvitel második szakaszában (zöld) megtörténik a forráscímen tárolt adat DMA-vezérlő általi olvasása. A DMA-vezérlő ekkor címet vált, és az átvitel harmadik szakaszában (kék) megtörténik a célcímre való adatírás. A kijelzőmeghajtó rendre megkapja a kép kijelzéséhez szükséges adatokat (a második és harmadik szakasz véges ismétlődése során), majd az adatátvitel befejeztével egy hozzá csatlakoztatott kijelzőn megjeleníti a képpontokat. Nagyon fontos kiemelni, hogy az adatátvitel alatt a processzor újabb képtartalom kidolgozását végezheti el, és előkészíthet egy következő közvetlenmemória-hozzáférés adatátviteli ciklust.

- direct memory access
- DMA controller

9.3. Ki- és bemeneti perifériák rendszerbe kapcsolása

Az emberi felhasználó számára a számítógéprendszer legfontosabb alkotóelemeit azok az áramkörök képezik, amelyek a számítások, algoritmusok bevitelét (programozását), illetve az ezek végrehajtásából fakadó eredményeket az emberi érzékszervek számára értelmezhetően megjelenítik. Már a számítástechnika fejlődésének kezdeti szakaszaiban is sikerrel alkalmazták az írott szöveget rögzíteni (adatbevitel) és megjeleníteni (kijelzés) képes írógépek automatizált változatait, később a katódsugárcsöves kijelzőket a képinformációk közlésére, csengőket, sípokat, majd hangszórókat a hanghatások képzésére.

A szimulációs környezetben létrehozott számítógéprendszer két ilyen, a szimulációs programkörnyezetbe beépített perifériával rendelkezik, az egyik egy szöveges információt megjeleníteni képes kijelzővezérlő, nevezzük kimeneti perifériának. A másik egy billentyűzet segítségével bevitt karakterek rögzítésére alkalmas áramkör, nevezzük bemeneti perifériának. Ezt a két perifériát az előzőekben leírt módon csatlakoztatjuk a számítógéprendszer cím-, adat- és vezérlőjelsínjeire (9.8. ábra).

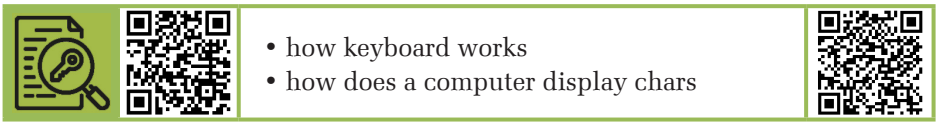


9.8. ábra. *Kijelző és bemeneti egység (billentyűzet) rákapcsolása a számítógép cím-, adat- és vezérlőjelsínjeire. Példa teljes címdekódolásra*

Rögzítsük, hogy a kijelzőn csak olyan karakterek és ezekből összeálló szövegrészletek jelennek meg, amelyeket valamilyen előre megírt programrészlet létre tud hozni. Ugyanakkor a karakterek bevitele nem jelenti minden esetben a számítógéprendszer közvetlen utasításokkal való ellátását, hanem csak adatot szolgáltat valamilyen feladat elvégzéséhez.

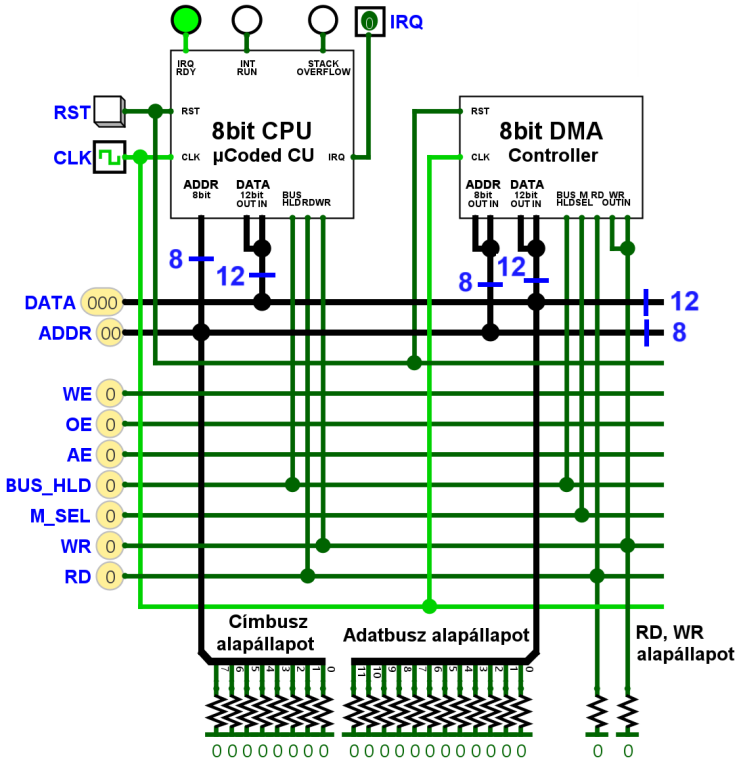
Kísérleti rendszerünkben a kijelzőre való írás egy **STORE 0xE0h** utasítással valósítható meg. Nyilván a **STORE** utasítás előtt az adatot a processzor munkaregiszterébe kell tölteni (**LOAD**), vagy valamilyen, előzőleg végrehajtott aritmetikai, logikai művelet munkaregiszterben található eredményét tudjuk megjeleníteni. A kijelző periféria ASCII kódokat kezel (az ASCII kódról bővebben lásd a MELLÉKLET második alfejezetét, 11.2.).

A billentyűzet vagy beviteli mező egy beállítható mélységű bufferbe gyűjti a lenyomott billentyűk karakterkódjait. Ha a buffer nem üres, a bemeneti periféria fogadni tudja a rendszerórajelet, amelyet a **RD** jelzéssel **ÉS** összefüggésben az órajel bemenetére vezetve megtörténhet a bufferbe legelsőnek beírt karakter kód kiolvasása egy **LOAD 0xE1h** utasítással. A kiolvasás mindaddig folytatható, amíg a buffer ki nem ürül.



9.4. Összetett számítógép-architektúra kialakítása

Ezen a ponton a végére értünk a tervezési folyamatoknak, a kialakított alkatrészekből összeállítjuk a didaktikai példának szánt számítógép-architektúrát, amelynek működtetéséből további, rendszerszintű megfigyeléseket jegyezhetünk fel és következtetéseket vonhatunk le. A cím-, adat- és vezérlősínjelek kialakításán osztozó processzor és közvetlenmemóriahozzáférés-vezérlő kapcsolódási pontjainak közelében látható a vonalak alapállapotát beállító polarizáló ellenállás-hálózat (9.9. ábra).



9.9. ábra. *A 8 bites mikroprocesszor és DMA-vezérlő rákapcsolása a számítógép cím-, adat- és vezérlőjelsínjeire*

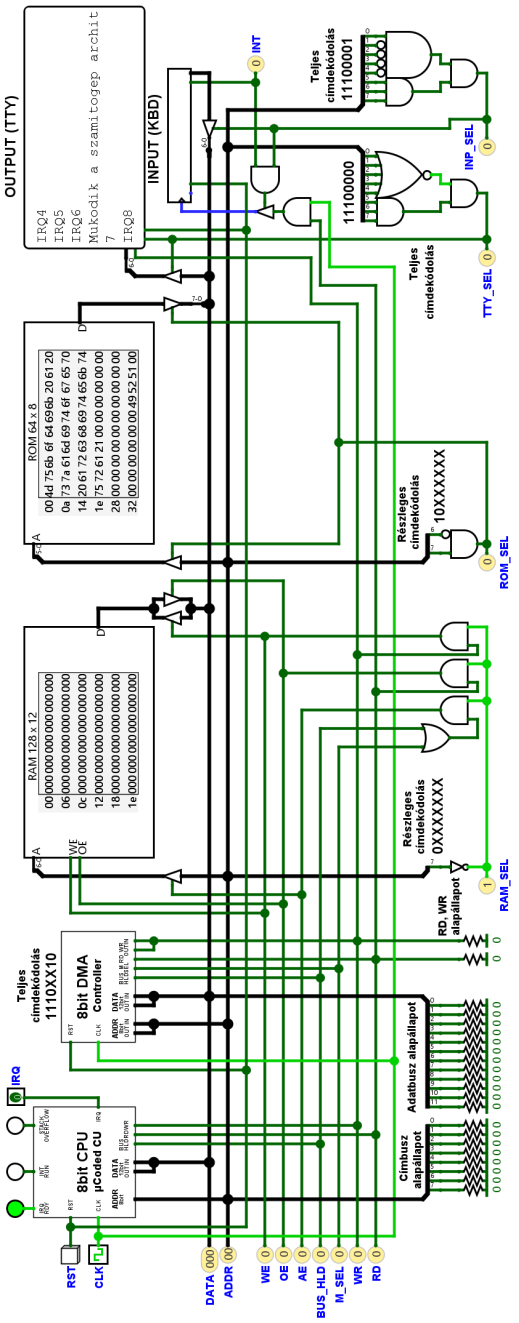
Ezeken az ellenállásokon a földpotenciál felől a bemenetek irányába folyik egy nagyon kis értékű áram (pár tíz mikroamper is elegendő). Amikor a sínek vonalai egyik periféria kimenetéhez sem kapcsolódnak (minden kimenet magas impedanciás – High Z – állapotban van), az esetlegesen bekapcsolt állapotban lévő bemeneti csatornák irányába az ellenállásokon folyó áram biztosítja a stabil, 0-ás logikai szintnek megfelelő állapotot minden vonalon. Ezzel kiküszöbölhető, hogy a sínrendszerek vonalain a kimenetek rákapcsolódása nélkül véletlenszerű értékek, zavarjelek lépjenek fel.

A 9.9. ábra kiemeli a sínrendszeren megfigyelés alá vonható jelzéseket és értékeket. Fentről lefele haladva láthatjuk a 12 bites adatsín (DATA) értékmutatóját, alatta a 8 bites címsín (ADDR) következik. A vezérlősín jelei az alapállapotba hozó RST jelzéssel kezdődnek, majd sorban a központimemória-írást kiváltó és adatsín-kapcsolódást engedélyező jelzés (WE), a központimemória-kimeneti engedélyező jelzés (OE) és a címsínkapcsolódást engedélyező jelzés (AE) következnek. Továbbá ott sorakozik a processzor sínfoglaltságot beállító jelzése (BUS_HLD), a DMA-vezérlő központi memória kiválasztó jelzése (M_SEL), végül ott találjuk a sínrendszeren lebonyolított adatátvitel irányát beállító írás (WR) és olvasás (RD) jelzéseket.

A 9.9. ábra összekapcsolható az előző alfejezetekben bemutatott két másik ábrával (9.2. ábra és 9.3. ábra), és a három együttesen mutatja az összetett számítógép-architektúra kapcsolási rajzát (9.10. ábra). Az ábra terjedelme miatt 90°-kal balra fordítva áll, áttekintéséhez döntsük jobbra a könyvet. A kapcsolási rajz felső felében az alkatrészek a számítógéprendszerben betöltött szerepüknek megfelelő legmagasabb absztraktizációs szintű, szimbolikus formában vannak ábrázolva. Balról jobbra haladva azonosíthatjuk a 8 bites processzort, a 8 bites DMA-vezérlőt, a 128 szavas RAM memóriát, a 64 szavas ROM memóriát, majd az egymás alá helyezett kijelzőt és billentyűzetperifériát. A kapcsolási rajz középső részén a cím-, adat- és vezérlőjelsínek láthatók.

A rajz alsó felében, szintén balról jobbra haladva a sínek alapállapotát beállító polarizáló ellenállás-sorozatot látjuk, majd a RAM memória elsődleges, részleges címdekódolóját, a ROM memória elsődleges, részleges címdekódolóját, aztán a kijelző és a billentyűzet teljes címdekódoló áramköreit láthatjuk. A címdekódolók környékén további sárga értékmutatókat találunk, ezek az áramkörök találatjelzői (RAM_SEL, ROM_SEL, TTY_SEL, INP_SEL). Az értékmutatók a számítógéprendszer működésének részleteit feltáró idődiagramok összeállításában játszanak fontos szerepet. Segítségükkel azonosítható, hogy mikor, hol és mi történik éppen a számítógéprendszerben. A következő alfejezetekben több ilyen idődiagramot is láthatunk majd.

Az így összeállított számítógéprendszeren példaprogramokat futtatva megfigyelhetjük a különböző utasítások végrehajtásának menetét, például a ciklikus értéknövelő algoritmusban szereplő ugrás és feltételes ugrás utasítások hatását, a DMA-átvitelt a ROM memóriából a RAM memóriába, az eljárás (procedúra) hívásokat, a megszakításkiszolgálás működését. Kísérletezhetünk szöveges üzenetek



9.10. ábra. *A számítógép-architektúra kapcsolási rajza*

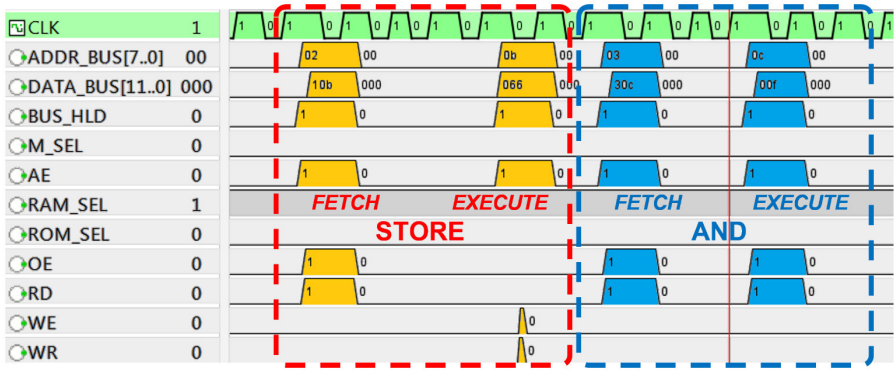
megjelenítésével a kijelzőn, és beolvashatunk a billentyűzetről bevitt karaktersorokat is. Végül előidézhetünk mesterséges rendszerhibát is a verem túlcsordulásával.

A rendszer működését jelen kiépítésében korlátozza a kis kapacitású központi memória, a cím- és adatsín szélessége, ezért a kísérletezésre szánt programok nem lehetnek nagyon hosszúak és bonyolultak, így kihívást jelentő feladat marad a kibővítése.

- i386 system block diagram
- i9 system block diagram

9.5. Programfuttatás összetett számítógép-architektúrán

Az utasítás-végrehajtás menetének vizsgálata a kialakított számítógép-architektúra szempontjából is fontos. Most nem a processzorban keletkező vezérlőjelek időbeliségét és hatását tanulmányozzuk, hanem azt vizsgáljuk, hogy a cím (ADDR_BUS), adat (DATA_BUS) és vezérlősínek állapota hogyan változik egy-egy utasítás végrehajtási ideje alatt. Ezekből a jelváltozásokból vezethető le a rendszeren belüli adatátvitel iránya (RD, WR) és üteme (CLK), ezek jelzik a címdekódolók állapotát (RAM_SEL, ROM_SEL), a memóriák cím- és adatsínjeinek a számítógéprendszer cím- és adatsínjeihez való csatlakozását (AE, OE), valamint a sínfoglaltságot (BUS_HLD, M_SEL).



9.11. ábra. *Végrehajtási ciklusok STORE és AND utasítások esetében*

Az idődiagram (9.11. ábra) az órajel üteméhez igazodva értelmezendő, egy képzeletbeli, függőleges vonal mentén, fentről lefele olvasva. A STORE utasítás (piros, szaggatott vonalas keretben) végrehajtásának előkészítő szakaszában (FETCH),

az órajel első ütemére megjelenik a cím- és adatérték, valamint aktiválódnak a sínfoglaltságot, síncsatlakozást és olvasási irányt jelző vezérlővonalak. Ezek a jelzések a következő órajelütemben deaktiválódnak.

Az ötödik ütem végén, az órajel lefutó élével szinkronban, érvényes cím- és adatérték és síncsatlakozás jelzés mellett az írási irányjelző aktiválódik.

Továbbá megfigyelhető, hogy az idődiagramban ábrázolt esetben 0x02h címértékről történik beolvasás. Az érvényes adatérték (0x10Bh) a címérték megjelenése után kerül az adatsínre, az eltolódás az adatátvitelben részt vevő kapuáramkörök összeadódó átfutási idejéből származik.

A 0x10Bh értéket felbontva következtethetünk arra, hogy egy **STORE** utasítás a 0x0Bh címre ír ki adatot. Az utasítás-végrehajtás második szakaszában (**EXECUTE**) egy időben a címsínen megjelenik a 0x0Bh címérték, az adatsínen a kiírandó érték (0x066h). Ugyancsak aktívak a sínfoglaltságot és síncsatlakozást jelző vezérlővonalak. Végül a hatodik órajel felfutó élére aktiválódnak a **WR** és **WE** jelzések is, amelyek hatására megtörténik az adatírás.

Hasonló folyamat játszódik le az összeadó (**ADD**) utasítás (kék, szaggatott vonalas keretben) végrehajtásakor is. Mivel az összeadás utasítás eredménye a munkaregiszterben tárolódik, végrehajtása során csak memóriaolvasás történik.

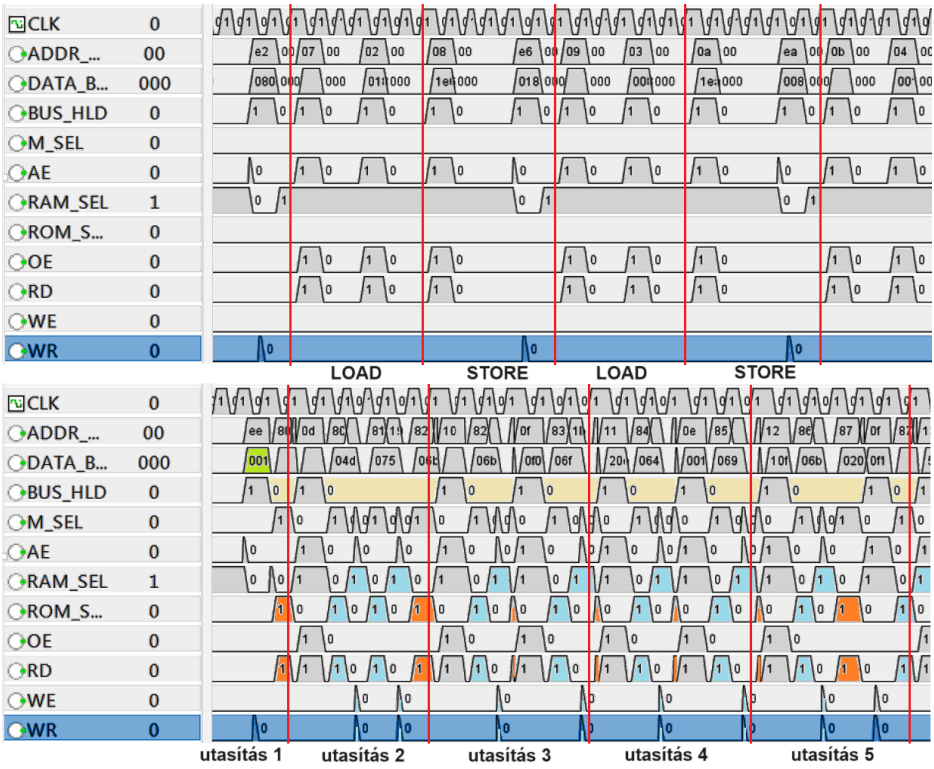
9.5.1. A DMA-átvitel szakaszai és idődiagramja

A számítógép-architektúra részeként a DMAC jelentősen növeli a rendszeren belüli adatátvitel sebességét. Egy utasítás végrehajtási ideje alatt két átviteli ciklust tud lebonyolítani, kivételt képez a **STORE** utasítás, mivel végrehajtása során a sínhasználat a processzor részéről az ötödik órajel felfutó éle után történik, és ez alatt a DMAC kénytelen megszakítani az adatátvitelt. DMA nélkül az adatátvitel két végpont között 12 órajelnyi időt vesz igénybe. DMA-átvitelt alkalmazva ez 2 órajelre csökken. Átlagban utasításonként két DMA-átvitel valósulhat meg, tehát 12 órajelnyi idő alatt 1 helyett 4 adatátvitel történhet. Így az átviteli sebesség négyszeresére növekedhet.

A jelenséget a legjobban egy összehasonlító idődiagram elemzésével érthetjük meg. A 9.12. ábra két utasítás-végrehajtási sorozatot rögzít. A felső felében az adatmozgató utasításokkal megvalósított adatátvitel idődiagramját látjuk, ismétlődő **LOAD**, **STORE** párosok sorakoznak egymás után.

A sínvezérlő jeleket mindig egy képzeletbeli függőleges mentén, fentről lefele olvasva, látjuk, hogy a **LOAD** utasítás végrehajtása alatt az érvényes cím- és adatérték, valamint sínfoglaltság jelzéssel (**BUS_HLD**) szinkronban a **RAM_SEL**, **OE** és **RD** jelzések a megfelelő órajelütemeknél 1-es állapotokat mutatnak.

A **STORE** utasítás végrehajtása közben (éppen a DMA-vezérlő **FORRÁSCÍM** regiszterét írja) az érvényes cím- és adatérték, valamint sínfoglaltság jelzéssel (**BUS_HLD**) szinkronban a **WR** jelzés a megfelelő órajelütemnél 1-es állapotot mutat.



9.12. ábra. *Utasítás-végrehajtási ciklusok összehasonlító idődiagramja DMA-átvitel nélkül (fent) és DMA-átvitellel (lent)*

Az ábra alsó felében a DMA-val megvalósított adatátvitel idődiagramját látjuk. Itt az **utasítás 1** a DMA-átvitelt indító regiszterérték beírásával végződik (zölddel kiemelt adatsínérték), és a DMA-vezérlő máris próbálkozik a ROM memóriából való adatolvasással, de közben a sínfoglaltságjelzés (BUS_HLD) aktiválásával a processzor lefoglalja a sínrendszert. A következő utasítás (**utasítás 2**, JMP) végrehajtása alatt, a sínfoglaltságjelzés feloldása után (BUS_HLD=0) a DMA-vezérlő két sikeres adatátviteli ciklust valósít meg. Hasonlóan történik az **utasítás 3** (LOAD) és **utasítás 4** (ADD) alatt is. Az **utasítás 5** (STORE) végrehajtása alatt, az előzőekben leírt okok miatt, csak egyetlen DMA-adatátvitel valósul meg.

10. Háttértároló, tárrezidens programbetöltő és operációs rendszer

Egy kereskedelmi forgalomban lévő, komplex architektúrájú, több alkatrészből álló számítógéprendszer hatékony működtetéséhez több speciális eszköz és program is szükséges. Egy ilyen számítógép architektúrája és felépítése, bár az előző fejezetekben bemutatott elvekre alapoz, komplexitásában sokszorosan meghaladja a didaktikai célból bemutatott, egyszerűsített rendszert. A működési elveket megalapozó magyarázatokból kibontakozó architektúrát minden tekintetben bővíteni kell ahhoz, hogy egy ilyen rendszert kapjunk.

A processzort át kell alakítani úgy, hogy szélesebb cím- és adatsínekkel rendelkezzen. Ezáltal képes lesz nagyobb címterületet elérni, illetve nagyobb adatértékekkel műveleteket végezni. Növelni kell a belső munkaregiszterek számát, átalakítva őket általános célú munkaregiszterek hálózatává. Ki kell terjeszteni a végrehajtható utasítások számát, hogy minél többféle számítást, adatmozgatást vagy feltételesugrás-típust lehessen egy sajátos, az adott célra kidolgozott utasítás végrehajtásával megoldani.

A központi RAM memória kapacitását a sokszorosára kell növelni, összhangban a processzor által kezelhető cím- és adatvonalak szélességével, vagy amennyiben ez nem lehetséges, cím- és adatvonal-bővítő áramkörök (multiplexer) közbeiktatásával. A háttértárolóként kezelt ROM memória kapacitását szintén sokszorosára kell növelni, és ki kell bővíteni további, nagy tárolókapacitású (több száz GB vagy több TB) elektromechanikus vagy elektronikus háttértárolókra.

A megszakításrendszert több-bemenetűre kell bővíteni, és egy olyan vezérlőt kell alkalmazni, amely prioritási sorrend szerint kezeli a beérkező megszakításkérés-jelzéseket. Ugyanez vonatkozik a DMA-alrendszerre is, a processzorhoz hasonlóan bővíteni kell a cím- és adatsín vonalainak számát, a belső adatregisztere helyett egy RAM-tárolót kell alkalmazni, nagyobb adattömbök ideiglenes tárolására és lehetőség szerint több, párhuzamos adatátviteli csatornát (multi channel DMA) is ki kell alakítani, amelyek egymással versengenek majd a szabad cím- és adatsínidők kihasználásáért.

Szintén bővíteni kell a kijelző- és adatbeviteli rendszert. A kijelzőmeghajtót működtető áramkör kijelzési sebessége sokat nő a cím- és adatsín-szélesítéssel, valamint a helyi adattároló RAM memória (videomemória) beépítésével. A továbbiakban érdemes a rendszert egy adatátviteli csatolóáramkörrel is felszerelni, amelyen keresztül más számítógéprendszerek hasonló adatcsatornáival kommunikálhat.

A felsoroltak mellett a rendszerhez hozzá kell adni egy megfelelő tárkapacitású háttértárolót is. A kibővített, adatátviteli és számítási kapacitásában sokszorosára növelt számítógéprendszert egy alapprogram, az operációs rendszer

(OS – Operating System) segítségével és közbeiktatásával működtetik. Ezt a programot a számítógéprendszerben bináris adatként a háttértárolón rögzítik, és a rendszer indulásakor egy speciális program segítségével innen töltődik be a központi memóriába.

A kibővített rendszerről alkotható átfogó kép árnyalásához vizsgáljuk meg a háttértárolók, betöltő programok és operációs rendszerek működési elvét.

10.1. Háttértároló rendszerek

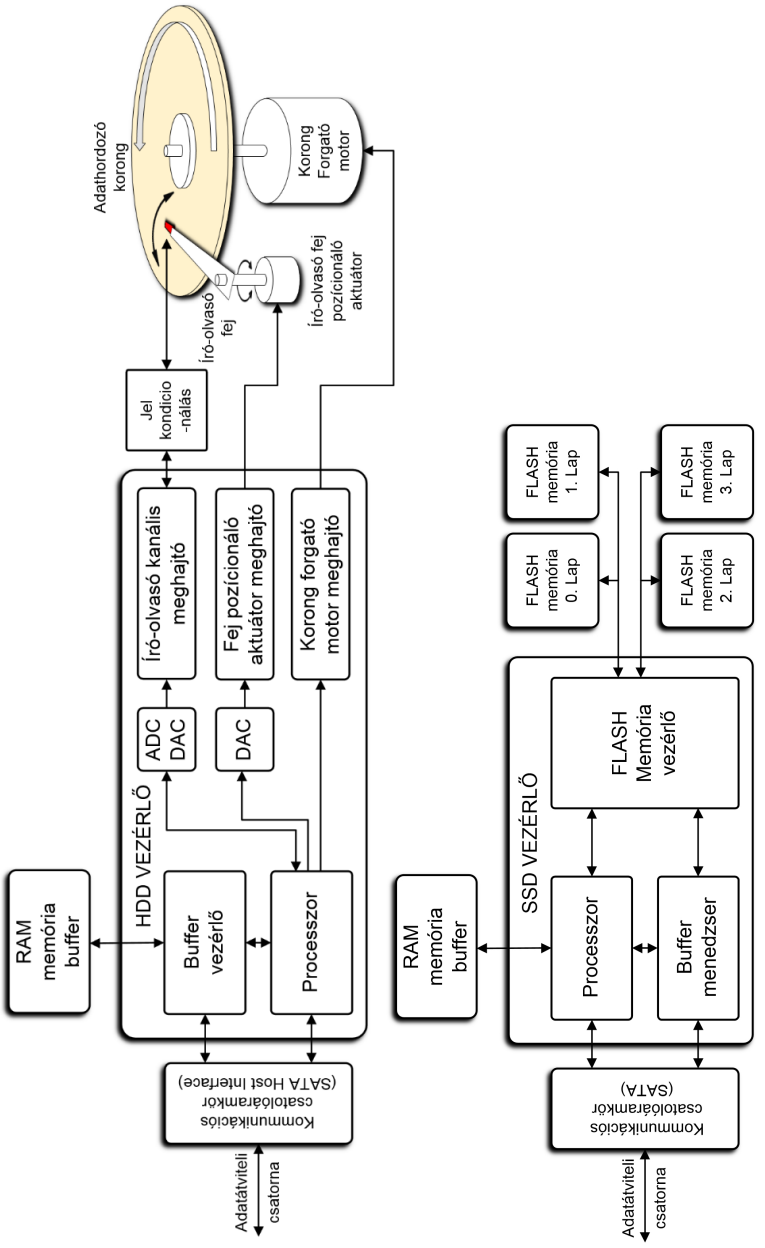
A számítógéprendszerek a futtatható programok tárolására olyan adathordozókat használnak, amelyek tápellátás hiányában is hosszú ideig (akár évtizedekig) képesek megtartani információtartalmukat. Az adathordozók története valahol a középkori mechanikus vezérlésű zenélő gépeknél kezdődik, tengelyen forgó fahengerbe vert szegekkel, amelyek billentyűket nyomogatnak vagy rezgő fémlemezeket pengetnek. Majd következnek Jacquard szövőszékének fából készült lyukkártyái, aztán Babbage analitikus gépének kartonlapkái, amelyeket mechanikus úton hoznak létre és „olvasnak” le. Az első számítógépek papírból készült lyukkártyáit, aztán lyukszalagjait még mechanikus módszerrel „írják”, de már fényérzékelőkkel „olvassák” le a rögzített adatokat. Végül Gustav Tauschek forgó dobra szerelt kondenzátorsorai következnek, ekkor az adatrögzítés és -visszanyerés már elektronikus módszerrel történik.

A kondenzátorsorokat hamarosan feltekert, hosszú mágnesszalagokra cserélik, amelyeket elektromágneses „író-olvasó fejek” előtt húznak el, ezek segítségével, elektromos impulzusokat használva tárolják és nyerik vissza az adatokat. Az eddigi módszerek nagy hátránya, hogy az adatokat csak sorban lehet elérni, így a szalagok végén tárolt adatok beolvasásához a teljes tárcsát végig kell pörgetni.

Ezt a megkötést oldja fel a mágnesezhető anyagréteggel bevont, gyorsan pörgő fémkorongot használó háttértároló megjelenése (HDD – Hard Disc Drive). A korong fölött lebegő, sugárirányban mozgó „író-olvasó” fej a korong felületének bármelyik pontját el tudja érni pár fordulat alatt. Később a tárolókapacitás növelése érdekében molekuláris szintre csökkentik a mágnesezhető részecskék méretét. Napjainkban ugyanazon a korongfelületen, amelyen kezdetben pár száz kilobájtot vagy később egy-két megabájtot tudtak tárolni, most sok tíz terabájtnyi adatot lehet rögzíteni (az adatsűrűség tízmilliószorosára nőtt).

Idővel a merevlemezes háttértárolók mellett megjelentek a mozgó alkatrészeket nem használó, memóriaalapú tárolók (SSD – Solid State Drive). Ezek olyan EEPROM (Electrically Erasable and Programmable Read Only Memory) memóriákat használnak, amelyek mikroszkopikus kondenzátorok feltöltésével évtizedekig képesek tárolni a beírt információt. Ma a két technológia egymás mellett él.

A két meghajtó között számos hasonlóságot fedezhetünk fel. A 10.1. ábra az összehasonlítást könnyebbé téve, egymás alatt ábrázolja a két technológia főbb



10.1. ábra. *Merevlemezes (HDD) és EEPROM memóriákból kialakított (SSD) háttértárolók tömbvázlata*

alkotóelemeit. A merevlemezes háttértároló vezérlőáramkörének gondoskodnia kell a korongforgató motor és az író-olvasó fej aktuátorának meghajtásáról és precíziós pozicionálásáról. A beolvasott jelalak egy kondicionáló áramkörön halad át, így megszűrve és felerősítve kerül beolvasáskor az ADC-áramkör bemenetére.

A vezérlőáramkör processzora digitális jeleket dolgoz fel, és egy lokális, RAM memória bufferrel gyorsítja az írás-olvasás folyamatát. A „kiolvasott" vagy eltárolandó adatokat egy kommunikációs csatolóáramkör (SATA – Serial Advanced Technology Attachment, vagy más szabvány, például IDE – Integrated Drive Electronics, SCSI – Small Computer System Interface) közvetítésével kapcsolja a számítógép cím-, adat- és vezérlőjelsínjeire.

A memóriamodulokat használó háttértároló vezérlőáramköre mellőzi a motormeghajtó modulokat, jelkondicionáló és digitalizáló áramköröket, helyettük egy FLASH memóriavezérlő modul csatlakozik a processzorához, amely a lapokra osztott memóriamodulokat kezeli. A vezérlő egy lokális, RAM memória bufferrel gyorsítja az írás-olvasás folyamatát. A „kiolvasott" vagy eltárolandó adatokat a HDD-vel egyező módon egy kommunikációs csatolóáramkör (SATA, SCSI) kapcsolja a számítógép cím-, adat- és vezérlőjelsínjeire. A merevlemezes háttértárolóval szembeni nagy előnye a memóriamodulok használatából fakadó, sokszorosára növekedett írási és olvasási sebesség (több mint 70-szeres írási-, több mint 50-szeres olvasásisebesség-növekedés) és a lecsökkent energiafogyasztás.

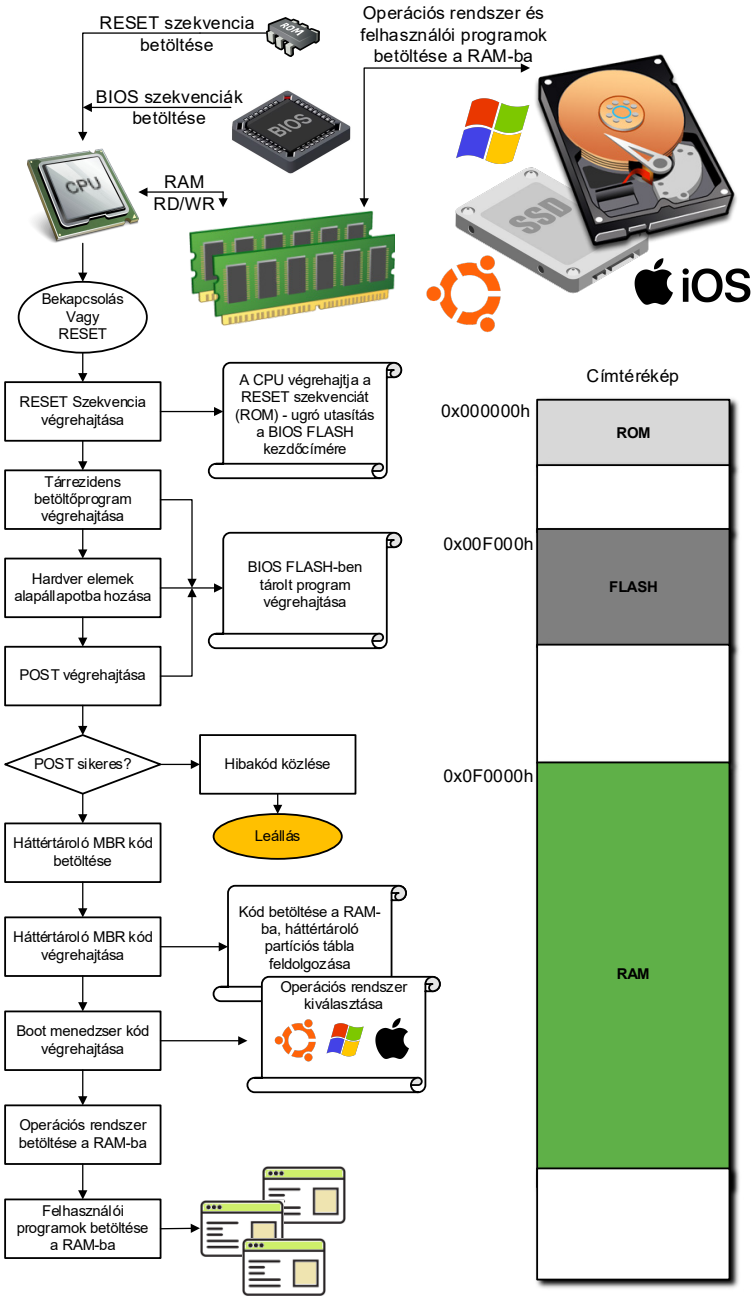
A felsorolt adattárolási módszerek mellett más módszerek is megjelentek, majd később eltűntek a technikatörténet süllyesztőjében, mint például a hajlékony (Floppy Disc), a CD (Compact Disc) és a DVD (Digital Video Disc) lemezek.

- how HDD works
- how SSD works

10.2. Tárrezidens betöltőprogram (bootloader)

A tárrezidens programbetöltő egy olyan alapprogram, amelyet a számítógéprendszer indításakor (elektromos energiával való betáplálása után) a processzor elsőként hajt végre. Ez a program (BIOS – Basic Input/Output System) rendszerspecifikusan tartalmazza az adott számítógép-konfiguráció adatait, a csatlakoztatott áramkörök jegyzékét és az alapbeállításaikhoz szükséges adatokat, valamint egy alapvető működési paramétereket felmérő algoritmust (POST – Power On Self-Test).

A 10.2. ábra szerint az indításkor alapállapotban lévő processzor egy előre meghatározott címről, általában a legkisebb vagy legnagyobb elérhető címértékről (RESET VECTOR – 0x000000h) egy ugró utasítást hajt végre a BIOS-program



10.2. ábra. *A tárrezidens betöltőprogram (bootloader) működési elvének vázlata*

kezdőcímére. A BIOS-programot egy EEPROM-típusú memóriában tárolják, így lehetőség van az újraírására.

A BIOS-csomag elsőként végrehajtott programmodulja egy tárrezidens betöltőprogram. Ennek feladata minden olyan további programmodul betöltése a központi memóriába, amelyek majd a számítógéprendszert alkotó hardver elemek (alkatrészek) alapállapotba hozását végzik. Az alapállapotba hozott alkatrészek működésének ellenőrzését a betöltött POST programmodul végzi el. Ha valamelyik, a számítógéprendszer működése szempontjából kritikus alkatrész hibás válaszokat ad vagy egyáltalán nem válaszol, az indítási folyamat itt leáll. Sikeres POST után az elsődleges betöltőprogram a számítógéprendszer háttértárolójáról a központi memóriába másolt, másodlagos betöltőprogramnak (MBR – Master Boot Record háttértároló szektorban kezdődő, végrehajtható program) adja át a kezdeményezést. Ez a másodlagos betöltőprogram feldolgozza a háttértároló partíciós (adatelrendezési) táblázatát, ennek alapján megkeresi és végrehajtja a harmadlagos betöltésfelügyelő programot (Boot manager).

Ez a harmadlagos betöltőprogram megkeresi a háttértárolón fellelhető aktív partíciókat, és választási lehetőséget kínál a felhasználónak, hogy melyik operációs rendszert kezdje majd betölteni a központi memóriába.

A továbbiakban a kiválasztott operációs rendszer programmoduljai a háttértárolóból átmásolódnak a központi memóriába. Az operációs rendszer végrehajtása során további felhasználói programok betöltésére van lehetőség. A folytatásban vizsgáljuk meg az operációs rendszer működési elvét.

- BIOS
- how OS bootloader works

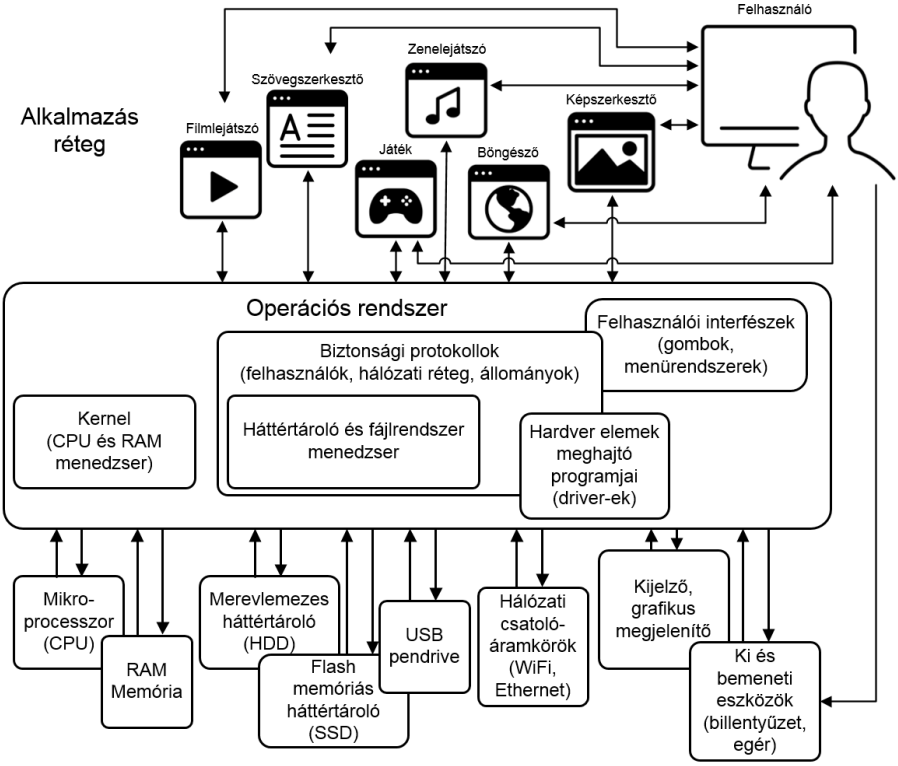
10.3. Operációs rendszer

A komplex architektúrájú, több alkatrészből álló számítógéprendszer hatékony működtetése egy alapprogram segítségével történik. Ez az alapprogram, az operációs rendszer (OS – Operating System), köztes logikai, virtuális rétegként helyezkedik el a felhasználói programok szintje és a számítógéprendszer alkatrészei között. Az alapprogram betöltése után (lásd a 10.2. alfejezetben) megkeresi a processzoron és rendszermemórián kívül a cím-, adat- és vezérlővonalakra kapcsolt számítógép-alkatrészeket, és regisztereik megfelelő adatokkal való feltöltésével alapállapotba hozza őket. Ezután következik az úgynevezett erőforrás-menedzsment, vagyis az alapprogram a futtatott felhasználói programok számára biztosítja az akadálytalan hozzáférést a rendszer alkatrészeihez, kiemelten felvigyázva a

rendszermemória és a processzor (magjainak és végrehajtási vonalainak), valamint a háttértárolók használatát.

Az operációs rendszer működési elvét bemutató 10.3. ábra szerint a felhasználót kép- és hanginformációkkal ellátó alkalmazások (felhasználói programok) az alapprogramon keresztül érik el a kimeneti csatolóáramköröket (kijelző, hangkártya). Ugyancsak rajta keresztül történik a felhasználói beavatkozásokat közvetítő perifériák jelzéseinek felhasználói programok felé közvetítése (billentyűzet, egér, érintőképernyő).

A felhasználói programok olyan alkalmazások, amelyek valamilyen felhasználói igényt elégítenek ki. Az alapvetően matematikai számítások elvégzésének gyorsítására kitalált számítógépek mára multimédiás, mindennapi használati eszközökké váltak, amelyeken számtalan variációban futtathatók szövegszerkesztő, képszerkesztő, zenelejátszó, filmlejátszó, világhálón való keresgélést és kommunikációt segítő (böngésző) programok is.



10.3. ábra. *Az operációs rendszer működési elvének vázlata*

A felhasználói élményeket kielégítő alkalmazások mellett számtalan matematikai számításokra alapozó program létezik, idesorolható minden tervezői (elektronikai, mechanikai, építészeti elemek) és szimulációs felület vagy a kimondottan matematikai feladatok megoldására (szimbolikus vagy analitikus módszerekkel) szakosodott program. Külön kategóriát képeznek a programok létrehozására szolgáló programozói-fejlesztői környezetek (IDE – Integrated Development Environment), amelyek a szövegszerkesztők, absztraktizáló rétegek, eszközválasztó és -beállító eszközök és forráskódfordítók sajátos kombinációiból állnak.

Az operációs rendszer, mint alapvető keretprogram, magába foglalja a processzor és rendszermemória erőforrás-felügyeleti kernel részt, a hardver elemek meghajtóprogramjainak csatolófelületeit (szoftver-interfészek), a felhasználói ki- és bemeneti felületek (grafikusan megjelenített menürendszerek) és eszközök meghajtóprogramjait. Ezek mellett a biztonsági eljárások (protokollok) rendszerébe ágyazottan kezeli a felhasználók jogrendszerét, a hálózati adatátviteli réteget és a tartalomfelügyeletet (állomány- vagy fájlmenedzsment). Ez utóbbit a háttértároló és fájlrendszer felügyeleti alapalkalmazásra építve működteti.

Az operációs rendszer több felhasználói program párhuzamos (de időben osztott módon kezelt) futtatását (végrehajtását) is megengedi (multitasking). Ilyenkor az egymással párhuzamosan futó programok között, azok „kérésére” (hozzáférési eljárások üzenetei) dinamikusan osztja el a számítógéprendszer alkatrészeihez való hozzáférést. Memóriaterületeket jelöl ki és foglal le a programok számára, hozzáférést enged a háttértárolóhoz, ki- és bemeneti csatolóáramkörökhöz, szabályozza az egyes programok processzor általi végrehajtásának időtartamát.

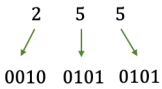
Ahogy a környező világ egyre több szegmenséhez létezik digitális hozzáférést biztosító csatolóáramkör és alkalmazás (program), úgy bővül azoknak a szolgáltatásoknak és felületeknek a száma, amelyeken keresztül ezek, egy számítógéprendszer közbeiktatásával, elérhetővé válnak a felhasználó számára. Ezért a számítógéprendszer indítása után betöltött és végrehajtott alapprogramot úgy képzeljük el, mint egy állandóan jelen lévő felügyeleti rendszert, amely a számítógépbe beépített alkatrészek és a végrehajtott felhasználói programok közötti kapcsolatot fenntartja és rendszerezi, kihasználva az adott számítógép erőforrásait (asztali vagy hordozható eszközök) a minél teljesebb felhasználói „élmény” érdekében.

- how OS works
- differences between operating systems

11. Melléklet

11.1. BCD-kódolás

A BCD (Binary Coded Decimal – binárisan kódolt decimális) a tízes számrendszer számjegyeinek bináris kódolási formája. Minden tízes számrendszerbeli helyértéknek megfelelő számjegyet 4 bitből álló sorozattal ábrázol.



Megoldás maradékos osztás módszerével:

$1111\ 1111_b$

11.1. ábra. *Tízes számrendszerbeli szám felbontása BCD-kódokra*

A számítástechnikai rendszerekben az adatokat 8 bites csoportokba (bájtokba), illetve a 8 bit többszöröseit kitevő formákba (word, double stb.) rendezik. Ezért a 4 bites BCD-számjegyek bájtalapú tárolására kétféle módszer is alkalmazható. Vagy minden tízes számrendszerbeli helyértéknek megfelelő számjegyet egy bájton tárolnak, ahol a felső négy bit, alkalmazástól függően, 1111 (az EBCDIC kódnál), vagy 0011 (ASCII kódnál) is lehet, vagy két számjegyet tárolnak minden bájton, a nagyobb helyértéket képviselő számjegyet bal felé rendezve.

A számítógépek hagyományosan BCD-formában tárolták és kezelték a dátum- és időértéket, de épültek a számjegyeket BCD-formában kezelő processzorarchitektúrák is. Ezzel az ábrázolási módszerrel a számok konverziója és megjelenítése egyszerűbbé válik, de az aritmetikai számításokhoz több áramkörre van szükségünk, és a tárolóterületet sem tudjuk annyira kihasználni, mint a tisztán bináris ábrázolás esetében.

11.2. Karakterek kódolása

Az ASCII (American Standard Code for Information Interchange) kód jelkészlete az angol ábécé nagy- és kisbetűit, a tízes számrendszer számjegyeit, írásjeleket, grafikus szimbólumokat és vezérlőkódokat foglal magába. Ezeket a jeleket 0-tól 255-ig terjedő, előjel nélküli, egész számokra képezi le. Ezeket a számokat a számítástechnikai rendszerek bináris formában tárolják, használják és értelmezik. Az alap-karakterkészlet 0-tól 127-ig terjed (11.2. ábra, tinyurl.com/yc4m4mm3).

ASCII kontrol karakterek ASCII control characters			
000	0x00h	NULL	Null character
001	0x01h	SOH	Start of Header
002	0x02h	STX	Start of Text
003	0x03h	ETX	End of Text
004	0x04h	EOT	End of Trans.
005	0x05h	ENQ	Enquiry
006	0x06h	ACK	Acknowledgement
007	0x07h	BEL	Bell
008	**0x08h**	**BS**	**Backspace**
009	0x09h	HT	Horizontal Tab
010	**0x0Ah**	**LF**	**Line feed**
011	0x0Bh	VT	Vertical Tab
012	0x0Ch	FF	Form feed
013	**0x0Dh**	**CR**	**Carriage return**
014	0x0Eh	SO	Shift Out
015	0x0Fh	SI	Shift In
016	0x10h	DLE	Data link escape
017	0x11h	DC1	Device control 1
018	0x12h	DC2	Device control 2
019	0x13h	DC3	Device control 3
020	0x14h	DC4	Device control 4
021	0x15h	NAK	Negative acknowl.
022	0x16h	SYN	Synchronous idle
023	0x17h	ETB	End of trans. Block
024	0x18h	CAN	Cancel
025	0x19h	EM	End of medium
026	0x1Ah	SUB	Substitute
027	0x1Bh	ESC	Escape
028	0x1Ch	FS	File separator
029	0x1Dh	GS	Group separator
030	0x1Eh	RS	Record separator
031	0x1Fh	US	Unit separator
127	0x7Fh	DEL	Delete

ASCII nyomtatható karakterek ASCII printable characters								
032	0x20h	**space**	064	0x40h	@	096	0x60h	`
033	0x21h	!	065	0x41h	A	097	0x61h	a
034	0x22h	"	066	0x42h	B	098	0x62h	b
035	0x23h	#	067	0x43h	C	099	0x63h	c
036	0x24h	$	068	0x44h	D	100	0x64h	d
037	0x25h	%	069	0x45h	E	101	0x65h	e
038	0x26h	&	070	0x46h	F	102	0x66h	f
039	0x27h	'	071	0x47h	G	103	0x67h	g
040	0x28h	(	072	0x48h	H	104	0x68h	h
041	0x29h	)	073	0x49h	I	105	0x69h	i
042	0x2Ah	*	074	0x4Ah	J	106	0x6Ah	j
043	0x2Bh	+	075	0x4Bh	K	107	0x6Bh	k
044	0x2Ch	,	076	0x4Ch	L	108	0x6Ch	l
045	0x2Dh	-	077	0x4Dh	M	109	0x6Dh	m
046	0x2Eh	.	078	0x4Eh	N	110	0x6Eh	n
047	0x2Fh	/	079	0x4Fh	O	111	0x6Fh	o
048	0x30h	0	080	0x50h	P	112	0x70h	p
049	0x31h	1	081	0x51h	Q	113	0x71h	q
050	0x32h	2	082	0x52h	R	114	0x72h	r
051	0x33h	3	083	0x53h	S	115	0x73h	s
052	0x34h	4	084	0x54h	T	116	0x74h	t
053	0x35h	5	085	0x55h	U	117	0x75h	u
054	0x36h	6	086	0x56h	V	118	0x76h	v
055	0x37h	7	087	0x57h	W	119	0x77h	w
056	0x38h	8	088	0x58h	X	120	0x78h	x
057	0x39h	9	089	0x59h	Y	121	0x79h	y
058	0x3Ah	:	090	0x5Ah	Z	122	0x7Ah	z
059	0x3Bh	;	091	0x5Bh	[	123	0x7Bh	{
060	0x3Ch	<	092	0x5Ch	\	124	0x7Ch	|
061	0x3Dh	=	093	0x5Dh	]	125	0x7Dh	}
062	0x3Eh	>	094	0x5Eh	^	126	0x7Eh	~
063	0x3Fh	?	095	0x5Fh	_			

11.2. ábra. *Alap ASCII-karakterkészlet táblázata*

- BCD computer arithmetic
- ASCII character encoding

Érdekes információ, hogy amíg a mai számítógépek billentyűzetein egyetlen „sorvég és új sorba” nyomógomb jelenik meg – az ENTER –, addig egyes operációs rendszerekben (pl. Windows) ezt két kód jeleníti meg: a Line feed (10_D, $0A_H$) és Carriage return (13_D, $0D_H$). Ennek technikatörténeti oka az írógépek mechanikai kialakításában keresendő. Egy sor begépelése után a lapot egy sorközzel fennebb kellett tekerni (innen a Line feed), és jobb felé a lapszélig eltolni a laptartó szekeret

(innen a Carriage return) az írófej előtt. Ez két művelet. A számítástechnika fejlődéstörténetének kezdetén a kimeneti adatok megjelenítésére kézenfekvő volt motorizált írógépet használni, később a nyomtatók is ezen az elven működtek, így ez a két utasításkód máig megmaradt. Linux operációs rendszer alatt csak a line feed utasítás kódolja az új sorba ugrást.

11.3. A Logisim Evolution szoftver használata

A Logisim Evolution digitális logikai áramkörök tervezésére és szimulálására használható, nyílt forráskódú (open source) oktatási segédeszköz. Egyszerű felépítésével, funkciópalettáival és az áramkörök virtuális működésének szimulációjával megkönnyíti a logikai áramkörökkel, számítógéprendszerekkel kapcsolatos legalapvetőbb fogalmak elsajátítását. A Logisim Evolution környezet támogatja a hierarchikus tervezést, így képesek vagyunk általa kisebb áramkörökből nagyobb áramköröket építeni és síneket (adatvonalakat) húzni közöttük. Ebből fakadóan kiválóan alkalmazható komplex digitális áramkörök, processzorok, memóriák és számítógéprendszerek didaktikai célú tervezéséhez és szimulációjához.

A nyílt forráskód és a legfrissebb fordított verziók egy GitHub könyvtárban (repository) találhatók. Ugyanitt találjuk a program specifikációit és a letöltéssel, futtatással kapcsolatos információkat is (https://github.com/logisim-evolution/).

- github logisim evolution
- Logisim Evolution tutorial

11.3.1. Szerkesztőablak és szerszámpaletták

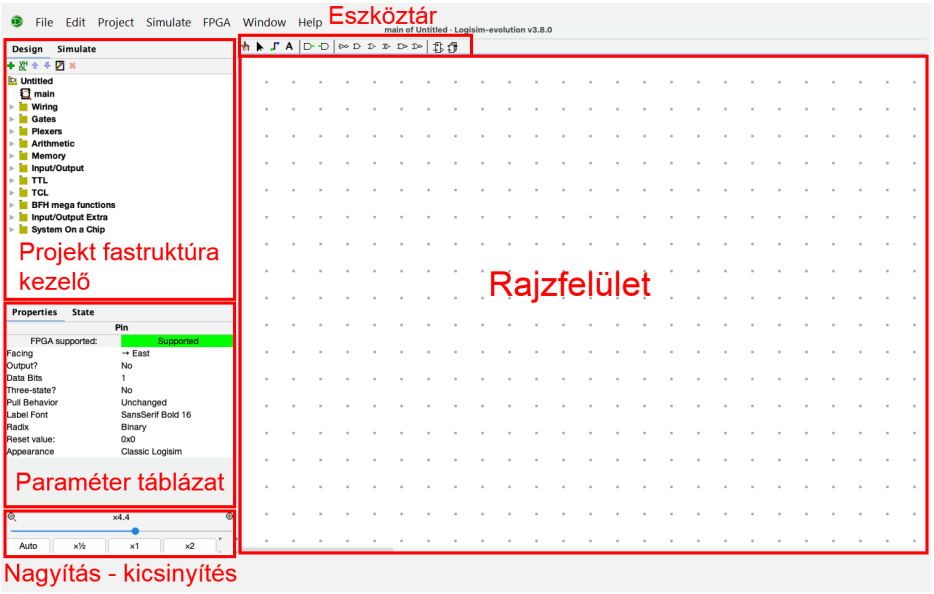
A 11.3. ábra a Logisim Evolution programkörnyezet kezdőképernyőjét mutatja, amelyen a fontosabb részeket piros keret és felirat jelöli.

A **Menü**sor: File, Edit, Project, Simulate, Windows és Help menüket tartalmazza. Néhány fontosabb beállítási lehetőség:

– Project: *Add Circuit* parancs segítségével új rajzfelületet (canvas) tudunk létrehozni.

– Simulate: Auto-Ticks Enabled parancs segítségével elindítjuk a szimulációt; Auto-Tick Frequency opcióval kiválaszthatjuk, hogy a szimuláció milyen sebességgel fusson (órajel frekvenciát megadva).

– Help: User's Guide alpontját megnyitva részletes leírást kapunk a programról és használatáról, valamint lépésről lépésre felépített bemutató (tutorial) is megtalálható.



11.3. ábra. *Logisim Evolution kezdőképernyője*

Az **Eszköztár** (11.4. ábra) a leggyakrabban használt eszközöket, áramköri elemeket tartalmazza, elősegítve a gyors hozzáférést (beállítható, hogy mely eszköz legyen elérhető innen).

– Poke Tool (11.4. ábra a) pontja): egy komponensekhez társított aktuális értékek manipulálására szolgál.

– Edit Tool (11.4. ábra b): lehetővé teszi a felhasználó számára a meglévő komponensek átrendezését és vezetékek hozzáadását. Az eszköz hatása ott nyilvánul meg, ahol a felhasználó megnyomja a bal egérbillentyűt a rajzfelületen.

– Wiring tool (11.4. ábra c) pontja): lehetővé teszi a kábelek elhelyezését az ábrán.

– Text (11.4. ábra d) pontja): egyszerű szöveges címke szerkesztése, amely bárhol elhelyezhető az áramköri rajzon.

– Pin Tool (11.4. ábra e) pontja): egy áramkör kimenete vagy bemenete a paraméter-táblázatban található „Output?” attribútum értékének beállításával határozható meg. Ettől függően az áramkör-kivezetés (pin) rajzolásakor a program a kimeneteket kör vagy lekerekített téglalap segítségével, a bemeneteket négyzetekkel vagy téglalapokkal ábrázolja (Classic Logisim megjelenítés esetében).

– Gates (11.4. ábra f) pontja): a Gates könyvtár gyorsított elérése, kapuáramkörök szimbólumait tartalmazza, amelyek mindegyike egyetlen kimenettel rendelkezik.

11.4. ábra. *Leggyakrabban használt eszközök szimbólumai*

Projektfastruktúra-kezelő: mappák segítségével csoportosítva megtalálható az összes eszköz és áramköri elem, melyeket felhasználva saját áramköröket építhetünk meg.

Rajzfelület: Ezen a felületen helyezzük el és kapcsoljuk áramköri hálózatba a különböző alkatrészeket.

Paraméter-táblázat: a rajzfelületen kiválasztott elem beállításainak szerkeszthető listája jelenik itt meg.

Nagyítás – **kicsinyítés**: a csúszka vagy az alatta lévő gombok segítségével állítható be a látható rajzfelület mérete. Ha erős nagyítást alkalmazunk, egyetlen alkatrészre koncentrálhatunk. Az „Auto” gomb megnyomása úgy állítja be a rajzfelületet, hogy a lehető legtöbb alkatrész láthatóvá váljon.

Az **állapotkijelző négyzet** akkor jelenik meg, amikor az egérmutatóval rákattintunk egy tetszőleges vezetékre vagy sínre a kapcsolási rajzban. Ilyenkor a vonal felett vagy alatt megjelenő sárga téglalapban leolvasható a pillanatnyi jelérték.

11.3.2. Virtuális alkatrészek használata példákkal

A következő alpontok pár fontosabb alkatrész-tulajdonságainak és beállításainak részleteit foglalják össze. Ezekkel az alkatrészekkel fogunk a laboratóriumi gyakorlatok során a leggyakrabban találkozni, ezekből építjük fel bonyolultabb áramköreinket, ilyenek például az órajelforrás, a logikai kapuk, az összeköttetéseket jelölő huzalok és sínek, a multiplexer és demultiplexer áramkörök, a RAM és ROM memória modulok, valamint a kijelző és billentyűzet mint ki- és beviteli eszközök.

10.2.1.1. Órajelforrás

Az órajelforrás a szimulációs környezet egyik fontos alkatrésze. Segítségével egy periodikusan ismétlődő, virtuális jelet hozunk létre, amely minden hozzá kapcsolt további alkatrész felé közvetíti az egységes, 0, 1, 0, 1 stb. logikai jelértékváltozást. Példánkban az órajelforrást egy **TAGADÓ** kapu bemenetéhez csatlakoztatjuk (11.5. ábra), és megfigyeljük a kimeneti érték változását. Ez közvetlen összefüggésben lesz az órajel frekvenciájával.

Alább tekintsük át a felhasznált alkatrészek fontosabb beállításait.

Órajel

Kiválasztás: Wiring → Clock

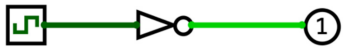
Facing: East – a komponens kimenete milyen irányba mutasson
Label: komponens feliratozása
Működés: az órajel ütemére (tick) a komponens a 0-ás és 1-es értékeket váltogatja kimenetén, a szimuláció elindítása után (CTRL + K) automatikus váltásokat figyelhetünk meg. A virtuális jelforrás frekvenciája a Simulate → Autó-Tick Frequency menüből állítható be;

TAGADÓ kapu

Kiválasztás: Gates → NOT Gate
Működés: a kapu a működési táblázata alapján a bemenetén megjelenő logikai értéket tagadja, ha 0 értéket kap bemenetként, akkor 1-es értéket kapunk a kimeneten, és fordítva;

Jelállapot-meghatározó

Kiválasztás: Wiring → Pin
Output?: Yes
Data Bits: 1 – 1 bit értéket kaphat bemenetként (lehet több is, igény szerint)



11.5. ábra. *NOT kapu használata*

10.2.1.2. Sínek képzése, adatok átvitele alkatrészek között

A kapcsolási rajzok érthetőségét nagyban növeli a sínek alkalmazása. Ilyenkor a több párhuzamos vezetékből álló sínt egyetlen vastag vonallal helyettesítjük. Példánkban négybites, bináris értéket kapcsolunk egy bemenetként konfigurált jelállapot-kijelző (pin) segítségével egy sínre, majd a sínen haladó jelek bináris értéket egy kimenetként konfigurált jelállapot-kijelző (pin) segítségével megjelenítjük (11.6. ábra). A bemeneti értékeket a *Pooke Tool* segítségével tudjuk megadni.
Alább tekintsük át a felhasznált alkatrészek fontosabb beállításait.
Pin - input:
Output?: No
Data Bits: 4 – 4 bites bináris értéket tudunk bevinni
Pin - out:
Output?: Yes
Data Bits: 4 – egy 4 bites bináris értéket tudunk megjeleníteni

11.6. ábra. *Egyszerű értéktovábbítás és -megjelenítés*

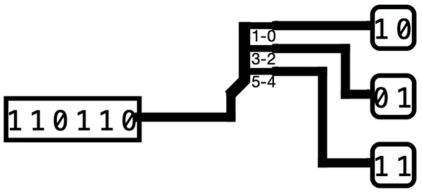
10.2.1.3. Sínek jeleinek szétszálazása

Adott sín jeleit csoportokra bontva is használhatjuk. Példánkban hatbites értéket dolgozunk fel egy sínszétszálazó vagy *Splitter* komponens segítségével, 3 egyenlő részre osztva a bemeneti értéket (11.7. ábra).

Alább tekintsük át a felhasznált alkatrészek fontosabb beállításait.

Jelállapot-meghatározó
Kiválasztás: Wiring → Pin
Output?: No
Data Bits: 6
Pin - input →1,2,3:
Output?: Yes
Data Bits: 2

Sínszétszálazó
Kiválasztás: Wiring → Splitter:
Fan Out: 3
Bit Width In: 6
Appearance: Left-handed



11.7. ábra. *Szétszálazó (Splitter) komponens alkalmazása*

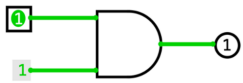
10.2.1.4. Logikai kapuk használata

Egy **és** kapu használatával szemléltetjük a műveletvégzést. A kapu egyik bemeneti értékét egy konstans komponens adja, a másikat egy bemenetként konfigurált jelállapot-kijelző (11.8. ábra).

Alább tekintsük át a felhasznált alkatrészek fontosabb beállításait.

Jelállapot-meghatározó
Kiválasztás: Wiring → Pin
Pin - input:
Output?: No
Data Bits: 1
Konstansérték-bevitel
Kiválasztás: Wiring → Constant

Data Bits: 1
Value: 0x1 – hexadecimálisan adjuk meg a konstans értékét
ÉS kapu
Kiválasztás: Gates → AND Gate
Data Bits: 1
Number of Inputs: 2
Pin - output:
Output?: Yes
Data Bits: 1



11.8. ábra. *Kétbemenetes AND kapu használata*

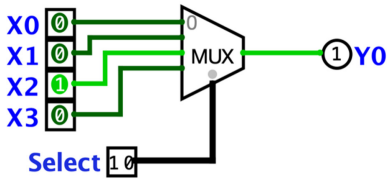
10.2.1.5. Multiplexer és demultiplexer áramkör használata

A multiplexerek és demultiplexerek a digitális rendszerek adatútválasztó alkatrészei. A **multiplexer** fő funkciója a több bemenetére érkező jelzés közül egyetlen, kiválasztott csatorna továbbkapcsolása a kimenet felé. A kiválasztás bináris kód alapján történik. N számú bit felhasználásával 2^N számú bemenet közül választható ki az, amelyik értékét tovább kívánjuk kapcsolni (11.9. ábra).

Alább tekintsük át a felhasznált alkatrészek fontosabb beállításait.

Jelállapot-meghatározó
Kiválasztás: Wiring → Pin
Pin - input → X0, X1, X2, X3 bemenetek címkézése
Output?: No
Data Bits: 1
Jelállapot-meghatározó
Pin - input → Select kiválasztó érték címkézése
Output?: No
Data Bits: 2
Jelállapot-meghatározó
Pin - output → Y0 kimenet címkézése
Output?: Yes
Data Bits: 1
Multiplexer
Kiválasztás: Plexers → Multiplexer
Select Bits: 2

Működés: a kimenet (Y0) felveszi a kiválasztó bemenetek (Select) segítségével megjelölt bemenet értékét (X2).



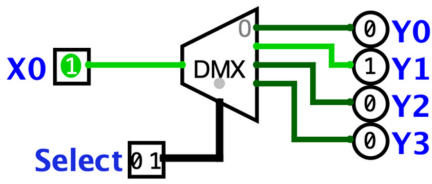
11.9. ábra. *Multiplexer használata*

A **demultiplexer** fő funkciója a bemenetére érkező jelzés egyetlen, kiválasztott kimeneti csatorna felé való továbbítása. A kiválasztás bináris kód alapján történik. N számú bit felhasználásával 2^N számú kimenet közül választható ki az, amelyik felé a bemenet értékét tovább kívánjuk kapcsolni (11.10. ábra).

Alább tekintsük át a felhasznált alkatrészek fontosabb beállításait.

Jelállapot-meghatározó
Kiválasztás: Wiring → Pin
Pin - input → X0 bemenet címkézése
Output?: No
Data Bits: 1
Jelállapot-meghatározó
Pin - output → Y0, Y1, Y2, Y3 kimenetek címkézése
Output?: Yes
Data Bits: 1
Jelállapot-meghatározó
Pin - input → Select kiválasztó érték címkézése
Output?: No
Data Bits: 2
Demultiplexer
Plexers → Demultiplexer
Select Bits: 2

Működés: a kiválasztó bemenetek (Select) segítségével megjelölt kimenet (Y1) felveszi a bemenet értékét (X0).



11.10. ábra. *Demultiplexer használata*

- MUX logisim evolution
- demultiplexer logisim evolution

10.2.1.6. Hétszegmenses kijelző használata

A szimulációs környezetben a hétszegmenses kijelző kiválóan alkalmas számértékek közvetlen megjelenítésére. A beépített alkatrész palettából kétféle változat érhető el, egyik dekódolóval szerelt, a másik dekódoló nélkül kialakított elem. A bináris dekódolót tartalmazó változat a Hex Digit Display elnevezéssel szerepel a listában (11.11. ábra).

Alább tekintsük át a felhasznált alkatrészek fontosabb beállításait.

Jelállapot-meghatározó

Pin - 4 bit:

Output?: No

Data Bits: 4

Jelállapot-meghatározó

Pin - 1 bit:

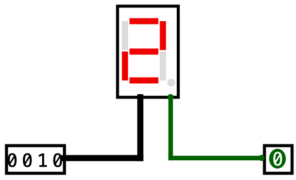
Output?: No

Data Bits: 1

7 szegmenses kijelző

Hex Digit Display:

Működés: a Hex Digit Display alkatrész egy adat- és egy tizedespontkijelzésengedélyező bemenettel rendelkezik. A bemenetére kapcsolt 4 bites értéket hexadecimális formában jeleníti meg. A tizedespont kijelzése az egybites bemeneten megjelenő értéktől függ.

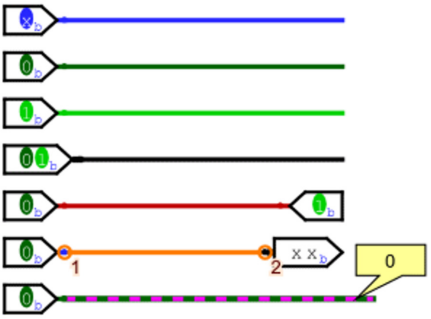


11.11. ábra. *Hétszegmens kijelző alkalmazása*

- 7 segment display logisim evolution
- wire colors logisim evolution

10.2.1.7. Színkódok

Az áramkörök építése és tesztelése során a szimulációs környezet a kapcsolási rajzban szereplő jelvonalaknak és -síneknek, azok állapotának függvényében különböző színeket feleltet meg. Az alábbi, 11.12. ábra mindenik használt színre megjelenít egy példát:



11.12. ábra. *Színkódok jelentése*

Szürke: A vezeték bitszélessége ismeretlen. Ez azért fordul elő, mert a vezeték nincs csatlakoztatva egyetlen komponens bemenetéhez vagy kimenetéhez sem.

Kék: A vezeték egybites értéket hordoz, de semmi sem vezet meghatározott értéket a vezetékre. Ezt hívjuk lebegő bitnek vagy magas impedanciaértéknek (High Z).

Sötétzöld: A vezeték egybites 0-ás értéket hordoz.

Élénkzöld: A vezeték egybites 1-es értéket hordoz.

Fekete: A vezeték több-bites értéket hordoz. Előfordulhat, hogy a bitek egy részének értéke nincs megadva.

Piros: A vezeték hibaértéket hordoz. Ez gyakran azért merül fel, mert egy kapu nem tudja meghatározni a kimenet értékét, talán azért, mert nincs értelmezhető bemenete, de abban az esetben is piros, ha két komponens összekapcsolt kimenete különböző értékeket próbál küldeni ugyanarra a vezetékre. A több-bites vezetékek pirosra váltanak, ha a hordozott bitek bármelyike hibaértéket jelez.

Narancssárga: A vezetékhez csatlakoztatott alkatrészek kimeneteinek bitszélessége nem egyezik. A narancssárga vezeték gyakorlatilag „elszakad”, így nem hordoz értékeket az alkatrészek között. Például egy kétbites összetevőt csatoltunk egy egybites összetevőhöz, így ezek nem kompatibilisek.

Zöld-fukszia: Ez a színséma azt jelzi, hogy a vezetéket a Poke Tool (11.4. ábra a) pontja) segítségével választották ki. A színes kiemelés lehetővé teszi a felhasználó számára az adott jelkapcsolat követését egy összetett sémában, ugyanakkor a vonal értékállapotának jelzése is megjelenik egy kis, sárga állapotjelző buborékban.

10.2.1.8. RAM memória komponens használata

A szimulációs környezet összetett, többfunkciós elemei a memóriatömbök. A RAM (Random Access Memory) típust főként a CPU által végrehajtandó programok és a feldolgozásra váró adatok tárolására használjuk. A fizikai alkatrészben az adatok csak addig maradnak meg, ameddig a számítógép feszültség alatt van, kikapcsoláskor a benne tárolt adatok elvesznek. A szimulációs környezet újraindításakor vagy a szimuláció alapállapotba hozásakor (CTRL+R) a RAM tömb tartalma kiürül, mindig újra be kell olvasni a kezdeti tartalmát meghatározó bináris állományt.

Az alkatrészt a szimulációs környezetben a Memory eszközmappában találjuk meg, RAM elnevezés alatt. Alább tekintsük át a RAM alkatrész működtetéséhez szükséges jelzéseket.

Address Bit Width: 8-8 bites címsín;

Data Bits Width: 8-8 bites adatsín;

A (Address): címbeviteli síncsatlakozó;

WE (Write Enable - Store): ha az értéke 1-re van állítva, akkor a bemeneten lévő érték beíródik a memóriába;

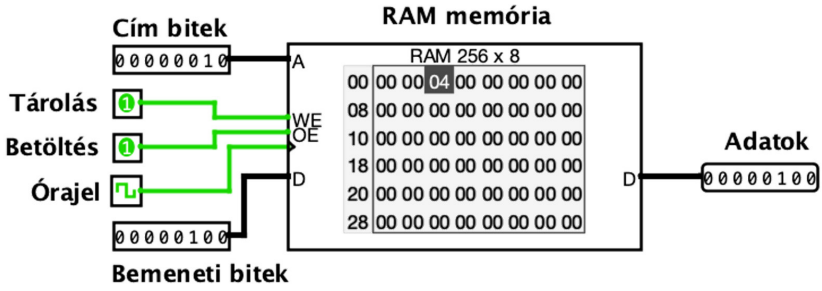
OE (Output Enable - Load): ha az érteke 1-re van állítva, a RAM modul adatkimenete engedélyezett;

Clock: Órajel, beállítás alapján jelszintre vagy élre is aktiválható;

D (Input): bemeneti adatsín;

D (Data): kimeneti adatsín;

Data bus implementation: a beállítás segítségével kiválaszthatjuk, hogy az adatsínt milyen formában szeretnénk megvalósítani: kétirányú adatsínként, vagy külön ki (olvasási) és be (írási) adatsínként használjuk őket;

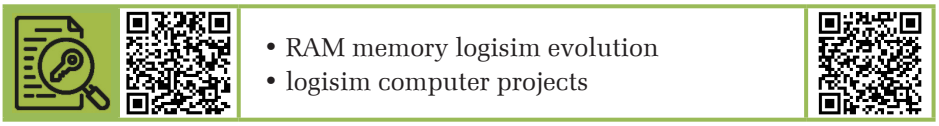


11.13. ábra. *RAM memória modul be- és kimenetei*

A rajzolófelületi megjelenítésben sötéttel kiemelt négyzet jelzi, hogy éppen melyik cella van kiválasztva a címérték alapján. Az egérrel a RAM modulra mutatva jobb klikk-kel előhozható egy menüsor.

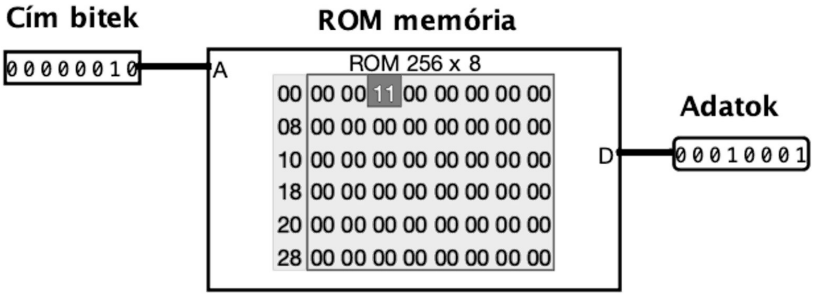
Edit contents: a megjelenő szerkesztőben lehet a RAM értékeit módosítani vagy egy bináris állományt feltölteni.

A 11.13. ábra szerinti kapcsolásban egy címértéket állítunk be a RAM modul címsín bemenetén, majd a következő órajellel szinkronban a memória vagy beolvassa az adott címen lévő adatokat és ezt megjeleníti a kimeneti adatsínen (**we**=0, olvasás funkció), vagy beírja azt az értéket, ami az adatsínen (we=1, írás funkció) van.



10.2.1.9. ROM memória komponens használata

A ROM (Read Only Memory) csak olvasható memóriát jelöl. A ROM tartalma a szimulációs környezetben csak a futtatást megelőzően változtatható, ekkor beírt vagy betöltött adatok véglegesnek számítanak. A szimulációs környezet újraindításakor vagy a szimuláció alapállapotba hozásakor (ctrl+r) a ROM tömb tartalma megmarad.



11.14. ábra. *ROM memória modul be- és kimenetei*

Az alkatrészt a szimulációs környezetben a Memory eszközmappában találjuk meg, ROM elnevezés alatt. Alább tekintsük át a ROM alkatrész működtetéséhez szükséges jelzéseket.

Address Bit Width: 8-8 bites címsínszélesség;
Data Bits Width: 8-8 bites adatsínszélesség;
A (Address): címbeviteli síncsatlakozó;
D (Data): kimeneti adatsín;

A rajzolófelületi megjelenítésben sötéttel kiemelt négyzet jelzi, hogy éppen melyik cella van kiválasztva a címérték alapján. Az egérrel a ROM modulra mutatva jobb klikk-kel előhozható egy menüsor.

Edit contents: a megjelenő szerkesztőben lehet a ROM cellák értekeit módosítani vagy egy bináris állományt feltölteni.

- ROM memory logisim evolution
- Logisim computer projects

10.2.1.10. TTY kijelző és billentyűzetbeviteli eszköz

A **TTY komponens** egyszerű, szöveges terminálként ASCII-kódok sorozatát fogadja (lásd a 11.2. alfejezetet) bemenetként, és minden nyomtatható karaktert megjelenít a kijelzőn. Amikor az aktuálisan írt sor megtelik, a kurzor a következő sorra lép, esetleg az összes aktuális sort felfelé görgeti, ha a kurzor már az alsó sorban volt. A támogatott vezérlőszekvenciák a következők:

backspace (ASCII 0x08h), amely törli az utolsó sor utolsó karakterét, kivéve, ha az utolsó sor már üres;

line feed (ASCII 0x0Ah), amely a kurzort a következő sor elejére mozgatja, szükség esetén görgetve;

form-feed (ASCII 0x0Ch, CTRL+L beírása), amely törli a képernyőt.

Az alkatrészt a szimulációs környezetben az Input/Output mappában találjuk meg, TTY elnevezés alatt. Alább tekintsük át a TTY alkatrész működtetéséhez szükséges jelzéseket.

Data: a következő írandó karakter ASCII értéke;

Clock: az órajel hatására a karakter hozzáadódik a bevitelhez;

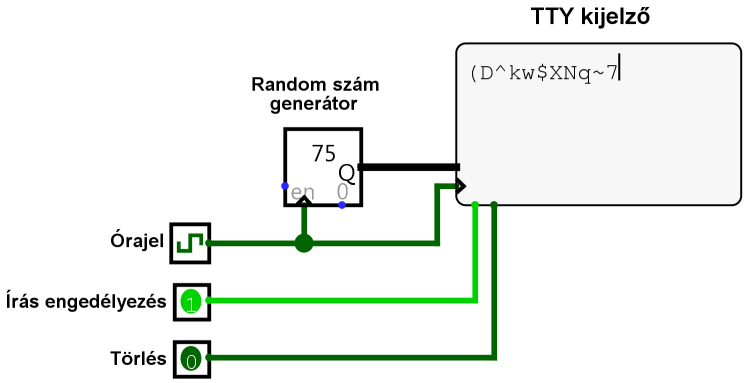
Write enable: a 0 érték tiltja az órajelt;

Clear: az 1 érték törli a képernyő tartalmát;

A 11.15. ábra egyszerű kapcsolási példán keresztül mutatja be a TTY komponens működését. Egy véletlenszám-generátor segítségével tetszőleges értékeket generálunk bemeneti adatként a TTY kijelző számára. Ha az **Írásengedélyezés** jel aktív (1-es), az ASCII kód szerint kijelezhető értékek sorozata megjelenik a kijelzőn. A kapcsolást az órajel-generátor kimeneti jele vezérli a beállított szimulációs frekvencián.

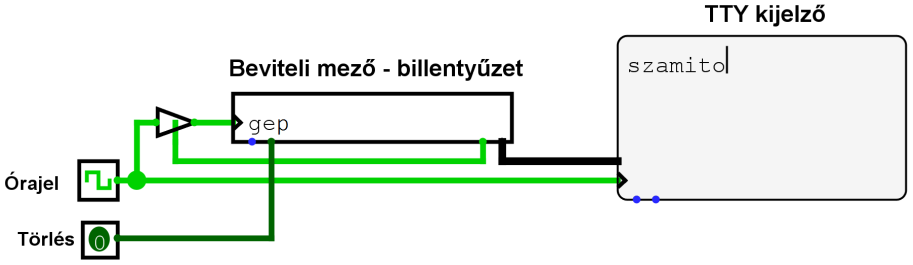
A **billentyűzet** (Keyboard) **komponens** lehetővé teszi, hogy beolvassunk a billentyűzetről begépelt betűkódokat, majd ezeket bináris adat formájában adatsíneken továbbítsuk egy másik komponens felé, amely ezt feldolgozza. Figyelem: csak a 7 bites ASCII kódban (reduced ASCII) reprezentálható billentyűkódok beolvasására van lehetőség.

Az alkatrészt a szimulációs környezetben az Input/Output mappában találjuk meg Keyboard elnevezés alatt. Alább tekintsük át a Keyboard alkatrész működtetéséhez szükséges jelzéseket.



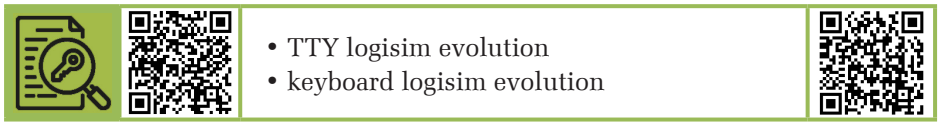
11.15. ábra. *A TTY komponens paraméterei*

Clock: az órajel felfutó élére a bal első karakter a kimenetre kerül;
Read enable: a 0-ás érték tiltja az órajelet;
Clear: az 1-es érték kiüríti a puffert;
Available*:* az értéke 1, amikor a puffer tartalmaz karaktert;
Data*:* a puffer első karakterének ASCII értéke;



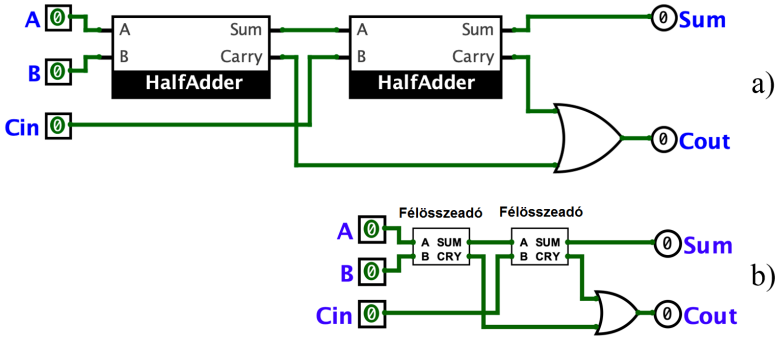
11.16. ábra. *A Keyboard komponens paraméterei*

A 11.16. ábra egyszerű kapcsolásban mutatja be a Keyboard alkatrész alkalmazását. A kapcsolás szimulációját aktiválva először a Poke eszközt kiválasztva, rá kell kattintani a komponensre. Ezután beírhatunk karaktereket, amelyeket az alkatrész beállítható hosszúságú (Buffer Length paraméter) puffere tárol. A „szamitogep” szót begépelve megfigyelhető, hogy amint a komponens órajel bemenete felfutó élet érzékel, a bal szélső karakter eltűnik a pufferből és megjelenik a kimeneten. Példánkban a kimenetet egy TTY kijelzővel kötöttük össze, így a billentyűzetről bevitt karakterek a kijelzőn válnak láthatóvá.



10.2.2. Saját szimbólum készítése

A szimulációs környezet hierarchikus felépítéséből fakadóan lehetőség van saját szimbólumok készítésére és használatára. Egyszerű kapuáramkörökből bonyolult kombinációs vagy szekvenciális áramköröket alkothatunk, majd ezek kapcsolási rajzát megfeleltetjük egy általunk kialakított szimbólumnak, és a továbbiakban ezt használhatjuk. Később lehetőség van a szimbólumokból alkotott áramkör újabb elvonatkoztatására, amikor ezeknek egy újabb szimbólumot feleltetünk meg.



11.17. ábra. *Teljes összeadó áramkör megvalósítása Logisim-Evolution (a) és Custom (b) szimbólumtípust használva*

Alkalmazási példaként egy teljes összeadó áramkört (11.17. ábra) valósítunk meg saját szimbólum segítségével. A teljes összeadóhoz szükségünk van 2 félösszeadóra és egy VAGY kapura. A félösszeadót külön kapcsolási rajzként készítjük elő, majd hozzárendelünk egy szimbólumot. A továbbiakban a teljes összeadó áramkört ezekből a szimbólumokból építjük fel. Miután elkészítettük a félösszeadó kapcsolási rajzát, a fő (main) lapon a félösszeadó nevére kattintva be tudjuk helyezni a félösszeadó szimbólumát a munkafelületünkre. A szimbólum tulajdonságait meghatározó paraméterek közül az **Appearance** segítségével választhatjuk ki, hogy milyen stílusban szeretnénk megjeleníteni a szimbólumot: Classic Logisim, Logisim-HolyCross, Logisim-Evolution vagy Custom.

A **Classic Logisim**, a **Logisim-HolyCross** és a **Logisim-Evolution** szimbólum megjelenítési stílusok esetében előre definiált megjelenítések közül választhatunk. A **Custom** esetben lehetőség van saját szerkesztésre, meghatározhatjuk a

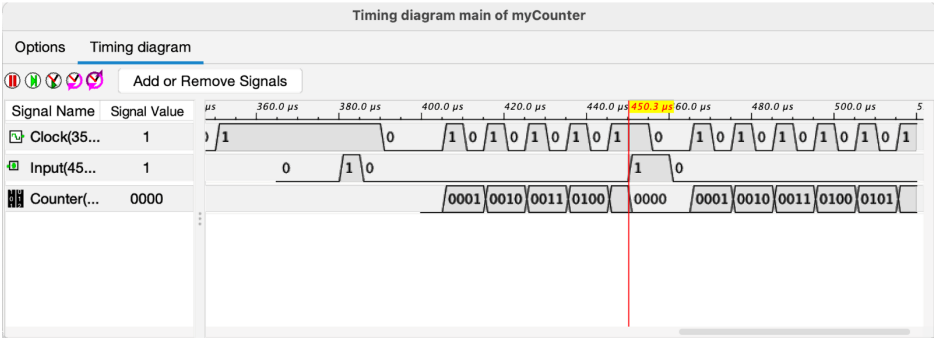
szimbólum formáját (keretét), a be- és kimenetek elhelyezését, feliratok helyét és szövegét. Ilyenkor a felső menüsávban található **Edit viewed circuits appearance** eszközt kiválasztva szerkeszthetjük meg tetszés szerint az áramkör kinézetét.

A 11.17. ábra a) pontjában a félösszeadó szimbóluma az előre definiált Logisim-Evolution típust használja, míg a 11.17. ábra b) pontjának áramköre esetében a Custom típusra láthatunk példát.

10.2.3. Idődiagram (Chronogram) használata

A Logisim-Evolution szimulációs környezetben lehetőség van egy szekvenciális áramkör idődiagramjának (timing diagram/chronogram) megjelenítésére. A kapcsolásban szereplő minden önálló jelforrás feltüntethető az idődiagramon. Ily módon az áramkör jelzéseinek időbeli változásai közvetlenül megfigyelhetővé válnak. Az idődiagram segítségével következtethetünk az áramkör működésére, és hibakeresésre is használhatjuk. A 11.18. ábra egy számlálóból, egy egybites jelhatározóból (bemenetet) és egy órajelforrásból álló, egyszerű áramkör idődiagramját szemlélteti.

Az idődiagramot a **Simulate** menü alatt érhetjük el **Timing diagram** néven. Aktiválásakor egy új ablakban jelenik meg az aktuális áramkör idődiagramja és a hozzá tartozó beállítások (**Options**). Az ábrán a legfelső sorban látható időskála segítségével tudjuk meghatározni, hogy a számláló hány μs ideig működött.

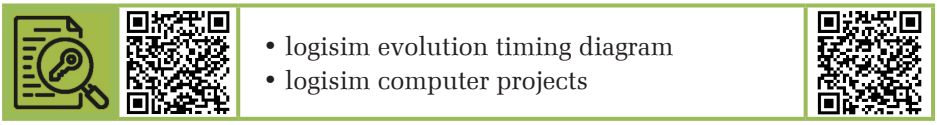


11.18. ábra. *Órajelforrást és bináris számláló kimeneti értékeit megjelenítő idődiagram*

Az idődiagramon megjelenő függőleges, piros vonal bárhol elhelyezhető az idődiagramon, segítségével az adott pillanat értékeit emelhetjük ki a bal oldali panel **Signal Value** oszlopában.

Az **Add or Remove Signal** parancs segítségével hozzáadhatunk vagy eltávolíthatunk jeleket az idődiagramról, de lehetőség van ezek ábrázolási sorrendjének felcserélésére is. A megfigyelő szempontjából legfontosabb jeleket egymás után

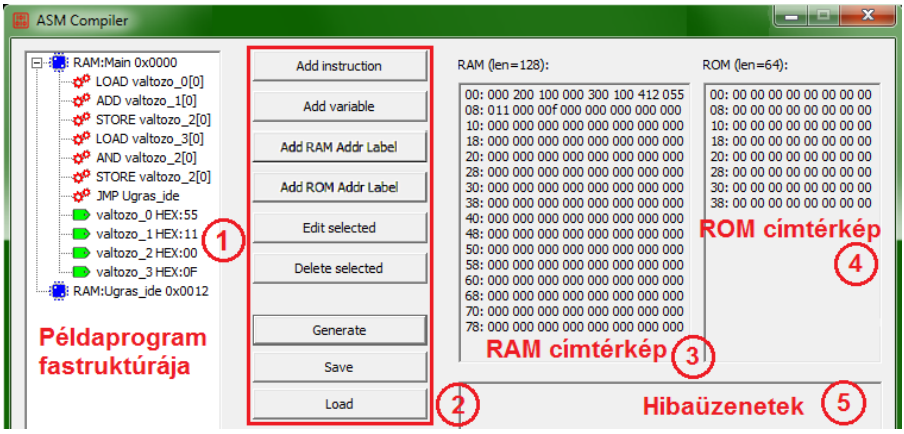
csoportosíthatjuk. Az idődiagram-generáláshoz a **Simulate** menü alatt található **Auto-Tick Frequency** listában megjelenő frekvenciaértékek közül választhatunk, és az **Auto-Tick Enabled** paranccsal tudjuk elindítani a szimulációt.



11.4. Assembler fejlesztőkörnyezet és fordítóprogram

A szimulációs környezetben kialakított számítástechnikai rendszer processzorán assembler nyelven írt, rövid példaprogramokat futtathatunk. Ehhez egy grafikus kezelőfelülettel rendelkező, saját fejlesztésű szerkesztő- és fordítóprogramot használunk (11.19. ábra).

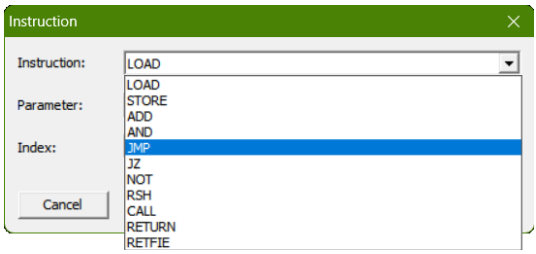
Ebben a fejlesztőkörnyezetben az utasításokat, változókat és kiemelt címeket (memóriacímeket) egy menürendszer segítségével, legördülő listák alkalmazásával visszük be. A bevitt utasítások, változók helye a memóriacímtérben attól függ, hogy grafikusan hol helyezzük el őket a programot megtestesítő fastruktúrában.



11.19. ábra. *Grafikus beviteli felülettel rendelkező assembler programfejlesztő környezet és fordítóprogram*

A fejlesztőkörnyezet kezelőfelületének fontosabb mezőit a 11.19. ábra szerint azonosíthatjuk, az 1-essel jelölt mező a programot megtestesítő fastruktúrát mutatja. Ebben a fastruktúrában szerkesztés közben szabadon mozgathatjuk a már bevitt sorokat is. A 2-essel jelölt mezőben a kezelőgombokat találjuk. Az **Add instruction**

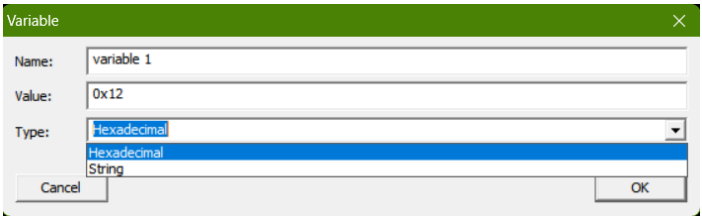
megnyomásával egy felugró ablakot kapunk, amely több legördülő menülistát is tartalmaz, ezek közül a legfontosabb az utasításokat tartalmazó lista (11.20. ábra).



11.20. ábra. *Előre meghatározott utasításkészletből való kiválasztás*

Egyes utasításokhoz (például LOAD, STORE) már előre meghatározott vagy utólag meghatározandó változó címeket rendelhetünk a **Parameter** listából, illetve ha a változó egy karakterlánc lenne, akkor az **Index** mezőben pontosíthatjuk a minket érdeklő elem láncban elfoglalt helyét.

Az **Add variable** megnyomásával egy újabb felugró ablakot kapunk (11.21. ábra), amelyben megadhatjuk egy változó nevét (**Name**), értékét (**Value**), és kiválaszthatjuk a típusát (**Type**), ami vagy hexadecimális szám, vagy karakterlánc lehet.

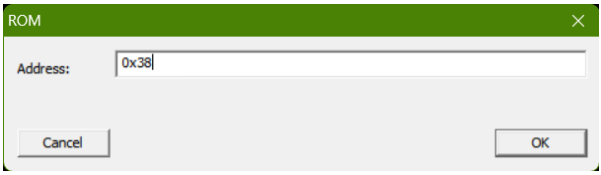


11.21. ábra. *RAM címterületen elhelyezett változó létrehozása*

Az **Add RAM Addr Label** megnyomásával egy újabb felugró ablakot kapunk (11.22. ábra), amelyben megadhatjuk egy függvény kezdőcímét (**Address**) hexadecimális formában, és rögzíthetjük a nevét is (**Function**).

11.22. ábra. *RAM címterületen elhelyezett címke létrehozása*

Az **Add ROM Addr Label** megnyomásával egy másik felugró ablakot kapunk (11.23. ábra), amelyben megadhatjuk egy, a ROM memóriában elhelyezett konstans érték vagy mező kezdőcímét (**Address**) hexadecimális formában.



11.23. ábra. *ROM címterületen elhelyezett címke létrehozása*

Az **Edit selected** megnyomásával a fastruktúrában előzetesen kijelölt sor paramétereihez férünk hozzá, az adott elemnek megfelelő beállítóablakon keresztül. A **Delete selected** megnyomásával a fastruktúrában előzetesen kijelölt sort töröljük.

A **Generate** gomb indítja a kompilálást és fordítást, amely folyamat után a 11.19. ábra 3-assal jelölt mezője, a RAM memória címtérképe feltöltődik a fordításból kapott hexadecimális értékekkel. Ugyanakkor az ábrán 4-essel jelölt mező, a ROM memória címtérképe szintén feltöltődik a fordításból kapott hexadecimális értékekkel. Ugyanakkor létrejön két állomány *.ram és *.rom kiterjesztésekkel. Ezeket az állományokat töltjük be később a szimulációs környezet memóriamoduljaiba. Az 5-össel jelölt mezőben jelennek meg a kompilátor fordítással kapcsolatos hibaüzenetei.

A **Save** és **Load** gombok segítségével tudjuk a megírt programot elmenteni (*.code kiterjesztésű állomány) vagy a már meglévő programot betölteni a fejlesztőkörnyezetbe.

IRODALOMJEGYZÉK

ARATÓ Péter
2004 *Logikai rendszerek tervezése.* Budapest, Műegyetemi Kiadó

BARUCH, Z. F.
2004 *Structura sistemelor de calcul.* Cluj-Napoca, Editura Albastră

BOIAN, Florian–VANCEA Alexandru
2003 *Arhitectura calculatoarelor.* Cluj-Napoca, Universitatea Babeş-Bolyai, Facultatea de Matematică şi Informatică

COHEN, I. B.
2002 *Howard Aiken and the Dawn of the Computer Age*. In the *The First Computers, History and Architectures*, R. Rojas and U. Hashagen, Eds., Cambridge, MIT Press

DUKA A.–RUSU M.–HALLER P.
2005 *Programarea interfeţelor în Labwindows. Îndrumător de laborator.* Târgu Mureş, Universitatea Petru Maior

HALLER Piroska
2021 *Arhitectura sistemelor de calcul.* Târgu Mureş, Editura UMFST

HALSTED, D.
2018 *The Origins of the Architectural Metaphor in Computing: Design and Technology at IBM, 1957–1964*. IEEE Annals of the History of Computing, 40(1), 61–70. doi: 10.1109/mahc.2018.012171268

KANN, Charles W.
2016 *Implementing a One Address CPU in Logisim.* Gettysburg College Open Educational Resources

LEDIN, J.
2020 *Modern Computer Architecture and Organization*. Birmingham, Packt Publishing

MALVINO, Albert P.–BROWN, Jerald A.
1999 *Digital Computer Electronics.* New York, McGraw-Hill Publication

NEUMANN, John Von
1945 *First Draft of a Report on the EDVAC.* Pennsylvania, Moore School of Electrical Engineering, University of Pennsylvania

NISAN, N.–SCHOCKEN, S.
2008 *The elements of computing systems: building a modern computer from first principles.* Cambridge, MIT Press

PAWSON, Richard.
2022 *The Myth of the Harvard Architecture.* IEEE Annals of the History of Computing Print ISSN: 1058-6180 Online ISSN: 1934-1547

RAFIQUZZAMAN, M.
2005 *Digital Logic and Microcomputer Design*. New Yersey, Wiley & Sons Inc. Publication
SELF, George
2019 *Logisim-Evolution Lab Manual*. Edition 4.0, Public Domain
TANENBAUM, Andrew
2006 *Számítógép architektúrák*. Budapest, Panem Könyvkiadó
TIETZE, Ulrich–SCHENK, Christoph
1993 *Analóg és digitális áramkörök*. Budapest, Műszaki Könyvkiadó
TURING, Dermot
2020 *Az Enigma feltörésének igaz története*. Budapest, Typotex Kiadó
TYLAVSKY, Daniel J.
2009 *Digital Design for the Laboratory: Hardware and Simulation (Using Logicworks 5)*. Tempe, CenterPoint Publishing
WANG, Han
2012 *Processor*. Computer Science, Cornell University, https://www.cs.cornell.edu/courses/cs3410/2012sp/lecture/07-cpu-i-g.pdf
WEATHERSPOON, Hakim
2012 *Numbers & Arithmetic*. Computer Science, Cornell University, http://www.cs.cornell.edu/courses/cs3410/2012sp/lecture/03-logic-minimization-and-numbers-i.pdf
WEATHERSPOON, Hakim
2012 *Gates and Logic*. Computer Science, Cornell University, https://www.cs.cornell.edu/courses/cs3410/2012sp/lecture/02-gates-i-g.pdf
WEATHERSPOON, Hakim
2012 *Memory*. Computer Science, Cornell University, https://www.cs.cornell.edu/courses/cs3410/2012sp/lecture/06-memory-i-g.pdf
VELOSO, A.–HUYNH-BAO, T.–MATAGNE, P.
2019 *Nanowire & Nanosheet FETs for Ultra-Scaled, High-Density Logic and Memory Applications*. Solid-State Electronics Published by Elsevier Ltd.
WESTE, H. E. Neil–HARRIS, David Money
2011 *CMOS VLSI Design – A Circuit and System Perspective*. Boston, Addison-Wesley Publishing
WHITE, Ron
2005 *How Computers Work, Fourth Edition*. Indianapolis, QUE a Division of Macmillan Computer Publishing
WISINGER István
2018 *Egy elme az örökkévalóságnak*. Budapest, Athenaeum Kiadó

REZUMAT

Principalul obiectiv al cărţii este de a introduce cititorii în conceptele fundamentale ale arhitecturilor de calculatoare. Pentru a studia capitolele sale, este suficientă o înţelegere de bază a conceptelor fizice şi matematice dobândite în anii de liceu. Ne bazăm pe aceste concepte pentru a oferi explicaţii şi pentru a sublinia necesitatea dezvoltării şi îmbunătăţirii abilităţilor de abstractizare ce ajută la interpretarea funcţionării sistemelor mai complexe.

Începem cu istoria dezvoltării tehnologiei calculatoarelor, urmată de discuţii despre reprezentarea binară, operaţiunile logice şi aritmetice. Prezentăm principiile esenţiale şi etapele tehnologiei şi producţiei semiconductorilor, evidenţiind relaţiile tehnologice dintre implementările circuitelor şi teoria proiectării acestora. Ulterior, explorăm principiile fundamentale ale circuitelor logice simple.

După ce am stabilit bazele teoretice, continuăm să descriem circuitele unui sistem de calcul. Aici devine evident cum este construită şi cum funcţionează o unitate aritmetică-logică sau o memorie, precum şi modalităţile în care sunt realizate stocarea instrucţiunilor şi a datelor, respectiv execuţia liniilor de program. După aceasta, organizăm circuitele sistemului de calcul realizate individual într-o structură mai complexă, dezvăluind modul în care sistemul rezultat funcţionează coordonat de semnalul de tact şi execută instrucţiunile unui program, respectiv susţine fluxul de date aferent execuţiei.

Explorăm, pe scurt, diverse arhitecturi de calculatoare, evidenţiind schimbările care au loc pe măsură ce sistemele evoluează. În cele din urmă, examinăm posibilităţile de accelerare a fluxului de date într-un sistem de calcul cu ajutorul controlorilor de acces direct la memorie şi a controlorilor de întreruperi.

În exerciţiile practice legate de subiectele discutate, investigăm conceptele de bază ale operaţiunilor de flux de date într-un mediu de simulare. Devine posibilă dezvoltarea, compilarea şi rularea propriilor programe pe sistemul de calcul simulat, permiţându-se astfel urmărirea procesului de execuţie al instrucţiunilor individuale până la nivelul porţilor logice elementare.

Sperăm ca cititorii noştri să găsească valoare în paginile cărţii noastre, descoperind detalii interesante care îi vor ajuta să îşi dezvolte şi să îşi extindă cunoştinţele despre tehnologia calculatoarelor.

ABSTRACT

The main objective of our book is to introduce our readers to the fundamental concepts of computer architectures. To study its chapters, a basic understanding of physical and mathematical concepts acquired during high school years is sufficient. We build upon these concepts to provide explanations and emphasize the necessity of developing and enhancing abstraction skills that aid in interpreting the operation of more complex systems.

We begin with the history of the development of computer technology, followed by discussions on binary representation, logical and arithmetic operations. We present the essential principles and steps of semiconductor technology and manufacturing, shedding light on the technological relationships between circuit implementations and theoretical designs. Subsequently, we delve into the fundamental principles of simple logical circuits.

After laying the groundwork, we proceed to describe the circuits of a computer system. It becomes apparent here how an arithmetic-logic unit or memory is constructed and operates, as well as how instruction and data storage and execution scheduling are realized. Following this, we organize the circuits of a computer system into more complex structures, revealing how the resulting system operates coordinated through program execution and data flow.

We briefly explore various computer architectures, highlighting the changes that occur as systems evolve. Finally, we examine the possibilities of accelerating data flow within a computer system using direct memory access controllers and interrupt controllers.

In the practical exercises related to the discussed topics, we investigate the basics of the operation of data flow tools in a simulation environment. It is possible to develop, compile, and run custom programs on the simulated computer system, allowing to trace the execution process of individual instructions, down to the level of elementary logic gates.

We hope that our readers will find value in flipping through the pages of our book, discovering interesting details that will assist them in establishing and expanding their knowledge of computer technology.

A SZERZŐRŐL

Csernáth Géza 1998-ban szerzett villamosmérnöki diplomát ipari automatizálás és informatika szakirányban a marosvásárhelyi Petru Maior Egyetemen. 2008-ban doktori fokozatot szerzett a brassói Transilvania Egyetem Villamosmérnöki és Számítástechnika Tudományok Karán, elektronika és számítástechnika szakágban. 1998-tól 2005-ig villamosmérnökként házi automatizálási rendszerek fejlesztésén dolgozik. Ugyanakkor külön szakterületként felhasználói interfészek tervezésével és programozásával is foglalkozik. 2003-tól társult oktatóként a Sapientia EMTE Marosvásárhelyi Karának Villamosmérnöki Tanszékén a mikrovezérlős rendszerek és számítógép-architektúra tantárgyak előadója, egyetemi diplomadolgozatok témavezetője. 2005-től magánvállalkozóként az elektronikai termékfejlesztés terén tevékenykedik. 2008-tól egyetemi adjunktusként folytatja oktatói tevékenységét a Sapientia EMTE Marosvásárhelyi Karának Villamosmérnöki Tanszékén. 2016-tól ipari teszt- és mérőberendezések tervezésével és kivitelezésével foglalkozik, továbbra is kitekintéssel az ipari formatervezés és grafikus felhasználói interfészek kialakítása felé. A vállalkozói szférában 2019-től műszaki igazgatóként tevékenykedik, feladata az elektronikai berendezésfejlesztések műszaki tartalmának meghatározása és ellenőrzése.

Scientia Kiadó
400112 Kolozsvár (Cluj-Napoca)
Mátyás király (Matei Corvin) u. 4. sz.
Tel./fax: +40-364-401454
E-mail: scientia@kpi.sapientia.ro
www.scientiakiado.ro

Korrektúra:
Szenkovics Enikő

Műszaki szerkesztés:
Metaforma Kft.

Tipográfia:
Könczey Elemér

Sorozatborító:
Tipotéka Kft.

Nyomdai munkálatok:
F&F INTERNATIONAL Kft.
Felelős vezető: Ambrus Enikő igazgató

sapientia
tankönyvek

Számítógépekről, azok felépítéséről, működéséről, felhasználásáról számtalan kötet, tudományos cikk, dokumentációs anyag született az elmúlt évtizedekben. De akár két évszázad távlatába is visszatekinthetünk, ha Charles Babbage írásaira vagy éppen Ada Lovelace tudományos cikkeire vetünk egy pillantást. A számítógépekkel kapcsolatos írások alapjait a működési elveket leíró szövegek, műszaki dokumentumok képezik. Ezeket értelmezik, magyarázzák, teszik közérthetőbbé a témában megjelenő könyvek, cikkek, egyetemi jegyzetek. Jelen kötet is erre tesz kísérletet. Próbálja felidézni a műszaki világ egyik jelentős vívmányának, a számítógép koncepciójának kialakulását, bemutatni a felépítéséhez szükséges technológiákat, logikai áramköröket, magyarázni a működési alapelveket, technikai megoldásokat, beleértve az utasítások végrehajtási módozatát is. A szerző célja az elsőéves egyetemi hallgatók bevezetése a számítógépek felépítésével, mikroprocesszorokkal, memóriaegységekkel, adat- és címsínekkel kapcsolatos fogalmak, műszaki összefüggések világába, utat nyitva ezzel a mélyebb, alaposabb ismeretek, a valódi szaktudás megszerzésének irányába.

ISBN 978-606-975-092-6