Kalmár László: Az elektronikus, digitális számítógépek eddigi fejlődése és a várható fejlődés fő irányai, (1972), 1-117.

A tanulmány áttekinti a számítógépek eddigi fejlődését, külön részletezi a Kalmár-féle formulavezérlésű számítógép alapelveit. A tanulmány a 15 pontos trend-összefoglalóval és a 6 pontos javaslattétellel zárul.

Ezeket a több mint 30 éves gondolatokat érdekes dolog szembesítenünk a mai valósággal, melyek voltak a helyes "előrelátások" és melyek kevésbé helyesek.

AZ ELEKTRONIKUS, DIGITÁLIS SZÁMITOGÉPEK EDDIGI FEJLŐDÉSE ÉS A VÁRHATO FEJLŐDÉS FŐ IRÁNYAI

- Tanulmány -

Szeged, 1972.

Kézirat

A Magyar Tudományos Akadémia Természettudományi I. Főosztályának vezetője, Páris György elvtárs, 1972. május 4-én kelt 3063/23.sz. levelében felkérte Kalmár László akadémikust, az MTA Matematikai Logikai és automataelméleti Tanszéki Kutató Csoportjának vezetőjét, egy összefoglaló tanulmány elkészítésére. A tanulmány feladata végigkövetni a különböző generációju számitógépek jellemzőinek alakulását, a generációváltozás főbb okait, jellemzőit és következményeit, valamint a Kalmár-féle formulavezérlésü számitógéppel kapcsolatos korábbi elgondolásokat.

1972. junius 15-én kelt megbizásaival Kalmár akadémikus az alább felsorolt személyeket vonta be a tanulmány-készitéssel kapcsolatos egyes feladatok megoldásába:

Hunya Péter tudományos munkatárs
(JATE Kibernetikai Laboratórium),
Kertész Ádám müszaki gazdasági tanácsadó
(INFELOR Rendszertechnikai Vállalat),
Quittner Pál kandidátus, igazgatóhelyettes
(Számitástechnikai Oktató Központ),
Sára Attila tudományos munkatárs
(JATE Kibernetikai Laboratórium),
Székely Sándor kandidátus, tudományos főmunkatárs
(JATE Kibernetikai Laboratórium).

1972. augusztus 15-én -- azt követően, hogy Révész György kandidátus nem tudta vállalni a tanulmánykészités munkájában való részvételt -- pótlólag, ugyancsak megbizás alapján, bekapcsolódtak a munkába:

Fodor Dezső tudományos csoportvezető (SzKI), és

Szentes János tudományos munkatárs (SzKI).

A tanulmány Kalmár László irányitásával és közremüködésével készült, figyelembe véve a megbizott személyek részstanulmányait. A tanulmány előzetes szerkesztői munkáját Székely Sándor, a végleges alakba való szerkesztést Kalmár László végezte.

BEVEZETÉS

A tudományos-technikai forradalom egyik legsajátosabb és alapvető jelentőségre szert tett fejleménye a számítástudomány és számitástechnika rendkivül gyors és nagy méretü térhóditása, amely szoros kapcsolatban van az emberi tevékenység mind több területét átfogó automatizálási folyamattal. Mind az anyagi javak termelésében, mind a szellemi munka legkülönfélébb területein mélyreható változások lehetőségei bontakoztak ki azzal, hogy a tevékenység csaknem valamennyi formája közvetlen technikai bázist kap. Érlelődnek a feltételei annak, hogy az ember kiváljon a közvetlen technológiai folyamatokból, sőt a szellemi munka körébe sorolt számos tevékenységét is gépekre ruházza át.

A számitástechnika legfőbb eszközei az elektronikus digitális számitógépek, amelyeknek megjelenése érett feltételek és parancsoló szükségletek rendkivül erős együtt-jelentkezéséhez kötődik. Ami a feltételeket illeti: a matematika és logika itt hatékony apparátusának, valamint a fizika alkalmazható elveinek a fejlettsége, az elektronikus technológia meglévő eredményeivel együtt, szilárd alapot biztositottak a számitógépek fejlődéséhez. A szükségletek oldalán viszont az információk rendkivül hatékony feldolgozásának társadalmi igénye jelentkezett, amely egyrészt merőben ujtipusu feladatokkal (szupergyors folyamatok vezérlése, a világürkutatás számos információfeldolgozást követelő részlete), másrészt (és méginkább) azzal függött össze, hogy a tudományos-müszaki, valamint a termelésirányitási tevékenység kinőtte a korábbi szervezeti kereteket, s oly mértékben vált bonyolulttá, hogy csak minőségileg magasabb szinten megvalósitott információs és szervezési tevékenységgel lehetett urrá lenni rajta. Nem lebecsülendő hatást gyakoroltak a katonai területekről származó számitási és egyéb információfeldolgozási igények sem. Mindezek hatására az 1940-es évek második felében kialakul, majd gyors fejlődésnek indul a számitógépek irányzata.

A szükségletek nagy társadalmi erőforrások mozgósitását váltották ki, amelyek páratlan ütemben lenditették előre az elektronikus számitógépekkel kapcsolatos valamennyi területen folyó kutató- és fejlesztőmunkát. A kezdetben csak néhány irányra szükült fejlődés mind sokrétübbé vált és számos irányban elágazott. Eközben a dialektikus fejlődés minden jellemvonását magára öltötte. Különösen szembetünőek a fejlődési folyamatokban bekövetkezett minőségi ugrások, valamint a

nyiltan felszinre bukkanó ellentmondások, amelyek megoldódása mindig ujabbak keletkezését vonta maga után, s ezáltal a fejlődési tendenciák legfőbb belső ösztönzőivé váltak. A fejlődési folyamat elemzése éppen ezért filozófiai kategóriák igénybevételét is szükségessé tette.

A tanulmánykészités munkálatai közben számos nehézség is akadt. Közülük egyről már itt a bevezetésben szólni kell. A vonatkozó iroda-

lom áttekintése meggyőzően mutatta -- és erről sok szerző is nyilatkozik --, hogy számos terminológiai, fogalom-meghatározási és -értelmezési eltérés van, ami zavarokhoz vezethet egy általános értékelés esetén. Emiatt a tanulmány szerzői igyekeztek a használatos legfontosabb fogalmakat pontosan értelmezni és ennek az értelmezésnek megfelelően felhasználni mondanivalójuk kifejtésénél.

Szem előtt tartva a tanulmány-készités fő szempontjait, a következő kérdésekkel foglalkozunk:

Vázlatosan áttekintjük az egyes számitógép-generációk főbb jellegzetességeit és a generációváltozásokat előidéző okokat, beleértve a fejlődési folyamatban jelentkező legfontosabb kölcsönhatásokat, ellentmondásokat stb. Ez teremt alapot egy viszonylag egységes koncepció kialakitásához a negyedik generációs osztály jellemzése szempontjából (1. fejezet).

Részletesen végigkövetjük a számitógépekkel kapcsolatos fejlődés hardware és software vonalait, ahol lehetőség nyilik számos részletkérdés és a fő tendenciák szempontjából viszonylag alárendelt jelentőségü folyamat elemzésére is (2. fejezet).

Részletesen ismertetjük a Kalmár-féle formula-vezérlésü számitógépre vonatkozó elgondolásokat, beleértve a gép strukturájának és müködésmódjainak részletekbe menő jellemzését is. (3. fejezet).

Az előzetes anyag és néhány prognózis alapján következtetéseket vonunk le a fejlődési folyamat várható további irányaira nézve (4. fejezet).

Végül javaslatokat teszünk hazai fejlesztési feladatokra (5. fejezet).

1. A SZÁMITOGÉP-GENERÁCIOK ÁLTALÁNOS JELLEMZÉSE

1.1. A számitógépek generációs osztályozása

A számitógépek fejlődési folyamatának áttekintését jelentősen megkönnyiti az un. generációs osztályozás. Egy-egy generációba általában azokat a számitógépeket sorolják, amelyek egymástól (különféle szempontokból) minőségileg eltérő fejlődési periódusokban jelentek meg tömegesen. Minthogy a generációs osztályozás a fejlődési szakaszok figyelembevételén alapul, alkalmas a számitógépek fejlettségének jellemzésére. Ha pedig az egyes generációk átmeneteit vizsgáljuk, akkor a rendkivül gyors és bonyolult fejlődésben bekövetkezett minőségi ugrások elemzése is lehetségessé válik.

Az elektronikus, digitális számitógépek eddigi fejlődéstörténetében három számitógép-generációt különböztetnek meg (1., 2. és 3. generáció). Szokás ujabban 0. generációról is beszélni, amelybe az 1940-es évek közepe előtt kifejlesztett (mechanikus és elektromechanikus) számológépeket sorolják, valamint 4. generációról, amellyel viszont a közeljövőben várható, minőségileg uj fejlődési szakaszt jelölik meg.

A jelenlegi generációs osztályozás (amely az 1., 2. és 3. generációs gépek vizsgálatára épül) a hardware-fejlődést (mindenek előtt pedig az egyes gépekben alkalmazott elem-bázis milyenségét) követi nyomon. Ennek az a magyarázata, hogy a számitógépek kialakulását követő jelentős időszakon keresztül a hardware tulajdonságok voltak egy gép legfontosabb jellemzői.

A számitógépek fejlesztésében érvényesülő korábbi fő tendencia a működési sebesség és a megbizhatóság gyors ütemü növelése volt, amit az 1960-as évek második feléig csaknem kizárólag az elem-bázis függvényeként lehetett tekinteni. (Emellett természetesen nem szabad figyelmen kivül hagyni számos más fejlődési tendencia kialakulását és érvényesülését sem, amelyeket gyakran éppen az ujabb elemek által teremtett lehetőségek hivtak életre).

A generációs osztályozás legfontosabb szempontjává az emlitett okok miatt az elem-bázis milyensége vált. Mivel pedig az elektronikus számitógépekben az eddigiekben három alapvető elem-tipust használtak: elektroncsövet, tranzisztort és integrált áramköröket (közülük az első kettő kapcsolóelem, a harmadik viszont kapcsolóelemeket tartalmazó bonyolultabb funkcionális elem), ezért — ezekhez kapcsolódóan — három generációt különböztetnek meg. (Egyébként ezen az alapon terjesztik ki az osztályozást az 1940-es évek

közepe előtt kifejlesztett és jelenleg is széleskörűen alkalmazott számológépekre, ezek "kapcsolóelemei" mechanikus és elektromechanikus elemek). Az egyes generációk keletkezését ahhoz az időponthoz szokták kötni, amikor az adott elemtipussal összefüggő, aktuálisan élenjáró fejlesztést már bevezették a gyártásba. Tehát az első

generációs gépek jellegzetes időszaka: az 1950-es évek, a második generációsoké az 1960-as évek első fele, a harmadik generációsoké az 1960-as évek második fele.

Az egyes számítógép-generációknál azonban az elem-bázis értékelésén túlmenően számos más jellemzőt is célszerű figyelembe venni, amelyek egyrészt lehetővé teszik az egyes generációkon belül végbemenő fejlődés árnyaltabb megmutatását, másrészt olyan egyéb fejlődési tendenciák közül a legfontosabbakat (azok kezdeti jelentkezését és kibontakozását) emelik ki, amelyek alapján sokkal gazdagabban rajzolható meg az általános fejlődési folyamat. Ezek körébe egyéb hardware-jellemzők, feldolgozási módok, software-fejlesztések, valamint alkalmazási területek tartoznak. (A következő fejezetben részletesen végigkövetjük majd a fejlődés menetét, az említett jellemzők kialakulásának okait és a közöttük érvényesülő kölcsönhatásokat, most csupán azokra a jelenségekre és fontosabb tendenciákra leszünk tekintettel, amelyek az egyes generációkban már széleskörűen érvényesültek.)

1.2. Az 1., 2. és 3. generációs gépek jellemzése

Az első generációs számítógépek megjelenése minőségi változást hozott a számolóberendezések egész megelőző fejlődéséhez képest. Ez mindenekelőtt két uj fejleményben jut kifejezésre: a működési sebesség nagyságrendekkel való megnövekedésében és a magasfoku automatizáltsági szintben. Az előbbi az elektronikus kapcsolóelemek (esetünkben elektroncsövek) alkalmazásával függ össze, amelyek több nagyságrenddel növelték meg a müveletek végrehajtásának sebességét (az első generációs számítógépek müveleti sebessége 10^3–10^4 művelet/sec nagyságrendben mozog). Az utóbbit a programtárolás elvének realizálása eredményezi, ami lehetővé tette, hogy a végrehajtandó müveletek rendjét meghatározó információkat is magában a számítógépben tároljék, és a gép tárolt utasitások vezérlése alatt működjék. A kettő viszonya abban áll, hogy a programtárolás, a működési sebesség nagyságrendekkel való megnövekedése, mint mennyiségi változás által szükségképpen kiváltott minőségi változás, a mechanikus számológépeknél általánosan alkalmazott egyenkénti utasitás-adással szemben. Ugyanis a működési sebesség növekedésével elért időnyereséget nem lehetne kihasználni, ha egy-egy müvelet végrehajtása után nagyságrendekkel nagyobb időt venne igénybe az utasitás adása a következő müvelet végrehajtására.

Az első generációs számítógépek alapstrukturája a (számítógépek történetében meghatározó szerepet játszó) Neumann-Goldstine tanulmány (1946-48) [5], [1] által rögzitett elvekhez igazodik, vagyis öt alapvető funkcionális egység (be- és kimenő egység, memória, aritmetikai egység, vezérlőegység) szintézise. A felépitès rendszertechnikailag processzorcentrikus, ami a soros feldolgozás lehetőségét biztositja. A perifériás egységekre (ideértve a háttér-tárolót is, amely általában mágnesszalagos berendezés) jellemző, hogy kizárólag az adott géphez készülnek.

Az első generációs gépekhez kapcsolódó programozási eszközök és software-elemek a következők:

- gépi kód, amely pontosan követi a számítógépek alapvetően bináris jellegét, tehát azt, hogy csak binárisan kódolt információk értelmezésére és tárolására alkalmasak; ezért a gépi kódoknak kettes számrendszerbeli számjegyek sorozatai felelnek meg (ami nehézkessé és bonyolulttá teszi a számítógépek használatát); majd
- assembly szintü gépi nyelv, amely már megengedi szimbólu-

utasitás egy gépi utasitásnak felel meg (ennek eredményeként kezdenek elterjedni az alfanumerikus input/output berendezések); és az ezeknek megfelelő

- fordító program (ez adott esetben assembler), amely gépi kódban irt és általában a gép memóriájában elhelyezett program az assembly szintü gépi nyelven irt feladat-program leforditására gépi kódban.

Az 1950-es évek elejétől kezdődően kidolgozzák a szubrutinok módszerét, megjelennek és széleskörüen elterjednek a programkönyvtárak. Ez a fejlemény azt jelentette, hogy a programozónak módjában állt kiválasztani és az általa készitett feladat-főprogramba beilleszteni a rendelkezésre álló programrészleteket, szubrutinokat, akár a gép által (pl. a szubrutin paraméterei aktuális értékeinek figyelembevételével) módositott formában is. Ez egyuttal vissza-hatott a hardware-fejlődésre is, amennyiben épp az ilyen módositás megkönnyitésére létrejöttek az indexregiszteres gépek is. A szubrutin-módszer -- a programkönyvtárak folyamatos bővitésével -- az adott gép, illetve programrendszer mind szélesebb körü felhasználhatóságát eredményezte, emellett a későbbi fejlődés során tért hóditó szegmentációs technika előfutárának is tekinthető.

Az első generációs számitógépek főbb alkalmazási területe: a katonai, majd a tudományos-müszaki számitások végzése volt.

A gépek elterjedését, költséges voltuk mellett, nehézkes kezelhetőségük is gátolta, ami fokozott követelményeket támasztott nemcsak a hardware-, hanem a software-fejlesztések iránt is. Ezek hamarosan a magasabb szintü programozási nyelvek kialakitásának szükségletében és fejlesztésében összpontosultak, ami a programozási tevékenység gyors fejlődését váltotta ki. Sürgető igény a müködési sebesség és a megbizhatóság (nagyságrendekkel való) növelése is, aminek leginkább járható utja az 1950-es években, a gyorsabb és megbizhatóbb müködésü kapcsolóelemek kifejlesztése és alkalmazása volt. Ezek a törekvések vezetnek el a második generációs számitógépek megjelenéséhez.

A második generációs gépekben kapcsolóelemek gyanánt félvezetőket és tranzisztorokat használtak, ami ujabb minőségi ugrást eredményezett a számitógépek fejlődésében.

A gépek alapstrukturája követi az első generációs számitógépeket, de megnövekszik a tárolók szerepe és kialakulnak a későbbi memóriacentrikus felépitési csirái. Az operativ tárolókban egyeduralkodóvá válik az 1954-es első alkalmazástól kezdve folyamatosan terjedő ferritgyürüs megoldás. A háttér-tárolók között a soros hozzáférésü mágnesszalagos és az ezeknél rövidebb hozzáférésü idővel rendelkező, addig operativ tárolónak használt mágnesdobos berendezések mellett megjelennek a közvetlen hozzáférésü mágneslemez-tárolók, amelyek használata, bár költségesebb, de jelentősen csökkenti a külső tárolók hozzáférési idejét. Az uj kapcsolóelemek, valamint a tárolókban végrehajtott változtató-

sok mintegy két nagyságrenddel növelik meg a számítógépek müveleti sebességét és kedvezően befolyásolják a megbizhatóságot.

A második generációs gépek fejlődési időszakában jelentősen megnövekszik a software aránya és a számitástechnikai tevékenységben játszott szerepe (amihez mind lényegesebben járulnak hozzá az alkalmazási területekről származó igények.) Gyorsan fejlődnek és széleskörü felhasználásra találnak a magasabb szintü programozási nyelvek (FORTRAN, COBOL, ALGOL), amelyek a feladatok programjának leirását az emberéhez lényegesen közelebb álló nyelven teszik lehetővé. Használatuk azonban egyre több kiegészitő tevékenység ellátását követeli meg a számitógéptől. Ugyanakkor viszont a megnövekedett müködési sebesség elfogadhatóvá csökkenti a magasabb szintü programnyelven irt feladatnak (a forditás okozta hatékonyságcsökkenés miatt) lényegesen megnövekedett gépi idejét, a nagyobb tároló kapacitás pedig lehetővé teszi az egyre bonyolultabbá váló programozási nyelvek miatt szükségessé váló mind terjedelmesebb forditóprogramok (ezuttal már általában compilerek) használatát. A kiegészitő tevékenységnek a gépre való átruházására irányuló tendencia vezet el az operációs rendszerek kezdeti változatainak kialakulásához. Ugyancsak ehhez a fejlődéshez tartozik az, hogy rutinok készülnek az I/O tevékenység megkönnyitésére, majd megjelennek az első I/O rendszerek.

A gépek megnövekedett müködési sebessége -- ami lényegesen megemeli a gépi idő értékességét -- változtatásokat indit el a feldolgozási módokban is. A gépet, hogy kellőképpen kihasználják a benne rejlő kapacitást, mind célszerübb folyamatos üzemmódban müködtetni, ami hamarosan elvezet a feladatok kötegelt feldolgozásának kialakulásához. Ennek az a legegyszerübb változata, hogy a gép befejezett feladatnál megállás nélkül tér át -- a rendelkezésre álló -- soronlévő feladatra. Másrészt (szoros kölcsönhatásban a kötegelt feldolgozási móddal) már ekkor megjelennek a megszakitási rendszer csirái, és ezzel lehetőség nyilik egyrészt a soronlévő feladat kiválasztásában prioritási szempontok érvényesitésére, másrészt elemi I/O tevékenységek átlapolására számitási müveletekkel, ami ismét ugrásszerüen javitja a gép effektivitását. Ez utóbbi fejleményben vehetjük észre egy később rendkivül fontossá váló fejlődési tendencia (ti. a többszörös hozzáférésü időosztásos rendszerek) csiráját.

A második generációs gépekkel kapcsolatos software-nek további jellegzetes vonása az egy programozási nyelvre orientált vezérlőprogram, valamint az egy-ember egy-gép kapcsolat biztositása.

A gépek alkalmazási területe jelentős mértékben kibővül, az eddig is előtérben lévő tudományos-müszaki számitások elvégzése mellett egy sokkal szélesebb alkalmazási terület nyilik: a kereskedelmi és általában az üzleti élet. Ez a változás egyrészt tömegessé teszi a számitógép-termelést, másrészt jelentős hatást gyakorol a programozási nyel-

vek fejlődésére, ti. elinditja az adatfeldolgozó és listafeldolgozási nyelvek megteremtésének irányzatát. Ugyanakkor viszont a számitógépek

elterjedéséhez javitani kell kezelési módjaikon, sokkal könnyebb és kényelmesebb hozzáférést kell biztositani a gépekhez. Ez még fokozottabb mértékben állitja előtérbe az ember-gép kapcsolat optimálisabb módjainak kutatását, mind a hardware-, mind a software eszközök fejlesztésének tekintetében.

A tudományos-müszaki feladatok egy része (pl. a világürkutatás számitástechnikai oldala) további sebességnövelési törekvéseket erősit a gépek processzorainak müködése és memória-kezelése tekintetében. Még erősebb ösztönzést jelentett a müködési sebesség növelésére az operációs rendszerek térhóditása. Annak ellenértékeként ugyanis, hogy a számitógép lényegesen nagyobb részt vállaljon a feladatok megoldásában, számos — nem közvetlenül a tárgy-problémához tartozó -- információfeldolgozást (pl. az egyes feladatok állapotjellemzőinek nyilvántartását) is el kell látnia. A hatékonysági mutatók korábbi szinten való tartása, illetve további javitása emiatt csak a gép müködési sebességének jelentős növelése mellett lehetséges. A sebesség-növelésre még mindig nagy lehetőségek rejlenek uj tipusu elem-bázis kifejlesztésében, amely optimálisabban aknázná ki az elektromágneses információhordozó terjedési sebességét. Ugyanakkor mind erősebbé váló ellentmondás lép fel a központi egység és a perifériás berendezések müködési sebessége között, amit még tovább éleznek a gyorsabb müködésü alapelemek irányába mutató fejlesztések. A müködési sebesség növelésére és ugyanakkor az emlitett ellentmondás áthidalására a harmadik generációs gépek jelentenek megoldást.

A harmadik generációs gépek jellegzetes alapelemei az integrált áramkörök, amelyek maguk is egy külön fejlődési vonalat képviselnek. Az alacsony integráltsági foku áramköröktől (SSI), a közepeseken (MSI) keresztül, a magas integráltsági foku áramkörökig (LSI), halad a fejlődés, sőt jelenleg már a szuperintegrált áramkörök (CSI-10^4 komponens/tok) kifejlesztésén dolgoznak. (A harmadik generációs gépekben az SSI és az MSI elemeket használták széleskörüen).

A harmadik generációs gépek alapstrukturája lényeges változásokat mutat a megelőzőkhöz képest. Átalakulnak a gép alapvető funkcionális egységei és azok a szervezési módok is, amelyekkel az egységek rendszerbe foglalódnak. Az uj funkcionális egységek részint a korábbi gépek egységeinek részleges "egymásbaolvadásából" alakulnak ki (pl. a gyors, kis kapacitásu előtárolóval ellátott, párhuzamos müködésü aritmetikai-logikai egység), részint teljesen ujak (pl. az autonóm csatornák). Különösen jelentős változásokat idéz elő az autonóm csatornák alkalmazása, ami széleskörü hardware-lehetőségeket teremt számos tevékenység párhuzamos végzéséhez.

A harmadik generációs számitógépek felépitése rendszertechnikailag memóriacentrikus, vagyis az egyes funkcionális egységekkel a köz-

ponti tároló közvetlen kommunikációs kapcsolatban van. A jobb kihasználás érdekében érvényesül a modularitásra való törekvés (ez teszi

lehetővé tulajdonképpen, hogy pl. a perifériák az autonóm csatornák segitségével a központi processzortól függetlenül is igénybe vehessék a memória valamely "szabad" modulját). A perifériás berendezések széles és variálható skálája alakul ki (köztük megjelenik a jövőbeli fejlődés, elsősorban az ember-gép-kapcsolat további fejlesztése szempontjából igen jelentősnek tartott display). Általánossá válnak a mind nagyobb kapacitásu, közvetlen hozzáférésü és cserélhető mágnes-tárak.

Érvényre jut a perifériás berendezések szabványositásának irányzata, ami változásokat eredményez a számitógép-gyártás ipari bázisának szervezésében.

Az autonóm csatornák alkalmazása, a modularitási törekvések és a szabványositott, cserélhető perifériás berendezések kifejlesztése megköveteli a szabványos illesztés (standard interface) hardware-feltételeinek kidolgozását és biztositását, ami a gépi konfigurációk rugalmas változtathatóságát és az egyes felhasználási területekre való specifikálhatóságát eredményezi.

Lényeges változáson megy át a teljesitmény fogalma (pl. igen fontossá válik a csatornák áteresztő-képessége). A teljesitmény-mutatók növelésére uj módszerek alakulnak ki (pl. több processzoros rendszerek, cache-memória, scratch-pad memória stb. alkalmazása), amelyekkel a gép müveletvégző (vagy általánosabban feladatvégrehajtó) sebessége -- fontos szervezési feladatok megoldása révén -- változatlan müködési sebességü alapelemek mellett is növelhető. A harmadik generációs gépek müködési sebessége 10^6 -- 10^7 müvelet/sec nagyságrendben mozog (néhány gépnél pedig elérték a 10^8 müvelet/sec tartományt).

Az a mennyiségi növekedés, amely a számitógépek müködési sebességében bekövetkezett, a feladat-feldolgozási folyamat minőségi átalakulását eredményezte. A megnövelt müködési sebesség ugyanis mindenek előtt a gép központi egységeire (processzor és operativ tároló) vonatkozik. Bár a különféle (tömegesen alkalmazott) perifériás berendezések teljesitménymutatóit is sikerült lényegesen megjavitani, mégis ezek messze elmaradnak a központi egységek müködési sebessége mögött, sőt a különbség állandóan növekszik. Ennek az az oka, hogy a legtöbb perifériás berendezés müködési sebességének fizikai (és sok esetben racionális) korlátai lényegesen alacsonyabban huzódnak, mint a központi egységeké. Emiatt nem volt várható a különbség jelentős csökkentése azon az uton, hogy megemelik a perifériás berendezések müködési sebességét. Ha egybevetjük a feladatok korábban alkalmazott soros feldolgozási módját az ujonnan kialakult helyzettel, akkor világos, hogy a központi egységek kényszerü állásidői más egységek (lassubb) müködése közben, türhetetlen kapacitáskiesést jelentenének. Ez azzal a következménnyel járna, hogy (speciális feladatoktól eltekintve) a központi egységek a munkaidő legnagyobb részében nem dolgoznának, mivel éppen a nagy müködési se-

Az ellentmondás feloldására irányuló törekvések a párhuzamos működésmódok tökéletesitésének és az egész gépi tevékenységre történő kiterjesztésének tendenciáját váltják ki. Világosan megfogalmazódik az a cél, hogy biztositani kell a gép valamennyi funkcionális egységében rejlő kapacitós lehetőleg teljes kihasználását. Ez részben ujabb hardware-fejlesztéseket sürget (pl. megszakitási rendszerek hardware feltételeinek megteremtése, terminálok és többprocesszoros rendszerek kifejlesztése), részben pedig minőségileg magasabb szintre emeli a software-rel -- és mindenekelőtt az operációs rendszerekkel — szemben támasztott követelményeket. (Jellemző, hogy a harmadik generációs számitógépeknél a software kidolgozási költségei elérik a hardware előállitási költségeinek legalább az 50 %.-át).

A szóbanforgó ellentmondások feloldásának igénye hivja életre az időosztásos rendszereket, amelyek először software-módszerekkel realizálódnak. Történetileg a multiprogramozás [3] az első forma, amelynél, ha a központi egység valamilyen ok miatt egy program futtatását szüneteltetni kényszerül, a várakozási időben áttér egy soronlévő program futtatására. A multiprogramozás kétségtelenül alkalmas eljárás a gépi kapacitás kihasználásának megjavitására, de nem mentes ujabb ellentmondásoktól. Ilyen pl. az, hogy egyes feladatokra (kedvezőtlen körülmények lehetséges egybeesésénél) csak nagyon hosszu idő elteltével kerül sor, ami korlátozza, sőt egyes esetekben felhasználói szempontból értelmetlenné teszi a gép igénybevételét. Az ujabb ellentmondás feloldása lehetséges oly módon, hogy hardware-eszközzel (timer) gondoskodnak a központi egység tevékenységének szétosztásáról különböző feladatok között (idő-szeletelés). Ez lehetővé teszi azt, hogy több felhasználó (terminálok segitségével) egyidejüleg vegye igénybe a számitógépet, sőt (egybevetve a nagy működési sebességgel) real-time üzemmódban használja fel. Ezzel a fejleménnyel nagy távlatok nyilnak meg a többszörös hozzáférésü rendszerek kifejlesztésének irányában.

Ami a software-rel szemben támasztott igényeket, valamint a korábban már emlitett módszereket illeti, rendkivül lényeges, hogy megváltozik a feladat-feldolgozás szervezése és vezérlése. A korábban egységes feldolgozási folyamat szakaszokra bomlik (pl. forditás-szerkesztés, hibajavitás, futtatás). A végrehajtás feletti felügyeletet, az ujabb szakaszokra való áttérések vezérlése stb. célszerüen ismét csak a gép feladata kell, hogy legyen. (A gyakori emberi beavatkozás ugyanis lényegesen lelassitaná a feldolgozási folyamatot, s elveszne a részfeladatok végrehajtásánál nyert idő.) Ez vezérlő és szervező programok (általában fejlett operációs rendszer) meglétét igényli, s egyben lehetőséget teremt a feladatok párhuzamos feldolgozására is, ami pontosan illeszkedik a gépek párhuzamos működésmódjaihoz.

A felsorolt feldolgozási módok (amelyeket a második fejezetben részletesen is megtárgyalunk), a legváltozatosabb kombinációkban kerülnek alkalmazásra a harmadik generációs gépeknél.

Jól észrevehető, hogy ebben a bonyolult és sokirányuvá váló fejlődésben mind erőteljesebben érvényesül a hardware és software dialektikus kölcsönhatása, amire már a megelőző időszakokban is számos példa volt. Lényegesen uj vonás (amely az 1960 -as évek végén vált nyilvánvalóvá) a következő: a hardware és a software olyan szerves egységet alkotnak, hogy a (fejlett) harmadik generációs rendszerek már csak egységükkel jellemezhetők. Ennek nem csupán az az oka, hogy ugyanazon hardware-hez különböző software-rek készithetők, hanem, hogy a feladatok legkedvezőbb gépi elvégzését az ember és a gép eltérő adottságainak megfelelő munkamegosztás optimalizálása eredményezheti, s ennek a software a közvetitője. A hardware és software közötti kölcsönhatás igen gyakori megnyilvánulása a "hardware software-esitése" (pl.gépi utasitásoknak software eszközökkel való definiálása, ami a mikroprogramozáshoz vezetett, vagy az autonóm csatornák csatornaprogramokkal való vezérlése stb.) és ennek forditottja (pl. indexelési lehetőségek hardware eszközeinek megteremtése, a felhasználók on-line kapcsolatának eszközei stb.).

Külön említést érdemel a software fejlődés (az előbb említetteken tulmenő) néhány sajátossága. Ilyenek pl. a több felhasználó egy gép kapcsolat biztositása software eszközökkel, az a tendencia, hogy a software- és a hardware-struktura elkülönülnek egymástól (a software csak részben függvénye a hardware-nek), ami a hajlékony és önálló fejlődésre képes software kialakulásához vezet. Ez utóbbi sajátosság lehetőséget nyujt arra, hogy a harmadik generációs gépek fejlődésének időszakában ne csak az alkalmazott, hanem az alap software-kutatások is széleskörü fejlődésnek induljanak. Ennek során önálló fejlesztési tendenciák erősödtek meg (amelyek kezdetlegesebb formában az előző generációknál is léteztek): a programozási rendszerek, a kezelőrendszerek és a programozási nyelvek.

Jelentős változás következik be a harmadik generációs gépek alkalmazási területein is. Leglényegesebb vonás, hogy egyesül az univerzális gép két alkalmazási iránya: a tudományos müszaki feladatok elvégzésére való felhasználás és a kereskedelmi-üzleti alkalmazás. Ennek egyik következménye az információk szervezésének átalakulása (pl. a byte-szervezés). Uj alkalmazási területek alakulnak ki, amelyek igényeinek kielégitése (néhány megvalósitott kezdeti megoldástól eltekintve) jövőbeli feladat. Ezt a célt szolgálják a legfontosabb alkalmazási területeken tervbevett tervezés-automatizálási, tudakozó-informativ, valamint technológiai és gazdasági folyamatirányitó rendszerek.

Az univerzális számitógépekkel eredményesen kezelhető feladatok körének lényeges kibővitését jelentené a harmadik generációs gépeknél kezdeményezett dialógus üzemmód, amelynek rendkivül nagy jövőt jósolnak szakértői körökben. Ennél az üzemmódnál ugyanis a felhasználó nemcsakhogy folyamatos kapcsolatban marad a gépen futó programmal

(vagyis visszanyeri -- magasabb szinten -- azt az előnyt, amelyet a multiprogramozással elveszitett), hanem módositásokat is hajthat végre a folyamatban, ami nagyfoku rugalmasságot biztosit számára.

A dialógus üzemmód azonban egyuttal magasabb szinten "termel ujra" egy ellentmondást is, ami a meghonosodott fordító rendszerrel kapcsolatos. A gépek bemeneti nyelve ugyanis lényegesen különbözik a belső nyelvtől, s emiatt -- dialógus üzemmódban -- a munkaidő jelentős részét fordítóprogramok futása köti le. Ezen már nem segit feltétlenül a gép nagy működési sebessége (amely kielégitő megoldást jelentett akkor, amikor a magasabb szintü programozási nyelven irt program "egyszeri" fordításáról volt csupán szó), mivel a hatékony dialógus üzemmódnak valamely feladat megoldása során akárhányszoros hozzáfordulást kell biztositania a fordítóhoz. Az uj ellentmondás feloldására olyan rendszerek kifejlesztése célszerü, amelyekben a gépen belül ábrázolt információ nem, vagy csak kevéssé különbözik a bemeneti nyelven ábrázolt információtól. Ehhez az irányzathoz tartoznak a formulavezérlésü számitógépek, amelyek belső nyelve izomorf lehet egy magasabb szintü programozási nyelvvel. Ez a tulajdonság lényegesen kibővitheti a számitógépet közvetlenül felhasználók körét, és -- egyebek között -- matematikai-elméleti problémák vizsgálatában is hasznos és hatékony eszközökké teheti a gépeket. A formulavezérlésü számitógép Kalmár-féle megoldásával a tanulmány harmadik fejezetében részletesen foglalkozunk.

1.3. Észrevételek a számítógépek generációs osztályozásáról

A számítógépek generációs osztályozása -- amint ez az előzőekben kitűnt -- elem-centrikus szemléleten alapul. Ez a szemléletmód az 1960-as évek utolsó harmadáig jól tükrözte a fejlődés fő irányát: mind nagyobb működési sebességet és megbizhatóságot elémi az egyedi gépek effektivitásának növelése céljából olymódon, hogy közben az alkalmazhatósági kör is szakadatlanul bővüljön.

A harmadik generációs gépek fejlődésének időszakában viszont (az 1960-as évek utolsó harmadától kezdődően) fokozatosan megnövekszik a fejlődési folyamatban olyan más jellemzők sulya és aránya, a melyek (hardware és software szempontból) a rendszerre mint egészre vonatkoznak, amelyek tehát a rendszer-centrikus szemléletet állitják előtérbe. A harmadik generációs gépeknél adott vázlatos áttekintés egyik legfontosabb konkluziója éppen az lehet, hogy lényegesen megnövekedett a rendszerjellemzők jelentősége. Jól illusztrálja ezt pl. az a -- korábban már emlitett, de most ismét kiemelendő -- fejlemény, hogy a számitástechnikai rendszereknél a hatékony működés érdekében egy sor szervezési és vezérlési funkció ellátása a gép feladata lett, az un. rendszerprogramok formájában. Meg kell jegyezni, rendszer-jellemzők a korábbi gépeknél is léteztek, de vagy nagyon egyszerüek, illetve merevek voltak, vagy pusztán lehetőségeket jelentettek a későbbi fejlődéshez. A harmadik generációs számitógépek legfejlettebb csoportjánál azonban oly mértékig megnövekszik a jelentőségük, hogy átveszik az egyes számitógépek jellegének meghatározását. A rendszer-jellemzők fokozott előtérbe kerülése azzal a következménnyel járt, hogy az utóbbi években mind gyakoribbá vált a generációs osztályozás vezető szempontjának lazitása, ami abban nyilvánul meg, hogy az elem-bázist előbb "nem egyetlen", majd "nem alapvető" szempontnak tekintik egy, a mai igényekhez igazodó osztályozás megvalósitásánál.

A negyedik fejezetben részletesen is meg fogjuk vizsgálni a rendszerjellemzőket, most csak egy rövid felsorolást adunk róluk:

1. A rendszer strukturája. Ide tartozik a funkcionális egységeknek elemeikből való felépitése, valamint a rendszernek a funkcionális egységekből való megszervezése. (Az utóbbiban a harmadik generációs gépeknél minőségi átalakulás következett be, amely megteremtette az információ-feldolgozó hálózatok felépitésének lehetőségét.) Mind a hardware-, mind a software-stuktura fő jellemzőjévé válik a modularitásra való törekvés.

2. A feldolgozási folyamat megszervezésének módja. Ide tartozik egyrészt a rendszer belső munkájának megszervezése (egyebek között a fejlett hardware és software eszközökkel megvalósuló párhuzamos

feldolgozási módok, az egyenletes leterhelés biztositása, a rendszer egyes moduljai közötti kapcsolat szervezése és vezérlése, a végrehajtás alatti felügyelet stb.), másrészt a rendszer és a külvilág kommunikációjának megszervezése (egyebek között a rendszer kezelhetőségének lényeges fejlesztése, géptől független, könnyen elsajátitható programnyelvek kidolgozása, a legkülönfélébb információ-ábrázolások gépi értelmezésének biztositása, minőségileg uj perifériák kifejlesztése, a sokcélu alkalmazás lehetőségeinek biztositása stb.). A feldolgozási folyamat megszervezésének lehetővé kell tennie a gépek rendszere és az emberek nagy csoportja közötti szimultán kapcsolatot.

3. A rendszer megbizhatósága. Ide tartozik az automatikus hibafelfedés és belső hibaelháritás fejlesztése (hardware és software szempontból egyaránt), a rendszer saját munkájának adminisztrálása, az önszervezési tevékenységekre való alkalmasság biztositása stb.

A fenti jellemzőkre alapozott rendszer-centrikus szemlélet nem vonja kétségbe az elem-bázis fejlesztésének jelentőségét, csupán kizárólagosságát tagadja a számitógépek fejlődésében.

Ezek alapján arra a következtetésre kell jutnunk, hogy a számitógépek (ujabban inkább rendszerek) fejlődési folyamatát két fő szakaszra célszerü bontani:

1. Az egyedi gépek fejlődése, ahol az osztályozás alapját az elembázis tulajdonságai és milyensége képezte,

2. a számitórendszerek fejlődése, ahol az osztályozás alapját a rendszerre, mint egészre vonatkozó tulajdonságok képezik.

Az első fejlődési szakasz magában foglalja az első, második és harmadik generációs gépeket, a fejlődés ezen a szakaszon belül az 1960-as évek végéig tart. A második szakasz kiindulópontja a harmadik generációs gépek lényegesen uj, rendszertulajdonságokkal rendelkező csoportja, ez a szakasz az 1970-es évekre tünik érvényben lévőnek. Hozzá kell tennünk, hogy a második fejlődési szakaszban az első fejlődési szakasz (természetesen kellő módositásokkal), mint részfolyamat tovább folytatódik.

Ezek alapján -- nem elvetve az eddigi generációs osztályozást -- célszerünek látszik módositani a fejlődés periódizálását, s a negyedik generációsnak nevezett osztálynál már a rendszer-centrikus szemléletet érvényesiteni. Ez az osztály tehát már az előző osztályozáshoz képest is uj minőséget juttat kifejezésre.

A jelenlegi fejlődésben három fontos irányzat figyelhető meg(amelyekről feltételezhetjük, hogy a negyedik generációs rendszerek kialakulásában nagy szerepet játszanak):

1. A kisgépek irányzata. A kisgépek korszerű elem-bázissal és igen jó teljesitménymutatókkal rendelkező, korlátozott memóriakapacitásu és szóhosszuságu számitógépek, amelyek meghatározott feladat-osztályokat önállóan képesek megoldani és emellett bármely (számitógépes) feladat esetén elő- és utófeldolgozást tudnak végezni. Feltétlen követelmény, hogy más, (általában nagy) számitórendszerhez illeszthetők legyenek ugy, hogy csak szerves (perifériás) részét alkossák. [4].

2. A számitógép-hálózatok irányzata. A számitógép-hálózat, önmagukban is számitógép funkciókat teljesitő egységek hierarchikus, vagy párhuzamosan működő komplexuma, amely tetszés szerint bővithető (sok-gépes rendszer).

3. A számitórendszerek irányzata. A számitó- (információfeldolgozó) rendszer nagyszámu (és különböző rendeltetésü) processzor, tároló és periféria nagy rugalmasságu, szerves rendszere, amely lehetővé teszi az un. virtuális számitógép elvének teljes érvényre jutását.

Az utóbbi két irányzat (amelyekbe láthatóan beleilleszkedik a kisgép), országos, sőt ennél nagyobb információfeldolgozó hálózatok lehetőségét teremti meg.

Mivel a negyedik generációs osztály a számitógép-fejlődés minőségileg magasabb szintjén lép fel, megitélésénél nem hagyatkozhatunk az eddigi fejlődés egyszerű extrapolációjára, habár a szerzett tapasztalatok, valamint a fejlődás folytonosságát biztositó, "áthuzódó" részfolyamatok csiraszerű jelentkezésének és fejlődésének vizsgálata reális talajt biztositanak a negyedik generációról szóló prognózisnak.

Érvényesitve ezeket a megjegyzéseket, a következő szempontok szerint tekintjük át részletesen a számitógépek eddigi fejlődését:

1. A számitógépekben felhasznált technikai eszközök fejlődése;
2. A számitógép és a külvilág (ember) közötti kapcsolat hardware-eszközeinek fejlődése;
3. Az (univerzális, digitális) számitógépek strukturális fejlődése;
4. A gépi utasitásrendszerek és programozási rendszerek fejlődése;
5. A kezelőrendszerek fejlődése;
6. A programozási nyelvek fejlődése.

(Értelemszerüen az 1-3 pontban a hardware-fejlődés elemzése, az 5-6 pontban pedig a software-fejlődés elemzése dominál, mig a 4 pont bizonyos értelemben átmenet a kettő között.)

2. A SZÁMITOGÉPEK FEJLŐDÉSE

Hivatkozva az első fejezetben kifejtettekre, egy univerzális digitális számitógépet célszerű a következőkkel jellemezni:

- áramköreinek és berendezéseinek technikai kivitele (technikai fejlettsége), és ehhez kapcsolódó mennyiségi paraméterei (processzorteljesitmény, memória-kapacitás stb.),
- a gép által biztositott ember-gép kapcsolat fejlettsége,
- rendszertechnikai felépitése (általános blokkstrukturája, központi vezérlő egységének, aritmetikai-logikai egységének, memóriájának strukturája),
- gépi utasitásrendszere és cimzési rendszere,
- software rendszere.

A fenti szempontok szerint fogjuk áttekinteni a digitális számitógépek eddigi fejlődését, miközben különös figyelemmel kisérjük az egyes területek közötti igények és lehetőségek kölcsönhatásait a területek további fejlődésére, hogy megállapithassuk a fejlődés fő tendenciáit. Ezekből, valamint a jelenleg még kielégitetlen vagy csak részben kielégitett igényekből igyekszünk következtetni a számitógépek várható fejlődési irányaira.

2.1. A számitógépekben felhasznált technikai eszközök fejlődése

Az első programvezérelt digitális számitógép, Ch. Babbage "analitikus gépe" (1833), illetve ennek egyetlen működő modellje, tisztán mechanikai elemekből épült. Aritmetikai egysége és részeredmény-tárolója forgó számláló szerkezetekből állt. Annak, hogy elszigetelt, korát megelőző kisérlet maradt, amely nem illeszkedett be a számolóeszközök szerves fejlődésébe, legfőképpen az az oka, hogy akkor még sem a technológiai adottságok, sem a társadalmi igények nem indokolták a programvezérlés elvének alkalmazását. Igy tipikus példája lett az idő előtt felmerült ötletek megvalósitásának.

Az elektromechanikus számitógépek első példánya, a MARK-I. (1944) -- Howard H. Aiken tervei alapján -- már elektromágneses jelfogókat is felhasznált, elsősorban a számlálókat működtető tengelykapcsolók vezérléséhez. A jelfogók működését viszont lyukasztott kártyák vezérelték.

Az első elektronikus digitális számitógépben, az ENIAC-ban (1946), amely J.P. Eckert és J.W. Mauchly tervei alapján épült fel, már nagymértékben kiküszöbölték a mechanikai mozgást végző elemeket. A forgó mechanikai "főtengelyt" villamos impulzusok helyettesitették. A tengelykapcsolók helyett villamos (elektroncsöves) kapuáramkörök, a mechanikai számlálók helyett villamos számláló áramkörök szerepeltek, bár még mindig decimálisak. A külső programvezérlés nem lyukkártyával, hanem dugaszolható programtáblával történt.

Neumann János és Hermann H. Goldstine nagyjelentőségü elvi tanulmánya után [5] jelentkezett először komolyabb mértékben a memória-igény, mind az adatok, mind a program tárolására (belső programvezérlés). Ugyancsak ekkor lépett fel a kétállapotu elemek -- e jellegükre vonatkozó -- felhasználásának igénye is (bináris aritmetika és binárisan kódolt formáju tárolás).

A nagyobb memóriaigény kielégitésére az első próbálkozások az akusztikus (elsősorban higanyos és magnetostrikciós) késleltető müvonalas memóriák voltak (EDSAC 1949, EDVAC 1950, az első UNIVAC 1951). Átmenetileg megpróbálkoztak katódsugárcsöves tárolóval (Whirlwind-I, 1952, Sztrela, 1953, BESZM, 1953 [2])és elektrosztatikus tárolóval is (IBM 701, 1953). 1952-ben azonban megjelent az első forgó mágnesdob (SEC, Anglia), és a mágnesdobok, mint operativ memóriák kiszoritották a korábbi próbálkozásokat. (Érdekességképpen emlitjük még meg, hogy történetileg a mágnesszalag-memóriák a doboknál hamarabb jelentek meg: már az UNIVAC-I is használt mágnesszalagtárolót, természetesen nem operativ memória, hanem input-output eszköz gyanánt).

Az 50-es évek közepén a ferritgyürüs tárolók megjelenése [6] minőségi változást eredményezett a számitógépek fejlődésében. Bár a mágnesdob- és mágnesszalag-memóriák fejlődését, mint háttér-tárolókét, egyáltalán nem gátolták, operativ memóriaként azonban fokozatosan teljesen kiszoritották őket. A ferritgyürüs tárolók előnyei mindenekelőtt a párhuzamos cimszelekcióból adódtak, amely drágább és bonyolultabb volt ugyan a soros cimszelekciónál, de gyors hozzáférést biztositott az utasitások végrehajtása során előforduló tetszőleges cimhez (random access memory). A működési sebesség növelése ugyanis rendkivül erősen érvényesülő követelmény az 50-es években. A ferritgyürüs tárolók másik nagy előnye az volt, hogy mozgó alkatrészeket nem tartalmaztak, igy megbizhatóságuk és élettartamuk is nagyobb volt a dobokénál.

Az első ferritgyürüs tárolók un. szó-szervezésüek voltak, azaz a tároló minden egyes ciméhez egy-egy szelekciós vezeték tartozott, és ez keresztül haladt a megcimzett szó összes bitjéhez tartozó ferritmagon. Ezeket a vezetékeket szó-vezetékeknek nevezték. A bit-vezetékek viszont az összes szó ugyanazon helyérték-bitjéhez tartozó magokon haladtak keresztül. Komoly előrelépést jelentettek a Massachusetts Institute of Technology-ban kifejlesztett, un. bit-szervezésü ferrittárolók, amelyekben a szelekciós vezetékek két, egymásra merőleges csoportja rácsot ad, és a szelektálás utolsó fázisa magukban a ferritmagokban történik (áram-koincidencia elven). Igy "m" db un. X-vezetékkel és "n" db un. Y-vezetékkel már nemcsak (m+n) cimet lehet szelektálni, hanem (m.n)-et. A bit vezetékek (Z-vezetékek) változatlanul az összes szó ugyanazon helyérték-bitjeihez tartozó magokon haladnak keresztül (egy-egy bit-tábla összes magján).

Ezzel a szervezéssel természetes módon adódott a ferritmagok három-dimenziós elrendezése. A ferrit-"kocka" szervezés még ma is a leggyakrabban alkalmazott módszer ferritgyürüs operativ tárolók kivitelezésénél.

Az 50-es években foglalkoztak a ferrit-anyagok logikai áramkörökben való felhasználásának a gondolatával is. Igy születtek meg pl. a tekercselt ferritgyürüs logikai és egyéb (pl. léptető) áramkörök, valamint a többlyuku transzfluxorok [7]. A ferritelemes logikai áramkörök előnyei a kis méret és suly, robusztus felépités, olcsó előállitás és kisfrekvencián a kis energia-felvétel voltak. Hátrányai: az 1 MHz körüli felső sebességhatár, a kis magok tekercselése bonyolult, és a hőmérsékletváltozásra való, még a félvezetőknél is nagyobb érzékenység. Igy aztán ezeket az áramköröket a félvezetők megjelenése kiszoritotta.

Ugyanerre a sorsra jutottak a szupravezetés elvén alapuló próbálkozások (pl. a kriotronok felhasználásával kapcsolatos kisérletek) is, [8] amelyeknek döntő hátrányai a szupravezetéshez szükséges alacsony hőmérséklet biztositásának a nehézsége és a szupravezetésből a normál vezetésbe történő átkapcsolás viszonylagos lassusága voltak.

Az 50-es évek közepétől megindult a félvezetők elterjedése. A félvezető diódák és a tranzisztorok, mint kétállapotu kapcsolóelemek megjelenése a számitógépekben ujabb minőségi ugrást eredményezett, amely a második generáció kialakulásához vezetett. Az uj elemekkel a magas felső határfrekvencia nagyságrendi javulást tett lehetővé a számolási sebességben, és a méretek csökkenése is jelentős volt. További előnyök: megfelelő, nem ingadozó hőmérséklet esetén lényegesen nagyobb élettartam és nagyobb megbizhatóság.

A tranzisztoros logikai áramkörök fejlődési állomásai: RTL (ellenállás-tranzisztor logika), DCTL (direkt csatolt tranzisztor logika), DTL (dióda-tranzisztor logika), TTL (tranzisztor-tranzisztor logika)[9]. Az ujabb és ujabb áramköri változatokkal elsősorban a zavarérzékenység csökkentését, a terhelési viszonyok javitását és a sebesség növelését kivánták elérni. Az utóbbi szempontból jelentős volt még az 1957-ben felfedezett tunnel-diódák alkalmazása [10]. Ezek felső határfrekvenciája GHz nagyságrendü is lehet. További előnyük a hőmérsékletváltozásokkal és kozmikus sugárzással szembeni viszonylagos érzéktelenségük. Hátrányuk viszont a tolerancia-érzékenység, amelyet válogatással és a Goto által 1960-ban ismertetett kapcsolással lehetett csökkenteni. A Goto-kapuk még egy szempontból birnak jelentőséggel: un. többségi (majoráns) logikát realizálnak. A majoráns logikai áramkörökkel való foglalkozás vezetett el ugyanis később a sokkal általánosabb, un. küszöb-logikai áramkörök kialakulásához.

A tranzisztorok és a félvezető diódák kis méretei lehetővé tették egy uj technológia, a nyomtatott huzalozás bevezetését az áramkörök gyártásában. Ezzel a technológiával megbizhatóan és olcsón lehetett sokszorositani az áramköröket, és igy a számitógépek nagyobb sorozatokban történő gyártása gazdaságossá válhatott, ami nagy előnyöket jelentett a számitógépek tömegesebb elterjedésében.

A tranzisztorok és félvezető diódák elterjedésének az időszakában (az 50-es évek végén, a 60-as évek elején) a számitógépek operativ tárolóiban inkább csak mennyiségi fejlődés mutatkozott: a ferritgyürüs tárolók ciklusidejét sikerült 10 µsec alá leszoritani, kapacitásukat pedig bőviteni. Uj eszközök jelentek meg viszont a háttértárolóknál: a mágneslemezes, majd a mágneskártyás berendezések.

A mágneslemez-háttértárolók elsősorban viszonylag kis átlagos hozzáférési idejük miatt jelentősek, a mágneskártyás tárolók viszont hatalmas kapacitásuk miatt. Hogy mindezek mellett a mágnesszalag háttértárolókat változatlanul alkalmazzák, annak az az oka, hogy a mágnesszalagos tárolók esetén még ma is a legalacsonyabb a fajlagos (bitenkénti) tárolási költség.

A félvezető eszközök kutatása terén a 60-as évek elején jutottak el ahhoz, hogy a félvezető diódák és tranzisztorok gyártásánál alkalmazott technológiákkal passziv áramköri elemeket (ellenállásokat, kapacitásokat és korlátozottabb mértékben induktivitásokat) is előállitsanak. Igy

lehetővé vált komplett áramköröknek egyetlen félvezető "morzsán" (chipen) történő realizálása, azaz: az áramkör integrálása.

Az első, integrált áramköröket felhasználó számítógépek 1965 táján jelentek meg, ami ujabb minőségi ugrást eredményezett (harmadik generációs gépek). Azóta az integrált áramköri technológia óriási fejlődésen ment keresztül, ami lehetővé tette az integráció fokának a jelentős növelését.

Az áramköri integráció alapvető előnyei a következők:

- lényegesen csökkenti a szükséges galvanikus összekötések (forrasztások) számát, igy javitja a számológép megbizhatóságát,
- a méretek csökkentésével is növeli a számitógép működési sebességét (ugyanis a jelek (információ) terjedési sebessége jelenlegi ismereteink szerint a fénysebesség által korlátozott, 1 nsec alatt kb. 30 cm),
- az áramköri készletek széles skálájának a szabványositásával a számitógép-tervezőknek már nem kell áramköri tervezéssel foglalkozniok, csak logikai-rendszertechnikai tervezéssel,
- az integrált áramköri technológiák javulásával a fajlagos (pl. logikai kapunkénti) költségek jelentősen az egyedi tranzisztorokból stb. felépitett áramkörök költségei alá csökkennek.

Maguk az integrált áramköri technológiák alapvetően két csoportba sorolhatók: a félvezető alapu és a szigetelő alapu technológiákra [1].

A félvezető alapu (vagy más néven monolitikus) integrált áramkörökben az áramköri elemeket általában planáris-diffuziós, ujabban ion-implantációs technológiával alakitják ki a félvezető kristálytömbben. A monolitikus integrált áramköröknek két változata van: a bipoláris és a MOS (Metal-Oxide Semiconductor) áramkörök. Az elsőben a szokásos (bipoláris) tranzisztorok szerepelnek: jelentős előnyük a nagy sebesség. A MOS áramkörök viszont térvezérlésü tranzisztorokat tartalmaznak: kisebb helyigényükkel, egyszerübb technológiájukkal, kevesebb passziv alkatrész-igényükkel és kisebb teljesitmény-szükségletükkel tünnek ki.

A szigetelő alapu integrált áramköröknek is két változata van: a vékonyréteg- és a vastagréteg-áramkörök. A vékonyréteg-technológiánál vákuumpárologtatásos vagy katódporlasztásos módszerrel viszik fel a szigetelő alapra az ellenállás- illetve vezetőréteget, és oxidációval alakitják ki a kapacitások szigetelő rétegét, mig a vastagréteg-eljárásban szitanyomással alkitják ki az ellenállásokat. A réteg-technológiák pontosabb értéktartást tesznek lehetővé, mint a monolitikus technológiák, viszont önállóan nem alkalmasak aktiv áramköri elemek kialakitására, igy ezeket inkább a két alapvető technológiát összevonó, un. hibrid technológiákban alkalmazzák. Ezek közül a legjelentősebb a réteg-hibrid eljárás.

Ennél a kialakitott passziv réteg-áramkörbe forrasztással vagy termokompresszióval kötik be az aktiv áramköri elemeket tartalmazó félvezető kristályokat. Ilyen eljárással készülnek például az IBM vastagréteg áramkörei. A félvezető alapu hibrid eljárásban az aktiv elemeket tartalmazó kristály felületén vékonyréteg-eljárással alakitják ki a passziv elemeket.

A gyártástechnológiák fejlődésével egyre bonyolultabb áramkörök integrálása vált lehetővé. Az integráció mértékét SSI-vel (Small Scale Integration), MSI-vel (Medium Scale Integration) és LSI-vel (Large Scale Integration) szokás jelölni.

Egy SSI-áramkör általában néhány logikai kapuáramkört vagy triggert tartalmaz. Áramkörileg az RTL és DTL mellett kifejlesztették a TTL-áramkörök sokemitterü tranzisztoros változatát, majd az ECL (Emitter Coupled Logic) áramköröket [12]. Utóbbiak jelentősége, hogy sikerült velük kiküszöbölni a korábbi áramköri tipusoknál fellépő telitési problémát, amely a müködési sebességet lényegesen korlátozta. Igy az ECL áramköröknél már egy-egy logikai kapu átviteli ideje (a jel-terjedés késése) a néhány nsec nagyságrendjébe esik.

Az MSI-áramkörök már pl. 4-8 fokozatu teljes összeadó áramköröket, néhányszor 10 bitnyi léptető regisztereket, tárolókat, stb. tartalmaznak. Technológiailag is bonyolultabbak az SSI-áramköröknél, pl. a többi rétegben elhelyezkedő fémezés megoldásában.

Az LSI-áramköröket jelenleg még elsősorban maximum néhány Kbyte kapacitásu, különböző célu tároló-blokkok formájában alkalmazzák, de van már példa monolitikus integrált áramkörü főtárolókra is.

Néhány tárolási tipus és alkalmazási területük a következő:

- FIFO (first in - first out) tárolók: elsősorban puffermemóriaként alkalmazzák őket.

- LIFO (last in - first out) tárolók: szokásosabb nevük a veremmemória(nesting store, push-down, stack), jól alkalmazhatók forditások során kifejezésfeldolgozások, egymásbaskatulyázott szubrutin-hivások, ill. rekurziv szubrutin-hivások megkönnyitésére.

- Dinamikus shift- (léptető-) regiszter memóriák: jól alkalmazhatók például display-berendezések saját memóriájaként, a ciklikus megjelenitéshez. Itt jegyezzük meg, hogy ugyanerre a célra ujabban szivesen használnak ultrahangos üvegalapu késleltető müvonalas tárolókat is, amelyeket már 50 MHz frekvencián is sikerült üzemeltetni. Előnyük, hogy kedvezőtlenebb rezgési, ütési és nedvesség feltételek mellett is képesek müködni, mint a huzalból készültek. (Az 50 MHz müködési frekvencia azt jelenti, hogy pl. egy 1 msec késleltetésü müvonal 50 000 bit kapacitásu: soros kódolás mellett kb. 6000, egyenként 8 bites karakter keringtetésére és 1 msec alatti kibocsátására vagy fogadására alkalmas).

- Tartalom-cimzésü (asszociativ) memóriák: elsősorban lapcimzésnél használják a logikai lapszám--fizikai lapszám kapcsolatokat tar-

talmazó laptáblákhoz.

- Scratch-pad memóriák: a processzor regisztereit (a programállapot regisztereket, az akkumulátor-regisztereket, a báziscim- és indexregisztereket, a maszk-regisztereket stb.) sokszor egyetlen jól szervezett LSI tárolóblokkba gyüjtik. Ezt a tárolóblokkot szokták scratchpad (magyarul: "firkáló-füzet") memóriának nevezni.
- Átirható mikroprogram-tárolók: A mikroprogramozás bevezetéséhez rendkivül gyors mikroprogram-tárolókra volt szükség. Ezt kezdetben csak kiolvasható (RO, read-only) memóriákkal tudták megvalósitani. Ilyen megoldások voltak például a kapacitiv elven müködő, vagy a ferritmagos fix-programos tárolók. Az átirható (RW, read-write) monolitikus integrált áramkörü mikroprogram-tárolók alkalmazása olyan jelentős ujitások bevezetését tette lehetővé, mint pl. a hardware emuláció, probléma-orientált és magasszintü gépi utasitás-rendszer kialakithatósága, stb.
- Monolitikus integrált áramkörü főtárolók [13]: monolitikus bipoláris főtárolót alkalmaznak pl. az IBM 370-es sorozatban vagy az ILLIAC-IV processzor-elem memóriáiban. Monolitikus MOS tárolómodulokból épül pl. fel a Data General cég Supernova kisszámitógépének a főtárolója. Az LSI főtárolók előnyei általában nagy gyorsaságuk és kis fogyasztásuk, hátrányuk (egyelőre) viszonylagosan magas áruk.

Ha már technológiai kérdésekről esett szó, érdemes még megemliteni az integrált áramkörök külső, kereteken történő galvanikus összekötésében bekövetkezett változást is: a forrasztásos kötések helyett a vezetékcsavarásos (wire-wrap) technikát alkalmazzák egyre elterjedettebben. Az utóbbi ugyanis nagyobb megbizhatóságunak és hosszabb élettartamunak bizonyult, mint a forrasztásos kötéstechnika. A wire-wrap technika pillanatnyi hátránya a viszonylag drága célszerszám-szükséglet.

A 60-as évek közepén a számitógép-fejlesztők jellegzetes problémája volt a szokásos ferritgyürüs operativ memóriák ciklusidejének lemaradása a rendkivül felgyorsult processzorok ciklusidejéhez képest. A ferrittárolók müködési sebességét elsősorban a bennük használt magok méretei korlátozták, mivel az átmágnesezés sebessége nagymértékben függ az átmágnesezendő anyagmennyiségtől. A magok méretei viszont a bit-szervezésü ferrittárolók bonyolult füzési technikája miatt nem lehetett egy bizonyos határon tul csökkenteni, tehát az ellentmondás feloldásának ez az u'ja nem volt járható. Ezért meggyorsultak a mágneses vékonyrétegekkel kapcsolatos alap- és alkalmazott kutatások, és a 60-as évek végén megjelentek ezeknek első eredményei is.

Az első mágneses vékonyréteg tárolókat, a mágnesrudas operativ memóriákat az NCR cég jelentette be és alkalmazta a Century-sorozatban. Beirási--kiolvasási elvüket tekintve, a mágnesrudas tárolók és a ferritgyürüs tárolók között még nem volt különbség.

A következő állomás a huzal-kivitelü mágneses vékonyréteg tároló, amelyet pl. az UNIVAC 9300-as és 9400-as gépekben is alkalmaznak [14]. Itt már a müködési elv is lényegesen módosult: az anizotróp réteg-kialakitás és a mágnesezési vektornak ellenkező irányba állitása helyetti forgatása szükségtelenné teszi kiolvasáskor a visszairó félciklust (nemdestruktiv tároló). Ezért is, és természetesen döntően a rendkivül kis átmágnesezendő anyagmennyiségek miatt a huzal-kivitelü vékonyréteg tárolók ciklusidejével sikerült elérni a néhányszor 10 nsec értéket. (Az irás--olvasás elvével kapcsolatban még megjegyezzük, hogy a kétlyuku ferritekből felépitett tárolók szintén nem veszitik el információtartalmukat kiolvasáskor, igy ezeknél sincs szükség visszairó félciklusra.)

Huzal-kivitel helyett néha film-kivitelü vékonyréteg tárolókat is épitenek, amelyeknek a müködési elve azonos a huzal-kivitelüekével.

A jelenlegi mágneses vékonyréteg-kutatások (a kapacitás és a müködési sebesség növelésének szükségleteitől ösztönözve) a dinamikus mágnesbuborék háttérmemóriák létrehozására irányulnak, amelyek a mágneses vékonyréteg tárolók egy speciális változatát képezik. A Bell Telephone Laboratories-ban például egy kb. 15 millió bit tárolására elegendő, a jelenlegi mágneslemezes tárolók átviteli sebességénél kb. 10-szer gyorsabb dinamikus buborékmemória fejlesztésén dolgoznak [15]. Ezeknek a tárolóknak várhatóan a következő előnyeik lesznek a lemeztárolókhoz és egyéb, jelenleg használatos háttértárolókhoz képest:

- nem tartalmaznak mozgó alkatrészt (csak a buborékokat keringtetik, a film mechanikailag áll), ezért hosszabb az élettartamuk;
- a buborékmemóriáknál a tárolt adatok kiolvasása és visszairása nélkül is végezhetünk az adatokon logikai müveleteket;
- az információtömbök mozgatása a tároló egyik területéről egy másikra, egy lépésben (a főtárolóba való áthelyezés nélkül) végrehajtható;
- nagyobb az információ-sürüség (kb. 10^6 buborék/inch^2), nagyobb az információ-mozgatási sebesség (kb. 10^6 lépés/sec), rendkivül kicsi a teljesitmény-szükséglet (pl. a monolitikus integrált áramkörü, ciklizált dinamikus shift-regiszter memóriákhoz képest kb. 3 nagyságrenddel kisebb a teljesitmény-szükséglete: pl. 1 millió buboréknak 1 másodperc alatt 1 milliószor történő továbbléptetéséhez, azaz másodpercenként 10^{12} léptetéshez mindössze 40 mW kell).

A fénysugár mint információhordozó, jelenlegi ismereteink szerint a leggyorsabb átviteli "csatorna", további jelentős tulajdonsága a kis hullámhosszból adódó rendkivül felbontási finomság. Különböző vékonyrétegek kétdimenziós tárolási képességét speciális interferencia módszerekkel (holográfia) 3-dimenzióssá lehet alakitani, és ezáltal az információsürüséget jelentősen megnövelni.

Mindezek a tulajdonságok inditották a kutatókat arra, hogy foglalkozzanak az optikai módszereknek és eszközöknek a számitógépekben történő alkalmazásával. A legkülönbözőbb területeken sikerült már eddig is alkalmazni őket: pl. az LSI gyártástechnológiákban, mikrofilm-output (és ujabban input) berendezésekben, folyadék-kristályos és fényemittáló diódás kijelzőkben, lézeres adatátvitelnél stb. A legjelentősebb eredményeket azonban a jelenlegi kutatások közül a holografikus tömegtárolók kifejlesztése igéri.

Magának a holográfiának az elve már 1947 óta ismeretes, 1960-ra pedig részletesen ki volt dolgozva, gyakorlati alkalmazhatóságával azonban várni kellett megfelelő eszközök, a lézerek kifejlődéséig.

Holografikus tárolók információ-hordozó közegeként először olyan vékonyrétegekkel próbálkoznak, amelyekben a fénysugarak kémiai változásokat idéztek elő. Bár igy rendkivül nagy információ-sürüséget (kb. 10^6 bit/mm^2 sikerült elérni, a végbemenő fotokémiai folyamatok általában irreverzibilisek voltak, igy csak egyszeri beirásra volt lehetőség, törlésre, átirásra nem.

Az RCA-ban kifejlesztett holografikus tömegtároló már mágneses vékonyréteget használ információ-hordozó közegnek [16]. Ennél az információ-sürüség ugyancsak kb. 10^4 bit/mm^2, viszont lehetőség van a tárolóközeg bármely pontjának lokális törlésére és igy ujrairására is. Az emlitett tároló kapacitása kb. 10^{10} bit, egyszerre kb. 10^4 bites adattömböket (lapokat) lehet kiolvasni vagy beirni. A felhasznált lézer átlagteljesitmény-szükséglete kb. 0,1 W. A fénysugár deflektálását (azaz a cimszelekciót) akuszto-optikai elven végzi, igy semmiféle mozgó alkatrészt nem tartalmaz. A hozzáférési idő néhány nsec, tehát lényegesen jobb a hagyományos tömegtárolókénál.

Talán ez a néhány technikai adat is érzékelteti, milyen hatalmasat lendithet a számitógépek teljesitőképességén, ha ilyen tömegtárolókat sorozatban, megfelelő áron lesznek képesek gyártani.

2.2. A számitógép és a külvilág (ember) közti kapcsolat fejlődése

Az ember-gép kapcsolatot megteremtő perifériás berendezések sorát a lyukkártya-perifériák nyitották meg. Ezeket követték az elektromos I/O irógépek és a lyukszalag-perifériák. Utóbbiak egyszerűbbek és olcsóbbak, de lassabbak is voltak a lyukkártyás perifériáknál. Egy-egy adat vagy program-utasitás módositása a már elkészitett lyukszalagon nehézkesebb is, mint lyukkártyáknál. Igy a lyukszalag perifériák -- különösen az adatfeldolgozó számitógépeknél -- csak korlátozott mértékben terjedtek el.

Az elektromos I/O irógépek nyomtatási sebessége nem tette lehetővé nagyobb adatmennyiségek megfelelően gyors kivitelét, ami -- különösen a soros feldolgozás feltételei mellett -- rendkivül erősen korlátozza a gép kapacitásának kihasználását. Ezért ezeket az impakt sornyomtatók (repülőnyomtatók) bevezetésától kezdve már csak az operátor és a gép (program) közötti üzenetváltásokra használják.

A repülőnyomtatók jól fejleszthető, célszerü perifériáknak bizonyultak. A jelenlegi korszerü sornyomtatók jelkészlete 60-80 jel, egy sorban lévő pozicióik száma 120-160, a nyomtatási sebesség 1000-2000 sor percenként.

A kiviteli sebesség további fokozását lehetett elérni a nem-impakt elveken dolgozó sornyomtatókkal (ilyen pl. a Benson cég elektrosztatikus sornyomtatója), amelyek kb. 15 000 sort nyomtatnak percenként (ami megfelel kb. 20 000 karakternek másodpercenként), igaz, hogy csak egyetlen példányban.

Még elképesztőbbek a jelenleg még kuriózumnak számitó katódsugárcsöves, elektronsugaras, fényemittáló diódás vagy lézeres mikrofilm-output berendezések adatai: másodpercenként 100 000 karaktert képesek filmre rögziteni, és az adatok speciális (pl. diozo vagy vesicular tipusu) film alkalmazásával rögzités után azonnal kivetithetők és kiértékelhetők. Ujabban mikrofilm-input berendezések fejlesztésével is foglalkoznak, pl. száloptika alkalmazásával.

Már hosszabb ideje fejlesztik a vizuális és akusztikus információk közvetlen be- és kivitelére szolgáló berendezéseket. Sorrendben a vizuális kimeneti egységek jelentek meg először: az inkrementális plotterek (rajzoló berendezések), valamint az alfanumerikus és grafikus display-k (katódsugárcsöves megjelenitők). A következő lépés a speciális vizuális bemeneti egységek (bizonylat-olvasók) és az univerzális képbevivők (pl. a FIDAC film-input berendezés) kifejlesztése volt.

Az IBM már több éve hirdeti audio válaszadó egységeit. Elsősorban a Szovjetunióban, Japánban, az USA-ban (az IBM és a Bell cégnél) fáradoznak olyan berendezések létrehozásán, amelyek segitségével a szó szoros értelmében, emberi hangon lehet beolvasni az adatokat és utasitásokat a számitógépbe [17].

1961-ben a MIT kutatói javasolták először olyan, un. on-line számítóközpont létrehozását, amelyet nagyszámu felhasználó tud un. felhasználói végállomások vagy terminálok segitségével közvetlenül a munkahelyéről bármikor felhivni és egymással szimultán igénybevenni. A MIT-ben ezután konkrét lépéseket is tettek az elv használhatóságának a bizonyitására, és ennek eredményeképpen indult meg a felhasználói végkészülékek gyors fejlődése.

A terminálok kiépitettsége a felhasználó céljaitól függően rendkivül széles: állhatnak egyetlen villamos I/O irógépből vagy alfanumerikus display-ből is, de tartalmazhatnak kártyaperifériákat vagy akár sornyomtatót is. Ujabban két irányban fejlesztik tovább a terminálokat.

Az egyik irányt a pufferelt terminálok jelentik (pl. az IBM CMC-72 tipusu cserélhető mágneskártyás végberendezése), amelyek segitségével elérhető mind az adatátviteli vonalak, mind pedig a központi számitógép jobb kihasználása. A pufferelés lehetővé teszi, hogy rendszertelenül és lassan érkező adatokat összegyüjtsünk, és nagyobb csoportokban továbbitsunk a számitógép felé (lehet az összegyüjtött anyag egy, a terminál irógépén legépelt program is), vagy a számitógéptől nagy sebességgel érkező adatcsoportokat átmenetileg, a lassubb megjelenités idejéig tároljuk. A pufferelt terminálok természetesen változatlanul használhatók interaktiv módon is.

A másik fejlesztési irány a mikroprogramozást hivja segitségül olyan programozható (intelligens) terminálok megteremtéséhez, amelyeket emulációs technikával kompatibilissé tehetnek a legtöbb számitógéppel. Ilyen pl. az Electronic Associates, Incorporated (EAI) DCT-132 tipusu terminálja, szakaszos feldolgozásokhoz (Remote Batch Terminal). Az intelligens terminálok ugyanakkor kis számitógépekként viselkedve, pl. az adatok egyszerübb igényü előfeldolgozására is alkalmasak.

Az elért eredmények ellenére az ember-gép kapcsolat (a technikai eszközök vonatkozásában is) a számitógép-fejlesztési folyamat "szük keresztmetszete". A jelenlegi fejlődésben igen erősen érvényesülő szükségletek léptek fel az információ be- és kiviteli rendszerek fejlesztésére, amelyek -- a prognózisok szerint -- minőségi változást fognak előidézni ezen a területen. Várható, hogy a jövőben a klasszikus berendezéseket (lyukszalag, lyukkártya, mágnesszalag, klaviaturás eszközök), különböző jeladók váltják fel, amelyek közvetlenül az elsődleges adatforrásokról gyüjtik az információkat.

2.3. Az (univerzális, digitális) számitógépek strukturális fejlődése

2.3.1. Általános blokk strukturák

Az első, tárolt-program vezérlésü elektroncsöves számitógépek rendszertechnikailag processzor-centrikus felépitésüek voltak; legjobban ugy lehetne ezt jellemezni, hogy az I/O tevékenységeket is a központi vezérlő egység bonyolitotta le az aritmetikai egység regisztereinek (mint puffer-, gyüjtő- és lebontó-regiszteréknek) a segitségével. Ennek a felépitésnek egyetlen előnye, hogy az operativ memóriának csak egyetlen blokktól, a központi vezérlő egységtől kellett fogadnia parancsokat, hátránya viszont, hogy az aritmetikai egységet feleslegesen bonyolitotta, és lehetetlenné tette a számitási és I/O tevékenységek átlapolt elvégzését. A számitógépek müveleti sebességének a növekedésekor ez a hátrány nagyon jelentőssé vált, amit ugy lehetett kiküszöbölni, hogy az I/O tevékenységek vezérlése alól mentesitették a processzort, egy vagy több autonóm vezérlő egység (autonóm csatornák) létrehozásával. Igy a CPU-nak (központi processzorok) már csak ezek valamelyikét kellett beinditania a kivánt I/O müvelet teljes lebonyolitása helyett. A csatornáknak viszont a CPU-tól függetlenül is kapcsolatban kellett lenniök az operativ memóriával, illetve egy megszakitás-rendszeren keresztül értesiteniök kell a CPU-t az I/O müvelet befejeződéséről.

Ezért az operativ memória központi helyzetüvé válik: különböző helyekről gyakorlatilag véletlenszerüen érkező kéréseket kell valamilyen elsőbbségi sorrendben kiszolgálnia. Vezérlése nyilván bonyolódik, viszont lehetővé válik az aritmetikai-logikai és az I/O müveletek átlapolt végrehajtása.

A memória-centrikus felépitésre való áttérés még az 50-es években elkezdődött, de az I/O tevékenységeket még hosszu ideig I/O utasitások sorozatára lebontva programozták és hajtották végre; legfeljebb a megszakitás-rendszer alkalmazása nyujtott némi átlapolási lehetőséget a számitási munkák és az elemi I/O tevékenységek között. Az autonóm csatornák csak kb. 1960-tól kezdve kezdték kiszoritani a programozott csatornákat, és a 60-as évek közepére váltak általánossá.

Közben azonban a számitógép egyes részeinek jobb kihasználása érdekében (amit a gyorsan növekvő müködési sebesség egyre inkább előtérbe állitott) megindult egy folyamat: a modularitásra való törekvés. Ez azt jelentette, hogy az eredetileg egyetlen funkcionális blokk-ként müködő operativ memória, aritmetikai egység, stb. több blokkra osztódott, amelyek egyre nagyobb autonómiát kaptak a szimultán müködés elősegitésére. Igy az operativ memória autonóm modulokra osztása lehetővé tette, hogy ténylegesen egyidőben lehessen kiszolgálni különböző memóriához fordulási igényeket, feltéve, hogy azok különböző modulokra vonatkoznak. Az I/O egységek szimultán müködtethetőségét az idő-

multiplex módon dolgozó un. multiplexor-csatornával, ill. több szelektor-csatorna alkalmazásával érték el.

A központi feldolgozómü esetében a modularitás először csak az aritmetikai-logikai egység felbontását jelentette (pl. a lebegőpontos hardware vagy a szorzás-osztás hardware különválasztását, amelyeket azután opcióként kezeltek), majd a feldolgozómü teljes többszörözésével eljutottak az első multiprocesszor-strukturákhoz is.

A multiprocesszor-strukturák "ősei" már a 60-as évek elején megjelentek a SOLOMON-I. rendszerrel. Ebben egy hagyományos processzor helyett kétdimenziósan elhelyezett, egy bit szélességü processzor-elemek hálózata szerepel. Egy kisérleti 3x3 processzor-elemes hálózat 1963-ban meg is épült, majd ezt követte egy, a CDC-160/A gép által vezérelt 10x10-es processzor-elem hálózat [18]. A SOLOMON-II. rendszer már helyi memóriában helyezi el a programot. Realizálásra ugyan nem került sor, de ennek a rendszernek strukturális jellegzetességeit viseli magán egy jóval később, 1970-re megépitett, un. parallel-processzors gép, az ILLIAC-IV [19].

Szigoruan véve multiprocesszor-strukturát jelent az un. kommunikációs processzorok alkalmazása is. A nagyszámu perifériának, az adatátviteli és terminál-hálózatnak a kezelése olyan bonyolult feladat, amely megköveteli, hogy erre a célra külön processzort alkalmazzanak. Ilyen strukturával rendelkezik például az UNIVAC-1108 SP gép. Itt a kommunikációs processzor teljesen megegyezik a központi feldolgozómüvel, igy annak meghibásodása esetén képes átvenni a feladatait is.

Ahogyan az átviteli csatornák kommunikációs processzorokká fejlődtek, a hozzájuk kapcsolódó periféria-vezérlő egységeket is célszerübbé vált kisteljesitményü, un. perifériális processzorokkal helyettesiteni. Ezek különösen nem-standard perifériák alkalmazásakor vagy pl. bemenő adatok előfeldolgozásához használhatók jól, de elképzelhető, hogy távolsági feladatbeadásoknál elvégezhetik a magasabb szintü nyelven irt programok gépi bázisnyelvre történő fordiását, vagy az ember-gép dialógus megszervezését is. Strukturálisan a perifériás processzorok helyén szerepelnek a szatellit-gépek is, amelyekkel az előbbi feladatokon tulmenően, kisebb volumenü számitási feladatok önállóan is megoldhatók.

Az 1960-as évek végén megkezdték az első számitógép-hálózatok kiépitését is. Két fontos feltételnek kell teljesülnie ahhoz, hogy a számitógéphálózat létrehozható legyen: megfelelő adatátviteli hálózatnak kell rendelkezésre állnia, és egységesiteni kell az összekapcsolandó számitógépek programozási és interface-rendszerét. A kompatibilis számitógépcsaládok az utóbbi követelményt természetszerüleg teljesitik, igy a gépcsaládok nagyobb tagjait ugy alakitják ki, hogy azokhoz -- a perifériális processzorok módján -- a kisebb modellek mint szatellit-gépek kapcsolódhassanak. Igy a szatellitgépet használónak adott esetben egy jóval nagyobb teljesitményü gép is a rendelkezésére áll, ha a feladata megoldására a szatellit-gép már nem elegendő. Azonos teljesitményü modellek is össze-

kapcsolhatók, igy lehetővé válik a különböző központok hardware és software erőit egy-egy nagy volumenü probléma közös megoldására egyesiteni, és a teljes rendszer megbizhatóságát a felhasználók szempontjából az egyes központok megbizhatósága fölé emelni.

Országos méretü számitógép-hálózat kialakulása csak kb. 1980-ra várható, a legfejlettebb államokban. A nagy áteresztőképességü (legalább 300 kilobit/sec, de a jelzett időszakra célszerüen 3 megabit/sec) adatátviteli hálózat megteremtése ugyanis jelentős gazdasági és müszaki feladatot jelent. Viszont csak ilyen sebességü hálózattal képzelhető el egy olyan rendszer, amely biztositja a hozzá tartozó összes párhuzamosan müködtethető funkcionális egység (processzorok, memóriák stb.) optimális leterhelését és teljesitőképességének a kihasználását a mindenkori feladatok függvényében [20]. Természetes az is, hogy a hálózatot müködtető rendszernek a jelenlegi operációs rendszereknél jóval bonyolultabbnak és hierarchikus felépitésünek kell lennie [21].

Mindezek alapján a következő néhány évben hazánkban csak egymástól nem tul távol felépitett, központi nagygépből és szatellit-gépekből, ritkábban homogén gépekből álló hálózatok kialakulása várható.

A számitógép-rendszerek kapcsán meg kell emlitenünk a parallel hibrid géprendszereket is, amelyeknek az igénye olyan feladatokkal kapcsolatban merült fel, amelyeket tisztán digitális számitógéppel csak nagyon gazdaságtalan módon lehet megoldani. A hibrid rendszer két eljárás kombinációja: a gyártók igyekeznek rendelkezésre bocsátani mind az analóg gépek parallel strukturájából, mind pedig a digitális gépek szekvenciális szervezéséből származó előnyöket. A két rendszer eredményes együttmüködéséhez hatékony csatoló rendszer szükséges, amely az adatok mindkét irányu átalakitását és cseréjét biztositja. Az analóg gép konzoljának valamennyi funkcióját a digitális gépnek kell vezérelnie (pl. üzemmód-kiválasztás, egységek cimzése, potenciométerek beállitása, rendszerellenőrzés, stb.). Végül egy logikai interface-nek biztositania kell az analóg és digitális rendszer közötti megfelelő szinkronizmust.

Az analóg és a digitális gépek egyesitésének az előnyei:

- a digitális pontosság egyesitése az analóg gyorsasággal;
- egy valóságos rendszer digitális modellezésénél lehetőség van e rendszerrel való összekapcsolásra;
- az analóg modellezés rugalmasságát növelni lehet a digitális vezérléssel és tárolással;
- akár diszkrét, akár folytonos bemenőjelek közvetlenül feldolgozhatók.

A hibrid rendszerek főbb alkalmazási területei:

- Folytonos, diszkrét és osztott paraméterü rendszerek, ember-gép rendszerek, valamint szochasztikus folyamatok modellezése (pl. kémiai reakciók és reaktorok szimulációja, rádiófrekvenciás távközlő vonalak szimulációja, sok szabadságfoku aerodinamikai rend-

szerek szimulációja, űrhajós kiképzés, stb.).

- Dinamikus optimatizálási problémák (pl. sokparaméteres optimalizációs feladatok vagy a Pontrjagin-elméletet alkalmazó optimális szabályozási feladatok). Ezeket a feladatokat általában iterációs módszerekkel lehet megoldani, amelyek az integrál-szabályozáshoz hasonlóan nagyszámu integrálással járnak együtt, ezenkivül megkövetelik a logikai döntéseket, a közbenső számitásokat, a tárolást, stb. is. Ezért önállóan analóg gépen nem is oldhatók meg, a tisztán digitális megoldás pedig a numerikus integrálások ezrei miatt rendkivül hosszu időt igényelne.
- Biológiai rendszerek tanulmányozása (pl. az emberekhez, ill. állatokhoz kapcsolt érzékelőkből származó jelek, EKG, EEG, stb. analizálása és identifikálása).

Az első nagyteljesitményü hibrid rendszert 1967-ben hozták létre, négy COMCOR gyártmányu CI-5000 tipusu analóg és egy CDC-6400 tipusu digitális gép összekapcsolásával. A hibrid programok irására a CLASH nyelv szolgált. Az EAI cég 1970-ben kihozott EAI-690-es rendszere már sokkal korszerübb mind hardware, mind software tekintetében. Programozási nyelve a HOI (Hytran Operations Interpreter).

2.3.2. Központi vezérlő egység (CCU) strukturák

A digitális számitógépek központi vezérlő egysége programvezérelt szekvenciális automatának tekinthető. Ennek a szemléletnek megfelelően az elektronikus számitógépek központi vezérlő egységeit hosszu időn keresztül a szekvenciális müködésü kombinációs áramkörök kialakult tervezési, minimalizálási és realizálási módszerei szerint valósitották meg. A szekvenciális áramkörök klasszikus felépitése a következő: a bemenőjel-kombinációkat egy logikai áramkör (az un. kombinációs áramkör) figyeli. Az áramkör pillanatnyi belső állapotának jelzésére tároló elemek szolgálnak. A belső állapottárolók kimenetei szintén az áramkör bemenetére csatolódnak vissza. Adott bemenő jelsorozatra a szekvenciális áramkör állapotában a kialakitott kombinációs áramkörtől, a bemenő jelsorozattól és a korábbi állapottól függő változás következik be. A kimenő jelsorozat szintén a kialakitott kombinációs áramkör, a bemenő jelsorozat és a pillanatnyi belső állapot függvénye.

Ezt a felépitést legélesebben a CCU un. központi impulzus-elosztó részében figyelhetjük meg. Itt a bemenő jel általában a végrehajtandó utasitás kódja (dekódolón keresztül), néhány feltétel-kód és/vagy egy órajel-generátor impulzusa (aszinkron rendszereknél). A belső állapotjelzést többnyire számláló-regiszterek látják el. A kimenő impulzusok a számláló-regiszterek állapotától függően inditják a megfelelő funkcionális egységet.

Az ilyen felépitésü vezérlő egységeket huzalozott logikáju vezérlő egységeknek nevezzük, mert müködésüknek a megváltoztatását csak a logikai áramköri struktura (huzalozás) fizikai átépitésével érhetjük el.

1951-ben M. V. Wilkes (Cambridge-i professzor) vetette fel először, hogy az egyes gépi utasitások alkalmasan megválasztott mikroutasitások sorozataként is elvégezhetők lennének, azaz: a gépi utasitásokat lehetséges software eszközökkel, mikroprogramokkal is definiálni. Az elvnek (tehát a hardware "software-esitésének") a realizálását hosszu időn keresztül az akadályozta, hogy a gazdaságos megvalósitáshoz a mikroprogram--tárolóknak sokkal gyorsabb müködésüeknek kell lenniök az operativ memóriáknál. A kivánt sebességet először RO (csak-kiolvasható) tárolókkal sikerült elérni (pl. kapacitiv ROM-okkal), de az LSI-technika kialakulásával lehetővé vált a megfelelően gyors, viszonylag kis kapacitásu, átirható mikroprogramtárolók alkalmazása is.

A tárolt logikáju (mikroprogramozott) vezérlő egységek előnyei a huzalozott logikáju vezérlő egységekkel szemben:

1. A CCU egyszerübb felépitésü, megbizhatóbb müködésü. A mikroprogram-software (un. firmware) a használattól nem "hibásodik meg", nem szorul állandó "karbantartásra".

2. Átirható mikroprogram-tároló esetén az utasitás-rendszer flexibilis. Például kedvező lehetőség van probléma-orientált gépi utasitásrendszer, vagy olyan magas szintü gépi bázisnyelv kialakitására, amely jelentősen leegyszerüsiti a magasabb szintü nyelven irt programok forditását, a kompilálást firmware interpretálással pótolva. A másik lehetőség az emuláció technikájának az alkalmazása a kompatibilitás megteremtéséhez. Ebben az esetben a befogadó rendszer (host system) mikroprogram-tárolóját olyan mikroprogramokkal töltik fel, amelyek a célrendszer (target system) utasitásrendszerének felelnek meg. Igy a cél-rendszeren futó programok változtatás nélkül a befogadó rendszeren is futtathatók. (Az IBM 370-es gépein igy lehet futtatni pl. az IBM 440-re, illetve az IBM 7010-re irt programokat, vagy például az IBM 360/25-ön a Burrough Corparation B 220-ra irt programokat.) Az un. soknyelvü processzorok (ugyanilyen elven) a legkülönfélébb géptipusok gépi nyelvét képesek interpretálni (ilyen pl. a Standard Computer Corporation által tervezett, 1970-ben publikált MLP-900 processzor (MLP - a "multilingual processzor" röviditése), amely még a target system aritmetikai strukturájának az utánzására is képes [22]).

3. Hardware meghibásodás esetén egyszerübbek és gyorsabbak a tesztelési eljárások. A fejlettebb mikroprogramozott gépeknél beiktatnak a gépi utasitásrendszerbe egy diagnosztizáló utasitást, a számitógép (a processzor, a főmemória és a csatornák) funkcionális

egységeit hibafigyelő áramkörökkel egészitik ki. Bármilyen hiba jelzésekor megszakitás következik be, a vezérlés pedig egy olyan rutinnak adódik át, amely a DIAGNOSE utasitást tartalmazza. Az ehhez tartozó, viszonylag bonyolult mikroprogram először a hibajelzés alapján behatárolja a hibát, majd a számitásba jöhető funkcionális egységeket alkalmas mikroutasitásokkal rendkivül gyorsan kipróbálja, és igen nagy biztonsággal kiválasztja a hibás alegységet. Még a teljes gép letesztelése sem tart tovább néhány másodpercnél, de általában egy DIAGNOSE utasitás végrehajtása csak néhány msec-ig tart. Ismertebb, mikrodiagnosztikával is rendelkező gépek pl. az IBM 360/30 és 360/85 [23].

A program utasitásainak szekvenciális végrehajtása sok esetben kedvezőtlen korlát a digitális számitógépekben. Ha pl. egy utasitás nem az előző müvelet eredményétől függ, (tehát pl. nem az előző müvelet eredményéből tovább számoló utasitás), akkor az őt megelőző utasitással egyidőben is végrehajthatnánk. Mivel azonban egyetlen processzor áll rendelkezésre, igy csak a szomszédos utasitások különböző végrehajtási fázisait tudjuk egyidőben elvégezni, és ezáltal csupán a szomszédos utasitások átlapolására van lehetőség. Részletesen: Az utasitáskiolvasó blokk pl. kiolvassa az i-edik utasitást és átadja a cimkiszámitó blokknak. Ez kiszámitja az i-edik utasitás operanduszának a cimét, és átadja az operandusz-kiolvasó blokknak, miközben az utasitáskiolvasó blokk már az (i+1)-edik utasitást olvassa ki és adja tovább a cimkiszámitó blokknak. Hasonlóképpen szimultán játszódik le az i-edik utasitás operandusz-kiolvasása, az (i+1)-edik utasitás operanduszának cimkiszámitása és az (i+2)-edik utasitás kiolvasása. Ha nemcsak a központi vezérlőegység, hanem az aritmetikai-logikai egység is hasonlóan autonóm funkcionális blokkokra van osztva (pl. a lebegőpontos összeadást végző egység karakterisztika-kivonó blokkra, mantissza-toló blokkra, mantissza-összeadó blokkra, poszt-normalizáló blokkra stb.), akkor az utasitások igen magas foku átlapolása és ezzel a hagyományos processzorokhoz képest jelentős teljesitménynövelés válik lehetővé. Ismertebb, ilyen elven működő gépek pl. a CDC-7600 és az IBM 360/195.

Az átlapolt processzorok hatásfokát azonban nagymértékben lerontja, ha sürün fordulnak elő olyan utasitások, amelyek az előző müvelet eredményére támaszkodnak, vagy pedig gyakoriak az előző müvelet eredményjeleitől függő ugró utasitások. Például az IBM 360/91 elméleti sebessége 70 millió müvelet másodpercenként, a gyakorlatban mért átlagos sebessége azonban -- az emlitett okok miatt -- csak 6 millió müvelet/sec [24].

Az átlapolt processzorok természetesen lényegesen bonyolultabbak a hagyományosaknál, ezért még a teljesitménynövekedéshez képest is aránytalanul drágák. Emiatt inkább csak olyan utasitás-sorozatok átlapolására törekedtek, amelyekben ugyanazt a müveletet kell sokszor, de különböző operanduszokon elvégezni. Itt a nyilvánvaló nehézség az operanduszok megfelelően gyors adagolása.

Több megoldás is született a probléma megoldására.

Az un. paralell-processzoros ILLIAC-IV számitógép [25] a tervek szerint teljes kiépitettségében 4x64 processzor-elemet tartalmaz majd, 1970-ig 64 processzor-elemet tartalmazó egyetlen kvadránsát épitették meg. A 64 processzor-elemet egyetlen központi vezérlő egység irányitja. Minden processzor-elemhez helyi memória is tartozik, amelyeket összességükben az ILLIAC-IV főtárolójának is tekinthetünk, bár a processzor-elemek közvetlenül csak saját memóriájukhoz tudnak hozzáférni. A központi vezérlő egység bármelyik processzor-elem memóriájához hozzáférhet közvetlenül, pl. a program kiolvasása céljából. A 64 processzor-elemet 8x8-as kétdimenziós (sikbeli) elrendezésben elképzelve, egy processzor-elem memória sorszerinti és oszlop-szerinti szomszédjával áll közvetlen kapcsolatban, igy az adatok viszonylag gyorsan átküldhetők az egyik processzor-elem memóriából egy tőle távolabbiba is. (A sorok ill. oszlopok első és utolsó helyen álló memóriái is közvetlen kapcsolatban állnak egymással, igy szigoruan véve a kétdimenziós sikbeli elképzelés nem igaz, helyette a tóruszfelület-elrendezés felel meg a valóságnak).

A processzor-elemek un. skeleton- (váz-) processzorok, csak az alapregisztereket és a szükséges minimális vezérlő logikát tartalmazzák. Ugyanabban az időben az összes processzor-elem csak ugyanazt a müveletet végezheti (esetleg nem mind dolgozik), az operanduszoknak a processzor-elem memóriákban szétosztottan és megfelelő sorrendben kell elhelyezkedniök. Ez a követelmény nagyon megnehezíti az ideálisan számitott müveleti sebesség elérését (ez kvadránsonként 240 millió összeadás/sec.).

A másik "mammut-gép" a számitógépek világában az un. pipeline (csővonal-) processzoros CDC STAR-100 (megépült 1971-ben [26]. Nevét a "STring and ARray" szervezésü adatok nagysebességü feldolgozása folytán kapta (kb. 100 millió operandusz/sec.). A memóriában sorba rendezett adatokat ennek a sebességnek megfelelő ütemezésben küldik a feldolgozó egységbe, amelynek a müködését csővonal-szerüen a legkönnyebb elképzelni. A csővonal egyes szakaszai az aritmetikai-logikai egység egyes funkcionális blokkjainak felelnek meg. A csővonalban sürü egymásutánban haladnak az operanduszok, mindegyiken az előirt müvelet más és más fázisát hajtják végre ugyanabban az időben az egyes blokkok. A csővonalból kiérkező müveleti eredményeket a memória szintén megfelelően nagy sebességgel képes fogadni.

A számitógépek közül a jelenlegi leggyorsabbnak tartott STARAN nevü gépet (500 millió müvelet/sec) a Goodyear Aerospace cég fejlesztette ki 1971-72-ben, katonai célra [26]. Ennél a gépnél az asszociativ memóriák szervezési elveit hivták segitségül az operanduszokhoz való gyors hozzáférés megoldására. A STARAN gép egy regiszter tartalma és egy tárolómodul összes cimének a tartalma között egyidőben képes egyszerübb müveleteket elvégezni, gyakorlatilag magában a memóriában. Tároló-moduljai ugyanis a következő szervezésüek: 256 szavasak,

szavanként 32 bitesek. Egy-egy modult azonban nemcsak soronként (32 bites szavanként) lehet kiolvasni, hanem oszloponként (256 bites "szavanként") is, igy egy modul kiolvasása a tiszta szó-szervezés esetén szükséges 256 ciklus helyett 32 ciklussal is lehetséges. Megemlitjük, hogy ezzel a megoldással pl. ugyanannyi processzor-elem kaphatna időben egyszerre operanduszt, mint egy teljes kiépitettségü ILLIAC IV, de bármelyik memória-modulból, ezért az operanduszok elhelyezése a főtárolóba sokkal egyszerübb lehetne.

2.3.3. Aritmetikai-logikai egység (ALU) strukturák

Eltekintve néhány (elsősorban oktatási célt szolgáló) gép soros működésü aritmetikájától, a digitális számitógépek aritmetikai-logikai egységére a párhuzamos működés a jellemző. (Ez a megállapitás szigoruan véve csak az összeadásra, a kivonásra, a léptetésekre és a helyértékenkénti logikai müveletekre igaz, a szorzásra és az osztásra csak ritkán érvényes).

Az alkalmazott számábrázolás leggyakrabban komplemens bináris vagy binárisan kódolt decimális, de pl. a többértékü logikai algebra területén folyó kutatások alkalmazásaként épitettek már ternáris aritmetikáju gépet (Szetuny, 1961), sőt a számrendszerekkel kapcsolatos vizsgálatok nyomán maradék-aritmetikás gépet is (EPOS-2, 1961). Az utóbbi időben terjed a redundáns (hibajelző és hibajavitó) kódrendszerek alkalmazása is.

Az 50-es évek végéig kizárólag egy-akkumulátoros aritmetikákat épitettek. Ez azt jelentette, hogy amennyiben egy számitási részeredményt nem közvetlenül a soron következő utasitás használt fel, azt be kellett irni az operativ memóriába, és a későbbi felhasználáskor onnan kellett kiolvasni.

A félvezetők elterjedésével az aritmetikai egységek működési sebességét sikerült jelentősen megnövelni. Ennek a kihasználását viszont erősen korlátozta az operativ memóriák viszonylag nagy hozzáférési ideje. Kézenfekvőnek látszott ezért egy gyors, kis kapacitásu elő-memória létrehozása, amelynek regisztereit a részeredmények gyors tárolására és -- később bármikor -- gyors kiolvasására használhatjuk fel.

Megjegyezzük, hogy a legujabb gépek gyors monolitikus integrált áramkörü főtárolója indokolatlanná teszi a több akkumulátor használatát. Más kérdés, hogy pl. az IBM System/370-es gépekben használatos un. általános regisztereket nemcsak akkumulátornak, hanem más célra (pl. bázisregiszternek, indexregiszternek) is fel lehet használni.

Az akkumulátorokként használt regiszterek szervezése többféle lehet: pl. véletlen elérésü, veremmemória-rendszerü, stb. Utóbbi esetben az akkumulátor-regiszterekre való hivatkozás egyszerübb (nem köt le külön mezőt az utasitásban), viszont az aritmetikai kifejezések kiszámitásának algoritmusára nagyobb sulyt kell fektetni.

A hardware veremmemória-igény más oldalról is jelentkezett: egyes fordítási módszerek alkalmazását könnyitette meg. De már -- mint korábban emlitettük -- az egymásba skatulyázott szubrutinok hivása és a rekurziv szubrutin-hivás is legegyszerübben beépitett veremmemóriával oldható meg. Elsősorban ezért használtak veremmemóriát pl. az English Electric KDF-9 és a Burroughs Corporation B 5000-es gépben.

A több-akkumulátoros gépek aritmetikai egysége annyiban tér el az egyakkumulátoros gépekétől, hogy a bemeneti és a kimeneti oldalon nem kapcsolódik fix regiszterekhez, hanem mindig az utasitásban megjelölt regiszterek tartalmán végzi el a kijelölt müveletet, és annak eredményét szintén az utasitásban megjelölt regiszterbe küldi. Az aritmetikai-logikai müveleteket tehát ugy foghatjuk fel, mint regiszterek közötti olyan átvitelek sorozatát, ahol az átviteli láncba beépülnek bizonyos "módositó" (komplementáló, összeadó, léptető, stb.) áramkörök. Bonyolultabb aritmetikai müveleteknél (szorzásnál, osztásnál) az elő-memória egyes regisztereit munkaregiszterekként is felhasználhatjuk. Ez a felfogás lényegében megegyezik a mikroprogramozással, és igy természetes, hogy a mikroprogramozott gépek aritmetikai-logikai egységének a felépitése követi a fentebb elmondottakat [27].

A mikroprogramozott processzorokban mindig megtalálható: egy regiszterkészlet bemeneti és kimeneti kapuáramkörökkel, egy ADDER áramkör az elemi müveletek (komplementálás, inkrementálás, összeadás, stb.) végrehajtására és egy SHIFTER áramkör az ezután következő esetleges léptetésekhez. A mikroutasitások egyes mezői a kimenő kapuáramköröket, az elemi müvelettipus kiválasztását (az ADDER munkáját), a léptetés irányát és mértékét, végül a bemenő kapuáramköröket vezérlik.

Ismeretes, hogy az átlagos müveleti sebesség döntően az összeadás sebességétől függ -- vagy ha ugy tetszik attól, hogy az ADDER milyen gyors. Ez viszont a választott aritmetikai algoritmus függvénye is. Számos módszert dolgoztak ki az az alapvető aritmetikai müvelet, az összeadás gyorsitására. A legismertebbek ezek közül: a rekurziv módszer, az aszinkron átvitelképzés, az átvitel-ugrás, az átvitelelőrelátás és a maradék-számrendszer alkalmazása [8], [28].

Néhány gépnél a nagyobb müveleti sebesség biztositása érdekében a szorzás és az osztás párhuzamositására is vállalkoztak. A szorzás és az osztás gyorsitásának a leggyakoribb módszerei a következők: a szorzó több jegyének egyidejü figyelése, a parallel szorzás a Dadda-féle vagy a carry-save (átvitel-tárolásos) technikával, a maradék-számrendszer alkalmazása, parallel osztás az un. redundás bináris kódrendszer alkalmazásával, stb.

2.3.4. A főtároló (operativ tároló) szervezésének fejlődése

Már az általános blokk-strukturák tárgyalásánál is utaltunk rá, hogy a memória különböző részeihez való szimultán hozzáférés megteremtése érdekében az operativ memóriát modulokra kezdték bontani. A multiprogramozás kapcsán jelentkező dinamikus tárkiosztási igény az operativ

memória még kisebb egységekre való bontását követelte meg, mégpedig ugy, hogy ezeket az egységeket (lapokat) fizikai cimtartománybeli elhelyezkedésüktől függetlenül lehessen egymás után füzni, folytonos logikai cimtartománnyá. Igy születtek meg a lap-szervezésü tárolók.

Fizikailag egy lap-szervezésü tárolónak semmiben sem kell különböznie a korábbi tárolóktól, hiszen az egyes lapokhoz való szimultán hozzáférés nem követelmény. Az egyetlen többlet az un. laptábla, amelynek sorai az aktuális logikai lapszám -- fizikai lapszám megfeleltetéseket tartalmazzák, bár a laptáblát is gyakran a főmemóriában alakitják ki, software eszközökkel.

Lap-szervezéssel megoldható a főtároló virtuális bővitése is, amelyre különösen nagyszámu felhasználó időosztásos kiszolgálásánál van szükség. Ilyenkor egy gyors háttértároló (rendszerint diszk, néha dob) szolgál az összes cimezhető logikai lap tárolására. A főtároló fizikai kapacitása ennél jóval kisebb lehet. Az éppen használni kivánt logikai lapot a supervisor megfelelő rutinja tölti be a fizikai tároló valamelyik lapjára, a laptáblán rögzitve egyuttal a logikai lapszám--fizikai lapszám kapcsolatot.

Szintén a multiprogramozás követelte meg a memória-védelem bevezetését. Az operativ memória védelme a következő célokat szolgálhatja:

- A rendszer-programok védelme a felhasználói programoktól, elsősorban véletlen felülirás ellen.
- A felhasználói programok és adatok védelme egymástól, felülirás és kiolvasás ellen is (program- és adat-védelem, adattitkositás, program-tulajdonjogfenntartás).
- A felhasználói programok védelme önmaguktól (pl. az egyes szegmentumok védelme egymástól felülirás ellen).
- A rendszer-programok védelme önmaguktól (pl. géphiba esetére).

Ezeknek megfelelően a memória-védelemnek különböző védelmi fokozatai vannak. Valamely processzor állapotban egy cimtartomány lehet:

- védetlen,
- felülirás ellen védett,
- csak végrehajtható, de mint adat nem olvasható ki,
- teljesen védett.

A memória-védelem megszervezése a fizikai cimtartományban vagy a logikai cimtartományban történhet.

A fizikai szintü memória-védelemnek a két legismertebb eljárása a cimhatárregiszteres és a kulcs-regiszteres memória-védelem.

Az első eljárásban határregiszter-párokkal jelölik ki az egyes programok vagy program-szegmentumok által használható összefüggő fizikai cimtartományokat. Alsó cimhatár-regiszterekként a báziscimregiszterek is felhasználhatók. Ennek az eljárásnak az elterjedését az akadályozta, hogy közvetlenül csak folytonos tartományok védelmére alkalmas. Ezért több szegmentumra bontott program esetén minden szegmentumhoz egy határregiszter-párt kell hozzárendelni, és a cimet, amelyre hivatkozás történt, a programhoz rendelt összes határregiszterpár tartalmával össze kell vetni.

A kulcs-regiszteres fizikai memória-védelem esetén a főtárolót fix, nem tul nagy terjedelmü blokkokra osztják, és mindegyikhez hozzárendelnek egy kulcs-regisztert. Ebben egy kulcsszó és a védelmi fokozatot jelző bitek tárolhatók. Valamely blokk egy cimére történő hivatkozáskor a programállapotszó védelmi kulcs-része összehasonlitásra kerül a blokk kulcs-regiszterének a tartalmával. Ha a két kulcs-szó különbözik egymástól (és egyik sem nulla), a védelmi fokozatot jelző bitek is megvizsgálásra kerülnek, és ha ezt is megsértené az utasitás végrehajtása, akkor müködésbe lép a memóriavédelem, a kiolvasás vagy beirás helyett megszakitást okozva.

A logikai szintü memória-védelem megvalósitásához jól felhasználhatók a laptáblák. Egy-egy sorukba ugyanis a logikai lapszám--fizikai lapszám kapcsolat mellett, a logikai lapszámhoz kapcsolódó védelmi kulcs-szó is elhelyezhető. Ha egy program utasitásaiban hivatkozás történik valamely logikai lapszámra, a fizikai lapszám kikeresésekor egyuttal a kulcs-szó is összehasonlitható a programállapot-szó védelmi kulcsrészével.

A következőkben a processzor és az operativ memória ciklusideje közti különbség feloldására irányuló tároló-struktura módositásokkal foglalkozunk. A félvezetők és különösen az integrált áramkörök elterjedésével a központi feldolgozómüben alkalmazható ciklusidő sokkal gyorsabban csökkent, mint az operativ memóriáké. Ez a tény mind jobban korlátozta a processzorok elméleti számitási sebességének a kihasználását. A problémát végül is a félvezető alapu integrált áramkörü főtárolók bevezetése oldotta meg, megkerülésére azonban addig is számos megoldás született. Ezeket két nagyobb csoportba oszthatjuk:

1. kiskapacitásu gyors segédmemóriák alkalmazása,

2. a memóriához fordulási ciklusok átlapolása

A gyors-segédmemóriák alkalmazásának legjellegzetesebb példája a scratch-pad memória. Erről az LSI tároló-blokkok alkalmazása kapcsán már szó esett, itt csak utalunk rá, hogy akkumulátor-regisztereiben gyakran használt operanduszokat vagy korábbi részeredményeket tárolhatunk, a főtároló rekeszeihez képest lényegesen jobb hozzáférési idővel.

A veremmemóriákról szintén esett szó a több-akkumulátoros gépekkel kapcsolatban. Több akkumulátor használatával azonban csak az operanduszok kiolvasásának az időszükségletét tudjuk csökkenteni, maguknak az utasításoknak a kiolvasásához szükséges időt nem. Ezért vezették be pl. az IBM 360/185 gépnél a slave- (szolga-) memóriát [29]. A slave-memóriák kapacitása általában néhány 10-től néhány 100 szóig terjed, a szóhossz az (utasítások hozzához képest) egy toldalék-résszel meg van növelve. A toldalék-rész annyival rövidebb egy fizikai cim hosszánál, mint ahány bit szükséges a slave-memória rekeszeinek a cimzéséhez. Amikor egy utasitást először olvasunk ki a főmemóriából, az utasitás cimének utolsó bitjeivel cimzett szolga-memória rekeszbe is bekerül (pl. 32 rekesz kapacitásu szolga-memória esetén az utasitás cimének utolsó 5 bitjével cimzett rekeszbe). Ennek a rekesznek a toldalék-bitjeire ugyanakkor bekerül az utasitás cimének a többi része. Egy utasitás kiolvasásakor a vezérlő egység először megvizsgálja az utasitás cimének utolsó bitjeivel cimzett szolga-memória rekesz toldalékbitjeit, hogy egyeznek-e az utasitás cimének többi részével, és ha igen, akkor nem a főmemóriából olvassa ki az utasitást, hanem a szolgamemóriából. Ha a toldalék-bitekkel való összehasonlitás nem-egyezést mutat, akkor ugyanaz játszódik le, mint egy utasitás első kiolvasásakor.

Ezzel az eljárással jelentősen meggyorsithatók a rövidebb programciklusok végrehajtása.

Egy másik módszer az utasitások kiolvasási idejének a csökkentésére a PLA (program-look-ahead, program-előrelátó) memória alkalmazása [25]. A PLA memóriák néhány szekcióból (szekciónként néhány szóból) álló gyorstárolók. A szavak ebben az esetben is toldalék-résszel vannak ellátva. Egy utasitás első kiolvasásakor nemcsak a kérdéses utasitás hanem -- egy szekció hosszának megfelelő számu --őt követő utasitás is átkerül a PLA memória valamely szekciójába. Ezek végrehajtása alatt a hardware asszociativ módon megvizsgálja, hogy az őket követő néhány utasitás bent van-e már valamelyik szekcióban, és ha nincs, akkor áthozza ezeket a főtárolóból egy éppen nem használt szekcióba.

Ennek az eljárásnak csak hosszabb szekvenciális program-részek végrehajtásánál van jelentősebb gyorsitó hatása.

A több-processzoros rendszereknek igen nagy (néhány Mbyte) főtároló-kapacitással kell rendelkezniök. Nagyon költséges lenne ezt nagysebességü modulokból megépiteni. Ezért inkább a processzorokhoz rendelnek hozzá gyors, de csak kb. 10 Kbyte kapacitásu helyi (un. cache-) memóriákat [26]. Ezekbe a végrehajtásra kerülő programokat blokkokban töltik át a főmemóriából. Működésük a szolga- (slave-) memóriákéhoz hasonló. Annak a valószinüsége, hogy a futáskor szükséges információ csak a főmemóriában található, néhány %. Ezért az effektiv tároló-hozzáférési idő ennél a módszernél viszonylag kis anyagi ráforditás árán jelentősen javul. Ilyen tároló felépitése van pl. a japán DIPS számitógép-hálózat központi szerepet betöltő HITAC-8400 tipusu gépeinek.

A memóriához fordulási ciklusok átlapolásához (interleaving) a főtárolónak feltétlenül több, szimultán is működtethető blokkból kell állnia. A memóriához fordulási igények ütközése két okból következhet be:

1. Az egység a ciklusidőnél rövidebb időn belül ujból a memóriához fordul (pl. utasitás kiolvasása után az operanduszért),

2. több független egység (pl. egy processzor és egy csatorna) közel egyidőben fordul a memóriához.

A több, egymástól függetlenül működő blokkból álló főtárolókkal ezek a problémák részben megoldhatók. Például ha lehetőség van rá, hogy külön blokkba kerüljenek egy program utasitásai, és ugyancsak külön blokkba az operanduszok, akkor az utasitás-kiolvasási ciklus átlapolódhat az operandusz-kiolvasási ciklussal. Ilyen lehetőséget nyujtanak többek között az IBM System/360-as gépek.

Egy másik lehetőség az interleaving-re, ha a páros és páratlan cimeket külön blokkba rendezzük, és gondoskodunk róla, hogy pl. páratlan cimen található utasitásban páros cimü operanduszra történjen hivatkozás. Ilyen tároló-elrendezése van pl. az ICL System-4/70-nek.

A következőkben a szóhossz (operandusz-hossz) szervezésének a fejlődéséről lesz szó. Az első univerzális elektronikus digitális számitógépek szó-orientált gépek voltak. Ez azt jelentette, hogy az operativ memória legkisebb egysége a szó volt, hosszusága géptipustól függően 24 és 48 bit között volt. Az operanduszok fix, egy szó (ritkábban két szó) hosszuságuak voltak, az utasitások ugyszintén.

Ez a szervezés rendszerint csak a nagy pontosságot igénylő tudományos feladatok számára volt kedvező. Hogy az adatfeldolgozási feladatoknál jelentkező legkülönfélébb hosszuságu operanduszok tárolása ne okozzon memóriapazarlást, változtatható szóhosszuságu rendszereket kellett létrehozni. Az ilyen un. mező-orientált gépeknek két változata terjedt el.

A jelzett karakteres (vagy mark-bites) mező-orientált gépek memóriájának legkisebb cimezhető egysége a karakter volt. Ez lehetett pl. 6 vagy 8 bites is. Minden karakterhez tartozott további egy vagy két mark- (jelző-) bit is. Pl. az IBM 1440-es vagy az IBM 7010-es gépnél egyetlen word-mark (szójelző) bit tartozott minden karakterhez. A szó utolsó karakterének word-mark bitje 1-es volt, a többié 0, igy az utasitásban csak a szó első karakterének a cimét kellett megadni.

A Honeywell 200-as gépnél például két bit tartozott minden karakterhez: a word-mark bit és az item-mark (tétel-jelző) bit. A kettő kombinációjával lehetőség nyilt teljes adat-rekordok szerkezetének a megadására is, pl.:

WM = 0, IM = 0 : belső karaktert jelez,

WM = 1, IM = 0 : szó végét jelzi,

WM = 0, IM = 1 : tétel végét jelzi,

WM = 1, IM = 1 : rekord végét jelzi.

A méret-specifikációs mező-orientált gépekben (pl. az RCA-3301, az NCR-315, az UNIVAC-1050) egy-egy tároló-referenciás utasitásban megadásra került a müveleti kód, az első karakter cime és a szóhoz tartozó karakterek száma.

Megjegyezzük, hogy a mező-orientált gépekben lehetőség van nemcsak az operandusz-hossz változtatására, hanem változó hosszuságu utasitások kialakitására is.

A fejlődést megszabó ujabb követelmény az volt, hogy a számitógépek legyenek alkalmasak akár elektronikus adatfeldolgozásra, akár tudományos feladatokhoz. Ezt a célt leginkább a byte-orientált gépek teljesitik (pl. Spectra/70, IBM/360), amelyekben mind fixált szóhosszt, mind változó mező-formát lehet alkalmazni. Az IBM System/360 utasitásrendszerében megtaláljuk pl. a félszó, teljes szó és dupla szó hosszuságu operanduszokkal dolgozó decimális aritmetikai utasitásokat és az 1-256 byte hosszuságu operanduszokkal dolgozó logikai utasitásokat. Az utasitások hossza 2, 4 vagy 6 byte lehet.

x x x

A számitógépek fejlődéstörténete során egyre nyilvánvalóbbá vált, hogy fejlettségi szintjüket, alkalmazhatóságukat és teljesitményüket nemcsak a hardware határozza meg, hanem döntő jelentőséget kell tulajdonitani a vele mindinkább összefonódó software-nek is. A hardware ugyanis csak lehetőségeket teremt a gépek hatékony felhasználásához, a gyakorlatban azonban csupán megfelelő programokkal ellátva válnak jól használhatókká. Ez nyilván abból következik, hogy éppen a programok realizálják és közvetitik valamilyen módon a gép "képességeit" a felhasználó felé. Már itt megjegyezzük, hogy ugyanahhoz a hardware-hez különböző software-ek készithetők, tehát tulajdonképpen a hardware és a software együttesen határozzák meg az információfeldolgozó rendszer tulajdonságait a felhasználó szempontjából.

Software-on olyan programokat és a hozzájuk tartozó dokumentációk összességét értjük, amelyeknek közvetlen célja nem a gép felhasználása feladatok megoldására (mint a felhasználói programoké), hanem a gép felhasználhatóbbá tétele feladatok megoldására.

Az alábbiakban először megtárgyaljuk az utasitásrendszerek és a gépi szintü programozási technikák fejlődését. Ezt követően, a kezelőrendszerek, majd a programozási nyelvek és az azokat értelmező eszközök fejlődésével foglalkozunk.

2.4. A gépi utasitásrendszerek és programozási rendszerek fejlődése

A tudománytörténeti jelentőségü Neumann-Burks-Goldstine jelentés (1946, [5]) lefektette a mai értelemben vett számitógépek fejlesztésének alapjait és egyben hosszu időre meghatározta fejlődésüket. A jelentés — egyebek között a tárolt program elvével — igen hatékony szerepet játszott a software fejlődés elindulásában.

2.4.1. A tárolt program elve

A régebbi berendezéseknél (pl. a lyukkártyás gépeknél) az információfeldolgozás menetét (programját) teljesen különállóan kezelték a feldolgozandó adatoktól. A programok (pl. dugaszolt, fixen huzalozott programok, lyukkártyán, lyukszalagon rögzitett programok) a feldolgozás folyamán állandóak maradtak, a változtatásra semmilyen lehetőség nem volt. Ez a körülmény egyrészt igen nehézkessé tette az algoritmusok gépre vitelét, másrészt erősen korlátozta körüket. Nem volt lehetőség pl. arra, hogy az algoritmusokban kisebb eltérésekkel sokszor ismétlődő részeket a gép -- csekély változtatások eszközlésével -- ugyanazon programrész alapján hajtson végre (csupán változatlan program-elemeket lehetett ciklikusan végrehajtani).

A tárolt program megjelenése és alkalmazása megoldotta ezt a problémát, mivel ugyanolyan müveletvégzési lehetőségeket biztositott a program elemi részein, mint a feldolgozandó adatokon. (Emellett egyéb haszna is volt: pl. egyszerüsitette a programok "bevitelét" a gépbe, továbbá megszüntette a programok számára szolgáló külön tárolóegység-szükségletet.) A tárolt program elvének alkalmazásával tul lehetett lépni a direkt programozáson és az utasitásmódositás-nélküli ciklikus müveletvégrehajtáson. Az utasitások módosithatósága azután lehetővé tette bonyolultabb ciklusok leirását is. Ameddig egy ciklusban csak az adatok helye változik a végrehajtás során, csupán az utasitások cimrészének módositására vonatkozó igény lép fel. Most viszont lényegesen tágabbá váltak a lehetőségek, mivel az utasitások müveleti részét is meg lehetett változtatni. Az utasitásokon (tehát végeredményben a programokon) végezhető müveletek lehetővé tették a programozás megkönnyitését (a gyorsan fejlődő automatikus programozási eszközökkel), majd pedig a feladatok és a rendszer kezelésének automatizálását is (az operációs rendszerekkel). Ezek részletes tárgyalását megelőzően azonban röviden foglalkoznunk kell a gépi utasitások fejlődésével.

2.4.2. A gépi utasítások fejlődése

A részletesebb tárgyalásban elsőnek az utasítások formai fejlődését jellemezzük, mellőzve egyenlőre az I/O utasításokat, amelyekről később lesz szó.

Az egy-akkumulátoros szóorientált gépek utasításrendszere fix hosszuságu, tároló-referenciás (tárolóra hivatkozó) utasításokból állt. Az utasításokban megadták a müvelet kódját, a cim(ek) módositására vonatkozó információkat és egy, kettő, vagy három cimet.

Az egy-cimes utasításokban a müvelet egyik operanduszát az akkumulátor-regiszter tartalma, a másikat a módositott cim tartalma szolgáltatja. Ha a müvelet eredményét nemcsak az akkumulátorban, hanem a memóriában is tárolni kellett, azt rendszerint egy külön utasitással lehetett elvégezni.

A két-cimes utasításrendszerekben a müveletet, vagy az akkumulátor és az első cim tartalma, vagy a két cim tartalma között lehetett elvégezni. A müvelet eredménye az utasításban megadott második cimre volt beirható.

Az 1+1 cimes utasitások végrehajtása ugyanugy történt, mint az egycimeseké, a második cimmező a soron következő utasitás cimének a megadására szolgál. A szekvenciális végrehajtástól való eltérésre az operativ dobmemória használata miatt volt szükség, ugyanis a soronkövetkező utasitás kiolvasását meggyorsitotta, ha az közel volt elhelyezve (a dob forgásával ellenkező irányban)a dob azon alkotójához, amelyen az olvasófej állt, amikor az előző utasitás végrehajtása befejeződött. Az egyes utasitások végrehajtási idejének figyelembevétele azonban jelentősen megnehezitette a programozást.

A három-cimes utasitás-rendszerekben a müveletet vagy az akkumulátor és az első cim tartalma, vagy az első két cim tartalma között lehetett elvégezni. A müvelet eredménye az akkumulátoron kivül a harmadik cimre is bekerülhetett.

A 2+1 cimes utasitások végrehajtása a két-cimesekéhez hasonlóan történt. A harmadik cimmező a soronkövetkező utasitás cimét adta meg. Céljuk és egyben hátrányuk hasonló volt, mint az 1+1 cimes utasitásoké.

A korszerübb, harmadik generációs gépeknél az utasitásoknak mind formai, mind tartalmi szempontból rugalmasabbakká kellett válniok. Emiatt az egy-egy gép utasításrendszerének jellemzésére használatos, az előbbiekben ismertetett osztályozás -- részben merevsége miatt, részben pedig azért, mert nem tartalmazhatott minden lényeges jellemzőt -- már egyre kevésbé felelt meg. Figyelembe kellett venni ugyanis egyrészt, hogy az akkumulátorok többszöröződtek, másrészt, hogy a mezőszervezésü majd byteszervezésü gépek utasitásrendszerében az utasitások (hasonlóan az adatokhoz) változó hosszuságuakká váltak. Az ujabb osztályozásban (amelynél az utasitások hosszát rendszerint a müveleti kódhoz kapcsolt néhány bittel adják meg) a szokásos utasitástipusok a következők:

RR tipus (általános alakja:Müv.kód R_1 R_2): két akkumulátor-regiszter tartalma közt végez müveletet (un. regiszter-referenciás, regiszterre hivatkozó utasitás).

RX tipus (Müv.kód R_1 XA_2, X az indeX szóra utal): egy akkumulátor és egy indexeléssel kialakitott főtároló-cim tartalmán végez müveletet. Az eredmény rendszerint R_1-be kerül. (Tárolásra utaló müveleti kód esetén az R_1 tartalma kerül át a memóriába.)

RS tipus (Müv.kód R_1 R_3 A_2, S a Storage szóra utal): elsősorban csoportos áttöltő utasitásként használatos. Pl. a csoportos betöltésre utaló müveleti kód esetén a memória tartalma A_2-től kezdődően átkerül az akkumulátor-regiszterek egy csoportjába. Az első akkumulátort R_1, az utolsót R_3 jelöli.

SI tipus (Müv.kód I A_2): az I-mező un. közvetlen (immediate) operandusz tartalmaz, ezzel, valamint az A_2 cim tartalmával végez müveletet. Az eredmény a tárolóba kerül, az A_2 cim tartalma helyére. A közvetlen operandusz-megadást szokás literális cimzésnek is nevezni.

SS tipus (Müv.kód L A_1 A_2): az A_1 és A_2 kezdőcimü, (L+1) byte hosszuságu operandu szokon végez müveletet. Az eredmény rendszerint az első operandusz helyére kerül.

Ennél az osztályozásnál szigoruan véve nem lehet az utasitásokat tároló-referenciás és regiszter-referenciás utasitásokra osztani, hanem az utasitásokban egyes mezők tartalmazhatnak tároló-referenciás cimeket, regiszter-referenciás bitcsoportokat, vagy közvetlen operandu szokat.

2.4.3. Cimzési rendszerek és eljárások

A programozási módszerekben elért haladás egyre nyomatékosabban vetette fel a cimzési eljárások fejlesztésének igényét, amit tovább sürgetett a programozási munka megkönnyitésében megtett következő lépés: a szubrutinok használata. Ezek -- mint előre elkészitett programrészletek -- több helyen felhasználhatók a programokban, vagy ugy, hogy minden felhasználás helyén többé-kevésbé változatlan formában beiktatják őket (nyitott szubrutinok), vagy ugy, hogy csupán egy helyen kerülnek beépitésre, de vezérlés-átadással olymódon aktivizálhatók, hogy -- a paraméterek átadásának és átvételének valamely ismert módját alkalmazva -- végrehajtás után a vezérlést visszaadják az őket aktivizáló programnak (zárt szubrutinok).

A fentiek alapján jól látható a cimek kezelésének fontossága, hiszen mind a ciklikus programozásnál, mind a szubrutinok alkalmazásánál egyes kijelölt tevékenységek azonosak maradnak a különböző vég-

rehajtások során. Ez azt jelenti, hogy az utasításkódok nem változnak, de az operanduszok, tehát a nekik megfelelő címrészek rendre mások és mások lesznek, ami kényelmesen megvalósítható az indexeléssel. Az indexelésnek az a lényege, hogy az utasításban megadott címet az ugyancsak az utasításban specifikált indexregiszter tartalmával módosítják, (amig kell), és ezt a módosított címet használják fel a művelet elvégzésekor. Az indexregiszterek száma gépenként jelentősen eltér. Egyes gépeken egyáltalán nincs indexregiszter, másokon e célra speciális hardware-regiszterek használhatók, a fejlettebb megoldásokban pedig a központi tároló meghatározott rekeszei szerepelnek indexregiszterként (a hardware-indexregiszterekkel ellentétben ezeken tetszőleges műveletek végezhetők). Ez utóbbi esetben is számottevő különbségek találhatók a kijelölt rekeszek számában és elhelyezkedésében (pl. a MINSZK-22-n az első 17_8 rekesz, az M-3-on minden memóriarekesz, az ELLIOTT-803-8-n a gépi szavak első fele használható indexelésre a szavak második részében elhelyezkedő utasításokra nézve).

A hardware által indexeléssel elvégzett művelet természetesen indexelési lehetőségekkel nem rendelkező gépeken is végrehajtható (programozási eszközök segítségével), mivel a tárolt program lehetővé teszi bármely adott utasítás módosítását. Igy pl. az EDSAC gép első változata az indexelés helyett a program beolvasása során, amely utasításokat módosítva beolvasó szubrutin segítségével történt, került sor a címrész megfelelő módosítására. Nyilvánvaló, hogy a software-megoldás nehézkessége és lassusága volt a tevékenység hardware-esítésének kiváltó oka. Az indexregiszterek alkalmazása nagy előrelépést jelentett mind a ciklikus programozásban, mind a szubrutinkezelésben. Többek között megoldotta a tömbök kezelésével kapcsolatos ciklikus programozási problémákat. A software további fejlődése azonban olyan ujabb igényeket támasztott, amelyek már tulmutattak az indexelési technikán.

Indexelésre felhasználhatók az indirekt címzési eljárások is, de mivel ezek általánosabb célokat is szolgálnak, most külön részletesebben foglalkozunk velük.

Az indirekt címzési eljárások kialakulásának kiváltó okaként azt tekinthetjük, hogy az egyre bővülő kapacitásu operativ tároló direkt címzése az utasításban szereplő címzéssel mind inkább a hatékonyság rovására ment. Az indirekt címzési eljárásokban szereplő cim tartalma nem az operanduszt, hanem az operandusz címét szolgáltatja, és mivel a szavak hossza lényegesen meghaladja a címrészek hosszát, ezért ilyen módszerrel jóval nagyobb fizikai címtartományt is át lehet fogni. Az indirekt címzés többszintü is lehet, pl. kétszintü indirekt elemzésnél az operandusz címét annak a rekesznek tartalma szolgáltatja, amelynek cime az utasításban szereplő cim tartalma.

Az indexregiszteres és indirekt cimzés egy utasitáson belüli használatához a gépi utasitásokban egy ujabb mezőt kellett kialakitani, amelyben a cimzési mód irható le. Ilyen cimzési módok lehetnek ezután:

- direkt cimzés (a cimmezőben rögtön a tényleges cim szerepel),
- direkt relativ cimzés (az utasitásszámláló, vagy egy akkumulátor szerepelhet indexregiszterként, az előbbi esetben önrelativ cimzésről beszélünk),
- indirekt cimzés (egy vagy több szintü),
- indirekt relativ cimzés (ilyenkor elő- és/vagy utó-indexelés is lehetséges, azaz indexelés az indirekt cimzés előtt és/vagy utána.)
- akkumulátoron át történő cimzés (az abszolut cimet egy vagy több előző utasitás akkumulátor-regiszterben alakitja ki és ennek tartalma adja az operandusz-cimet).

Az indirekt cimzésnek ma különösen rövid szóhossz esetén (általában kis gépeknél) van jelentősége, ahol igy nagyobb főtároló-kapacitás esetén is külön segédregiszterek (pl. báziscim-regiszterek) nélkül tudunk hivatkozni bármely főtároló-cimre. Ilyen eljárás például az "indirekt lapcimzés" is, amelynél direkt módon csak a memóriatartomány legelejére (a "0-dik lapra") vagy az utasitás cimének a környezetére, vagy pedig az előzőleg végrehajtott utasitásban használt operandusz-cim környezetére hivatkozhatunk, de indirekt módon a "0-dik lap" valamely rekeszén keresztül a teljes operativ memória bármelyik cimét elérhetjük. Ebben az esetben a "0-dik lap" kérdéses rekesze a báziscim-regiszter szerepét játssza, amelynek a tartalmát utóindexeléssel még módosithatjuk. (Az utasitás cimének a környezetére való direkt hivatkozás "közelbe" való vezérlésátadás, vagy közeli utasitás módositása esetén célszerü. Az előzőleg végrehajtott utasitásban használt (abszolut) operandusz-cim környezetére való direkt hivatkozás a memória azonos, de a program utasitásaitól különböző lapján elhelyezett adatok sorozatos feldolgozása esetén célszerü.)

A szubrutinkönyvtárak kialakulása és az automatikus programozási rendszerek fejlődése -- további lépésként -- megkövetelte, hogy egy adott (valamilyen formában megirt) program ne legyen szigoruan a memória valamely helyéhez kötve, hanem különböző helyeken lehessen azt felhasználni. Ez a lehetőség a forrásprogramok szintjén lényegében biztositva van, hiszen a forrásprogramokban általában nincs rötzitve a memória-hozzárendelés, ezt csak a fordítóprogramok végzik el. A programok irásakor tehát (eltekintve a gépi kódu, abszolut cimes programoktól), biztositva van a változók, cimkék (utasitások) helyének logikai kezelése, azaz ekkor még nincsenek hozzárendelve valamely fizikailag meghatározott helyhez (memóriarekeszhez). Az természetesen nyilvánvaló, hogy a programok végrehajtásakor már meg kell lenni a fizikai helyhez való hozzárendelésnek, mivel a müvelet-

végzés fizikai elemeken (azok tartalmán) történik. A feldolgozás folyamán tehát valamikor meg kell történnie a logikai és fizikai címek megfeleltetésének, azaz a logikai címeket le kell képezni a fizikai címekre.

Az első fordítóprogramoknál ez a hozzárendelés a fordítás elvégzésekor történt meg, s ennek eredménye abszolut cimes gépi kódu program volt. Természetesen a fordítóprogramoknak a szubrutinkönyvtárban szereplő programokat relativ (tehát logikai) cimes programokként kellett kezelniök, mivel azok a tárgyprogramokban más-más helyre kerültek (bár egyes rendszerekben a standard programok a központi tárolók előre kijelölt részén helyezkedtek el, mint pl. az ELLIOTT autókódjánál).

A következő fejlődési fok az volt, hogy a gépi cimre való fordítást két lépésben végezték el. Az első lépésben egy, a gépi nyelvhez közelálló nyelvü tárgyprogram az eredmény, melyet könyvtárban (modulkönyvtár) helyeznek el. Ez lehetővé teszi a kész tárgyprogramok felhasználását, beépitését más programokba. A második lépésben valósul meg a modulok összefüzése (linking) valamint betöltése (loading), és végbemegy a gépi (fizikai) cimekre való leképezés. Ennél a technikánál tehát a fordításkor nem szükséges rögziteni az elhelyezést, mert a betöltési idő során (egy rövid munkafázisban) van erre lehetőség.

Az előzőkre jellemző, hogy a fizikai cimek valamely, a végrehajtás megkezdése előtti fázisban kerülnek rögzitésre, az utóbbiban azonban csak közvetlenül a végrehajtás előtt. Ennek az az előnye, hogy a leforditott, előkészitett programmodulok a memória különböző helyein is felhasználhatók, áthelyezhetők (relocation), azonban a módszernél csak statikusan (static relocation), tehát a betöltés után már nincs lehetőség változtatásra. Ez a memória hozzárendelési forma az időosztás nélküli rendszereket jellemzi, bár segitségével időosztásos (multiprogramozásos vagy időszeleteléses, lásd később) rendszerek is készithetők, ám ezek praktikus értéke csekély.

Az összefüzés és a betöltés idején software-technikával történő memóriahozzárendelés előrelépést jelentett. Segitségével megoldhatóvá vált a programok szegmentálása. Ez utóbbi esetben a tárgyprogram különböző modulokból épül fel, s ezek mindegyike önálló cimtartománynyal rendelkezik, ami -- a forrásprogram szintjén -- kétdimenziós logikai cimzési rendszert eredményez.

Mig a fentebb vázolt előrelépés software jellegü volt és a fordítóprogramokkal, valamint a tárgyprogramok célszerü kezelésével kapcsolatban vetődött fel, addig a dinamikus áthelyezés (dynamic relocation), amelyet az időosztásos kezelőrendszerek megjelenése helyezett előtérbe, (nagyrészt) hardware eszközök segitségével volt megoldható. A különböző hardware eszközök egyuttal különböző szinteket is képviselnek, ahol a magasabb szinteken általában felhasználják az alacsonyabbakat.

Megkülönböztetünk:

- bázisregiszteres,
- lapcimzéses és
- szegmentum-cimzésos rendszereket.

Bázisregiszterek alkalmazása esetén az összeállitott program lényegében egy statikus betöltési eljáráson megy keresztül, de végrehajtásakor az összes tároló-referenciás cimekhez hozzáadódik a bázisregiszter tartalma, és az igy kapott cim kerül felhasználásra (miközben természetesen más indexelési lehetőségek is megőrződnek). Igy a program áthelyezése végrehajtás közben is (dinamikusan) megtörténhet, mivel ez nem von maga után változtatásokat, csupán a bázisregiszter tartalmának megváltozását igényli. Ezzel a technikával a program számára a memória tetszőleges helyén folytonos fizikai cimtartomány jelölhető ki, melyen egyetlen egységként helyezkedik el. Egyes rendszerekben egy-egy program számára több bázisregisztert is biztositanak, ekkor pl. nemcsak a program, hanem az adatok számára is külön folytonos területek jelölhetők ki. A futtatásra kész program cimei ezáltal logikai cimekké alakulnak át, a fizikai cimekre való leképezés csak a végrehajtás során megy végbe.

A bázisregiszterek alkalmazása azonban az emlitett előnyök mellett néhány hátránnyal is jár. A feldolgozások során (még változó memória-felosztást feltételezve is) több felhasználatlan memóriaterület képződik, mivel nem várható, hogy a felszabaduló egybefüggő helyek teljesen kitölthetők lesznek ujonnan belépő programrészekkel. Amennyiben ezeket is fel akarjuk használni, állandóan gondoskodni kell a programok fizikai áthelyezéséről. A logikai cimtartomány terjedelme is korlátozott: nem lehet nagyobb a fizikai cimtartományénál, mivel a logikai cimtartománynak a központi tároló folytonos szelete felel meg [30]. Ezek a fogyatékosságok természetesen csak további igények szempontjából merültek fel, software oldalról, egyébként a bázisregiszterek alkalmazása jelentős előrelépés volt.

A bázisregiszterek megjelenésével lehetőség nyilt olyan eljárások, rutinok készitésére, amelyeket szimultán használhatnak különböző programok. A felhasználás – amennyiben a rutinok megfelelő feltételeket teljesitenek -- meglehetősen bonyolult lehet, pl. mielőtt valamely rutin az egyik program számára befejezné a tevékenységét, valamilyen megszakitás folytán egy másik program is beléphet ugyanazon rutinba (pure procedure, reentrant code [21], reentrant program [30].)

A lapcimzéses (paging) rendszereknél (amelyeket a tároló jobb kihasználása végett kezdtek alkalmazni), a memória fizikai blokkokra (lapokra) van osztva, a programok pedig ezekkel azonos méretü logikai lapokra. A programban szereplő cim lényegében két részből áll (a lapszámból és a lapon belüli sorszámból), ami azonban a felhasználó számára egyetlen egységet képez. A végrehajtás során a logikai cimben szereplő logikai lapszámhoz ugynevezett laptábla segitségével tör-

ténik meg a fizikai lap (blokk) hozzárendelése, majd ezen belül a megjelölt sorszámu tárolórekesz kiválasztása. A laptábla egy programra vonatkozóan a logikai lapszámoknak megfelelő blokkok sorszámait tartalmazza. Kezdetben egyetlen laptáblán kellett osztozkodniok a különböző programoknak, később általánossá vált külön laptáblák hozzárendelése az egyes programokhoz, ahol a laptábla kiválasztása egy laptábla-bázisregiszter segitségével történik meg.

A lapcimzés alkalmazásával lehetővé vált a szabad területek összefüzése (a laptáblák megfelelő módon való kitöltésével), miközben a programokat nem kellett menetközben fizikailag áthelyezni. A lapcimzés bizonyos kiegészitéssel lehetővé tette, hogy a programok csak azt a memóriaterületet foglalják le, amit egy adott időben ténylegesen használnak, és ezzel együtt biztositotta a logikai cimtartomány kiterjesztését a fizikai cimtartományt meghaladó mértékben (virtuális memóriakezelés). Ez a fogás leegyszerüsitette, sőt esetenként megszüntette az átfedési (overlay) technikával kapcsolatos nehézségeket. Az emlitett technikánál ugyanis a program különböző részei felváltva helyezkednek el ugyanazon memóriaterületen. Nincs azonban lehetőség arra, hogy közvetlenül hivatkozzunk a memóriába nem tartózkodó részekre, és csak software eszközök segitségével kezdeményezhető és valósitható meg a programrészek cseréje. A demand paging rendszerekben a laptábla egy jelzéssel egészül ki, mely arra mutat, hogy az adott lap az operativ tárolóban tartózkodik-e vagy sem. Amennyiben a hozzáforduláskor a hivatkozott lap (e jelzés szerint) nincs az operativ tárolóban, bekerül egy éppen nem használt lap helyére, miközben ez a tevékenység, illetve ennek eredménye, átvezetődik a laptáblákon.

A bázisregiszterekkel kapcsolatban felvetett két probléma megoldása során mind a báziscimzéses, mind a lapcimzéses rendszereknél a forrásnyelvi szinten kétdimenziós cimzésnek a betöltés után egydimenziós logikai cimzés felel meg. Ez azonban hátrányos abban az esetben, ha az összeállitott program egyes részei a végrehajtás során változó terjedelmüek, ilyenkor ugyanis számukra a maximális helyet kell biztositani az egydimenziós cimskálán.

Ezt a hátrányt megszünteti, továbbá kedvezőbb lehetőségeket biztosit a jelenleg legfejlettebb cimzési eljárás a szegmentum-cimzés [30]. Ilyenkor a programot szegmentumokra bonthatjuk, és minden szegmentumban ujra elölről kezdhetjük a lapszámozást. Egy logikai cim alakja ilyenkor a következő: szegmentumszám, szegmentumon belüli logikai lapszám, lapon belüli relativ cim. Minden programnak saját szegmentumtáblája, és minden szegmentumnak saját laptáblája van. Az éppen futó program szegmentumtáblájának a kezdőcimét a szegmentumtábla-báziscimregiszter tartalmazza. Az átcimzési folyamat a következő: először az indexelés zajlik le, amely azonban nem érinti a szegmentumszámot. Ezután a szegmentumtábla-báziscimregiszter tartalmához hozzáadódik a szegmentumszám, az igy kapott szegmentumtábla-cimről kiolvasásra kerül egy laptábla báziscime. Ehhez hozzáadódik az indexelt logikai lapszám, és az igy kapott laptábla cimről kiolvasásra kerül a fizikai lapszám. Ezzel konkatenálódik a lapon belüli indexelt cim, és ez adja végül az abszolut fizi-

kai cimet, amelyhez a hozzáfordulás történik.

Mind a lapcimzés, mind a szegmentumcimzés lényegesen összetettebb a korábbiaknál, s ez azzal a következménnyel jár, hogy lelassult a cimkiszámitás. Ha ugyanis a szegmentumtáblák és alaptáblák az operativ memóriában helyezkednek el, akkor pl. egyetlen operandusz kiolvasásához háromszor kell a memóriához fordulni. Ezért ajánlatos lenne olyan asszociativ táblát alkalmazni, amelynek egyik oszlopában a szegmentumszámok és a logikai lapszámok, másik oszlopában a nekik megfelelő fizikai lapszámok vannak.

Ilyenkor cimzésnél egy szegmentumszám-bázisregiszter tartalmához adódik hozzá a szegmentumszám, kialakitva egy abszolut szegmentumszámot, mig az abszolut logikai lapszám indexeléssel alakul ki. Ha az igy kapott szegmentumszám--logikai lapszám kombináció megtalálható a tartalom-cimzésü tábla első oszlopában, a hozzátartozó sorból kiolvasásra kerül a fizikai lapszám, amellyel konkatenálódik a lapon belüli indexelt cim. Mivel az asszociativ tábla viszonylag gyors lehet, ez a cimzési eljárás nem növeli lényegesen az operandusz hozzáférési idejét. Minthogy pedig a szegmentumok cimtartománya külön-külön változtatható, amellett, hogy megőrzik a lapcimzés minden előnyét, lehetőséget biztosit a programrészek, vagy adatstrukturák terjedelmének dinamikus bővitésére, vagy szükitésére.

A szegmentumcimzés a GE-645 és az IBM 360/67 gépeken jelent meg [35].

Igen rugalmas tárolórendszert kapnánk abban az esetben, ha az asszociativ tárolókat (melyeknek rendkivül bonyolult a szelekciós rendszerük) gyors és nagykapacitásu főtárolóként is sikerülne alkalmazni. Talán a holográfiának, mint hardware módszernek az alkalmazása eredményekre vezethet ezen a téren is. A közeljövőben azonban még nincs komolyabb kilátás asszociativ cimzésü főtárolókra.

A cimzési rendszerek fejlődése is amellett tanuskodik, hogy a software igények kielégitésére az ujabb hardware-megoldások a software-ben többé-kevésbé meglévő eljárások "hardware-esitése" utján jöttek létre.

2.4.4. Az I/O utasitások és tevékenységek fejlődése

A számitógépes információfeldolgozás leglassubb eszközei már a kezdeti időszaktól fogva az I/O berendezések voltak. A fejlődés során a központi egységekhez viszonyitott sebességi arány tovább romlott az I/O eszközök rovására. Tekintettel arra, hogy (kivéve a tudományos és müszaki számitásokat) a feldolgozások nagy hányadánál jelentős részt képeznek az I/O tevékenységek, a korábbi I/O kezelés javitására uj elveket kellett kidolgozni a feldolgozás hatékonyságának megnövelése érdekében.

A kezdeti időszakot a perifériák közvetlen, egyedi kezelése jellemezte. Az utasitások elemi, vagy periférián lévő inputanyag által meghatározott, pl. tömbbeviteli tevékenységeket inditottak el, miközben a központi processzornak várakoznia kellett az I/O tevékenység befejeződéséig. Nyilvánvaló, hogy a központi egység működésének meggyorsulásával egyre nagyobb probléma lett a várakozás.

A megoldásra irányuló erőfeszitések egyik eredménye az I/O csatornák bevezetése volt. Az I/O csatorna egy hardware eszköz, melynek segitségével a tevékenység elinditása után a központi processzortól függetlenül oldható meg az I/O eszköz(ök) vezérlése. Megkülönböztetünk programozott és autonóm (multiplexor vagy szelektor) csatornákat. Az előzők esetében csak elemi I/O tevékenységek végezhetők párhuzamosan és autonóm módon (ez rávilágit a terminológia megtévesztő voltára, mivel itt is autonóm működésről van szó), az utóbbiak esetében viszont az összetettebb tevékenységet leiró (csatorna) programok hajthatók végre a központi számitási tevékenységgel párhuzamosan.

A fejlődés ezen a területen olyan utasitások kidolgozásával indult el, melyek képesek az átviteli tevékenységek elinditására, majd egy későbbi időpontban a csatorna lekérdezésére annak megállapitása végett, hogy befejeződött-e már az átvitel. Ezzel elvileg lehetővé vált a feldolgozási folyamat olyan szervezése, amelyben (kiküszöbölve a kihasználatlan várakozási időket) az I/O tevékenység teljes egészében átlapoltan folyik a központi processzor munkájával. A bevezetett ujitások azonban a gyakorlatban lényegesen bonyolultabbá tették az I/O tevékenység programozását, mivel a programozónak figyelmet kellett forditania a pufferezésre (az átvitel ugyanis nem az elsődleges tárolási területről(re) célszerű, hanem egy speciálisan fenntartott területről(re), ezzel tudniillik kiegyenlitetté tehető az I/O folyamat), a területek blokkolására, feloldására és az autonóm csatornákkal való szinkronizálásra. Ez ujabb nehézségeket jelentett amelyet I/O rendszernek (Input-Output Control System: IOCS) nevezett szubrutincsomagokkal igyekeztek megoldani. Az I/O rendszerek alkalmazása először az USA-ban terjedt el, később általánossá vált.

Az I/O tevékenységek hatékonyabb kezelésére kialakult másik irányzat a lassu I/O müveletek off-line elvégzése. Ennél valamilyen (hardware-) eszköz felhasználásával valósitják meg az átvitelt a lassu működésü periféria és mágnesszalag (később mágnestárcsa) között, függetlenül a központi egység munkájától. A programok végrehajtása során a hozzájuk tartozó I/O müveletek csak ehhez az eszközhöz, mint közbeeső külső tárolóhoz fordulnak (erre a technikára a későbbiekben még visszatérünk). Az előzőekben leirt eszközök felhasználásával készültek el az első operációs rendszerek, kötegelt feldolgozása (Share Operating System, az IBM 709-re, 1959 [31].)

A következő lényeges előrelépés a megszakitórendszerek megjelenése volt. A korábbiakban már utaltunk arra, hogy milyen nehézségekbe ütközött a program futásának és az I/O csatornák működésének szinkronizálása. Amennyiben az utasitások természetes sorrendjétől csak vezérlésátadó utasitásokkal térhetünk el (tehát a program által előre meghatározott módon), rendkivül nehézkesen lehet összehangolni a futást külső eseményekkel (pl. egy I/O tevékenység befejeződésével). Az összehangolásra kezdetben kényszerüségből software eszközöket alkalmaztak. Magasabb szintü megoldást a megszakitórendszerek alkalmazása jelentett természetesen ujabb software eszközök felhasználásával. A megszakitórendszernek az a lényege, hogy bizonyos külső események bekövetkezésekor (az eseménnyel kapcsolatban álló eszközről érkező megszakitójel hatására) a végrehajtási sorrend eltér a program által meghatározott utasitás-sorrendtől, és a vezérlés olyan programnak adódik át, amely értelmezi és feldolgozza a megszakitójelet, majd miután elvégzi a szükséges tevékenységeket, visszaadja a vezérlést a megszakitott programnak. Megszakitáskor a későbbi folytatáshoz szükséges regiszterek tartalma tárolódik, hogy a vezérlés visszavételekor az eredeti program zavartalanul működhessen tovább. Természetesen most csak egy leegyszerüsitett sémán mutattuk be azt a hardware adta lehetőséget, amelyet a felügyelő programoknak kell kihasználni. A kezdeti alkalmazásoknál a transzfer tevékenység befejeződésével megjelenő megszakitójel hatására a csatornát kezelő csatornaprogramnak adódott át a vezérlés, majd az I/O tevékenység (igény) kiszolgálása után ismét vissza a megszakitott programnak. (Ezáltal felesleges volt a csatorna lekérdezése annak megállapitása végett, hogy befejeződött-e már az átvitel.) Ezt az elvet eleinte csak egy programon belül alkalmazták, később azonban a multiprogramozásos rendszerek alapjává vált.

A megszakitó rendszerekhez kifejlesztett I/O rendszereknél szükségtelen, hogy a programozó közvetlenül forduljon az I/O eszközökhöz, mivel ezek kezelését rendszerprogramok végzik. Sőt, az esetleges zavarok elkerülése érdekében egyenesen célszerü volt megtiltani a felhasználók számára a hozzáfordulást, mivel a rendszer által kezelt perifériákról a felhasználó általában nem tudhatja, hogy pillanatnyilag milyen tevékenységet végeznek, mi az aktuális feladatuk, igy a közvetlen használat megzavarhatja a rendszer biztonságos működését. E probléma megoldását szolgálja az, hogy a központi egységben különböző állapotokat lehet kialakitani, amelyek közül nem mindegyikben hajtható végre az összes utasitás. Egyes utasitások privilégizáltak bizonyos állapotok számára, amennyiben másik állapotban kerülne rájuk sor, akkor speciális megszakitást okozva rendszerprogramoknak adják át a vezérlést. Ezen utasitások körébe tartoznak elsősorban az I/O utasitások

(START I/O: inditsd el az I/O tevékenységet; HALT I/O: fejezd be az I/O tevékenységet; TEST I/O: kérdezd meg a legutoljára végrehajtott I/O tevékenységre vonatkozó perifériális eszköz és eszközvezérlő egység állapotjellemzőit; TEST CH: kérdezd meg a csatorna és a csatornaprogram állapotjellemzőit), továbbá egyes kifejezetten rendszerfeladatok végzésére szolgáló utasitások (pl. a program állapotszó megváltoztatására, a memóriavédelem megszervezésére, az óraregiszter tartalmának megváltoztatására szolgáló utasitások).

A különleges processzor-állapot bevezetésével ugynevezett csapdázott utasitásokká váltak a korábban megállást okozó, valamint a nem definiált kódu utasitások. Ezek, ha a processzor normál üzemmódjában alkalmazzák őket, szintén megszakitásokat okoznak. Egyes gépeknél a program adott szakaszára csapdázottnak nyilvánithatunk normál utasitáskódokat is. Ilyenkor használva őket, normál végrehajtásuk helyett szintén megszakitás jön létre, és az ennek nyomán beinduló rutin helyettesiti a normál müködést. Ugyancsak csapdázott utasitásként viselkednek -- de most már programtól és processzor-állapottól függetlenül -- a szorzás, osztás vagy pl. a lebegőpontos utasitások, ha a hozzájuk tartozó opcionális hardware-t a processzor nem tartalmazza. Megemlitjük, hogy csapdázott utasitásokat először, még a hagyományos szó-orientált gépekre irt értelmező (interpretive) szubrutinok használtak, amennyiben az értelmező szubrutin hatása alatt végzett feldolgozás során csapdázásra került minden olyan müveleti kóddal rendelkező utasitás, amelyet a normál feldolgozásban használt módtól eltérően kellett értelmezni.(Akár értelmezve volt a normál feldolgozásra, akár nem). Ezek természetesen megszakitás helyett az (uj) értelmezésüket megszabó rész —szubrutinnak való vezérlésátadást okoztak.

Meg kell emlitenünk a normál üzemmódban is végrehajtható, szubrutin-hivás szerüen müködő un. definiálható utasitásokat, amelyeket pl. az ICL-1904-es vagy az ODRA 1204-es tipus használ. Ezek tulajdonképpen hardware segitségével kialakitott értelmező szubrutinokként müködnek. A normál szubrutin-hivó utasitásokhoz képest a különbség itt annyi, hogy a cimmezőben nem a szubrutin kezdőcimére hivatkozunk, hanem egy operandusz (vagy paraméter) cimére. Az utasitás hatására vezérlésátadás történik a müveleti kód által meghatározott fix cimre, az utasitást definiáló rutinnak itt kell kezdődnie. Az operandusz cimének a megjegyzése és az operandusz előhivása hardware-rel történik. Ezért a definiálható utasitással megvalósuló szubrutinhivásnál a paraméter-átadás egyszerübb.

A csatornák és megszakitási rendszerek adtak lehetőséget arra, hogy a felhasználók logikai szinten kezelhessék a perifériákat, hiszen gépi szinten a rendszer jelöli ki az aktuális fizikai perifériát és egyben rendet is tart a felhasználók perifériaigényei között. A kezdeti

I/O rendszerekben csupán azonos tipusu perifériák helyettesithetik egymást, tehát a program irásakor rögziteni kellett az eszköztipust, a konkrét fizikai eszközt azonban már nem (pl. MINSZK-23). A teljesen eszköz-független I/O kezelésnél csupán az adatfileokra vonatkozó információkat kell tartalmaznia a forrásprogramnak, a file-t tartalmazó I/O eszköz logikai specifikálása csak a futás előtt történik meg (a felhasználó adja meg!), a fizikai hozzárendelésről azonban már maga a rendszer gondoskodik (pl. IBM OS/360). A teljesen eszköz-független kezelésre módot ad az is, ha a hozzáfordulás valamely programhoz és tárolóterülethez történik (ez képviseli a központi egységben a perifériát), amely a hivatkozott eszköznek megfelelő formában közli az adatokat az aktivizálandó programmal [36].

2.4.5. Az időosztásos rendszerek kialakulása, fejlődése

A terminológia, hasonlóan más területekhez, itt sem egységes; különböző szerzők nagyon is eltérő jelentésekkel használják ugyanazon elnevezéseket. Zavaró tényező az is, hogy több elnevezés szinonimája egymásnak. (Az alábbiakban elsősorban az IFIP terminológiájára támaszkodunk [32].)

Időosztásról akkor beszélünk, ha a gép valamely egységét (a központi egységet) több egymástól független program használja időben váltakozva.

Az autonóm csatornák és a megszakitási rendszerek képezték a hardware alapot (a cimzési rendszerek és perifériák fejlődése mellett) az időosztásos rendszerek kialakulásához. Megjegyezzük, hogy az időosztásos elsősorban software fogalom, mivel a rendszerprogramoktól függ, milyen módon használjuk ki a hardware által biztositott lehetőségeket.

Az időosztás történetileg első formája a multiprogramozás volt, amely azt a lehetőséget használta fel, hogy az egyes programokhoz tartozó I/O tevékenységek befejeződésére való várakozási idejében más programok futtathatók a központi egységben. Ezzel párhuzamositani lehet több program futását. Gondoskodni kell természetesen arról, hogy a programok futását megfelelő operációs rendszer koordinálja, és hogy ahol csak lehetséges, a tevékenységek átlapoltan menjenek végbe. Ezt az elgondolást először a Ferranti Orion gépen valósitották meg részlegesen, teljes egészében pedig a Ferranti Atlas-on [21],[33] , [34]. Az időosztásos rendszerekben követelmény, hogy amint a központi egység felszabadul, azonnal valamely egyértelmüen meghatározott program foglalja le. Multiprogramozás esetén ez a kiválasztás prioritási számok összehasonlitása alapján történik (lásd 2.5.1.)

A prioritási rendszeren alapuló feldolgozási módok számos előnyt biztositanak a gép kapacitásának jobb kihasználása szempontjából. Felhasználói oldalról azonban lényeges kifogásokhoz is vezettek, amelyek közül a legfontosabb az, hogy kedvezőtlen esetben hosszu ideig kell várakozni a program futására. Ezek a hiányosságok vetették fel azt a gondolatot, hogy magának az időnek a mulását is jelentős faktorként kellene figyelembe venni. Ez megoldható olymódon, hogy hardware-eszközzel (timer) gondoskodunk arról, hogy meghatározott időintervallum elteltével megszakitójel érkezzék. A programok legmegfelelőbb ütemezését lehet elérni, ha e megszakitójel felhasználásával rögzitett időszakonként más-más program kapja meg a vezérlést, miközben megőrizhetők az időosztás előző formájánál használatos megszakitási formák is. Az igy kapott módszert időszeletelésnek nevezzük, tekintettel arra, hogy a legkarakteriszti-kusabb megszakitójelet a timer adja. Az időszeletelés igen sokfajta feldolgozási módot tesz lehetővé, melyekkel 2.5.1-ben foglalkozunk.

Ujabb számitógépeken már nemcsak a perifériás eszközök szerepelnek többszörözötten, hanem a központi feldolgozó egységek is. Ezáltal a feldolgozási tevékenység ujabb részei válnak párhuzamositha-tókká. Azokban az esetekben, mikor csak az aritmetikai-logikai egység fordul elő többszörösen (párhuzamos processzorok), csupán a memória megfelelő előkészitését kell elvégezni, mivel mindegyik egységben ugyanazok a müveletek hajtódnak végre. Ha mindegyik processzor egyenértékü (szimmetrikus processzorok), a munkák szervezésétől függ a felhasználási mód. Megfelelő kezelőrendszert feltételezve a gép ugyanazon munka különböző részeit hajthatja végre egyidejüleg, meggyorsitva ezzel a munka átfutását. Egyes modern programozási nyelvekben lehetőség van a programok elágaztatására és egyesitésére, ami ugyanazon programon belüli párhuzamos feldolgozást is lehetővé tesz (lásd 2.6). Mindkét esetben egyforma probléma jelentkezik: nehéz biztositani a processzorok egyenletes leterhelését. Ennek megoldását szolgálják az olyan rendszerek, melyekben több kezelőrendszer dolgozik egy magasabbszintü kezelőrendszer felügyelete alatt.

2.5. A kezelő-rendszerek fejlődése és jellemzése

A számitógépek történetében jelentős fejlődésen ment keresztül kezelésük módja is. Kezdetben a feldolgozásokkal kapcsolatos kiegészitő tevékenységeket csak manuális módszerekkel és kizárólag hardware-eszközök felhasználásával lehetett elvégezni. Ez nemcsak speciális ismereteket igényelt, hanem lassuvá, nehézkessé is tette a gépek kiszolgálását, és jelentős akadálya volt hatékony alkalmazásuknak. E mellett természetesen voltak előnyei is ti. közvetlen kapcsolatot tett lehetővé a (speciális számitógépes ismeretekkel rendelkező) felhasználó és programja között, ami jelentősen megkönnyiti a feladatok géprevitelét. A menet közben kiderült hibákra gyorsan lehetett reagálni, és a szükséges módositásokat azonnal keresztül lehetett vinni. Ezzel szemben viszont a munkaidő nagy részében álltak a gép egyes egységei, egy-egy feladat megoldására hosszu időt kellett forditani, tehát a gépek gazdasági, társadalmi hatékonysága igen korlátozott volt. Ezeknek a nehézségeknek a megoldását, a hardware megfelelő fejlettségi szintjén, software-eszközök segitségével kifejlesztett kezelő rendszerek (operációs rendszerek) jelentik.

2.5.1. Munkák (jobok) feldolgozásának módjai

2.5.1.1. Egyedi programkezelés (basic programming)

A korai számitógépeknél az egyedi programkezelés volt az általános módszer, amit egyébként még jelenleg is alkalmaznak a fejletlenebb rendszerprogramokkal rendelkező korszerü gépeken. Az egyedi programkezelés legfőbb jellemzője, hogy az egyes feldolgozási lépések között (fordító betöltése, forrásprogram bevitele, forditás, célprogram kiirása, célprogram bevitele, futtatása stb.) jelentős számu kézi beavatkozás szükséges; az egyes lépések -- még egy munkán belül is -- különálló egységet képviselnek. Bizonyos tevékenységeket (pl. a munkaprogram bevitelét a külső adathordozóról) még a változtatás nélkül ismételt feldolgozások esetén is ujra meg ujra végre kell hajtani. Viszont a folyó munkába a programozó szabadon beavatkozhat, megállithatja a futást, hibakeresést végezhet a gépen, stb. A gép és a felhasználó közötti kontaktus közvetlen. A gépek kihasználtsági foka azonban alacsony, egy-egy felhasználó ugyan (saját munkaidejét tekintve) jó hatásfokkal dolgozhat a gépen, több felhasználó azonban már nem.

2.5.1.2. Kötegelt feldolgozás

Amint a számitógépek sebessége annyira növekedett, hogy az egyes feldolgozási lépésekhez szükséges idő elhanyagolhatóvá vált a két ilyen lépés között vágzendő, kézi beavatkozást igénylő teen-

dőkhöz szükséges időhöz képest, éppugy szükségszerüvé vált az ope-

rációs-rendszerek megjelenése, mint ahogy a részeredmények tárolására alkalmas memóriával rendelkező, ill. a tárolt programu gépek megjelenése elkerülhetetlen volt, mihelyt az egyes (pl. aritmetikai) müveletek elvégzéséhez szükséges idő elhanyagolhatóvá vált a részeredmények kézi uton történő feljegyzéséhez, ill. a következő müvelet operanduszainak kézi uton való beállitásához szükséges időhöz képest.

Az operációs rendszerek megjelenése lehetővé tette a kézi beavatkozások számának lényeges csökkentését és -- adott munkákon belül -- a feldolgozási lépések automatikus összekapcsolását, a rendszert vezérlő utasitások segitségével. Az egyedi programkezelés lépései természetesen itt is adottak, de a megfelelő kezelőtevékenységeket már maga a rendszer végzi. Követelmény, hogy az egyes munkák ne különállóan kerüljenek a gépre. A munkáknak un. kötegeit (batch) kell az input eszközre, majd a külső tárolókba (pl. mágnesszalagra) vinni és ezek onnan kerülnek feldolgozásra.

Az operációs rendszerek kialakulásának természetes hardware feltételei voltak. Nevezetesen, szükség volt:

- elegendően nagy operativ memóriára a munka- és a rendszerprogramok egy (rezidens) részének egyidejü tárolása céljából;
- megfelelő nagy kapacitásu (pl. mágnesszalag) háttértárolóra a munkák kötegének tárolása céljából a feldolgozásra váró sorban (input queue); és
- (esetleg) megszakitási rendszerre, az egyes feldolgozási lépések közötti automatikus átmenet és egyéb események kezelése céljából.

A kezdeti kötegelt feldolgozást a következőkkel, lehet jellemezni [37]:

- ha egy munka megkapta a vezérlést, a teljes rendszer e munka számára dolgozott, az összes tevékenység befejezéséig;
- az I/O egységeket fizikailag cimezték, ami azzal járt, hogy ha a felhasználó meg akarta változtatni az I/O eszközt (akár azonos tipusura, akár másmilyenre), módositania kellett a programot, majd ujra forditani stb.;
- az I/O tevékenységeket teljes egészükben programozni kellett vagy assembly, vagy valami más szintü nyelven.

Az operációs rendszerek megjelenésekor a kötegelt feldolgozási forma a gépnek, mint egésznek állandó leterheltségét biztositotta ugyan, az egyes funkcionális egységeként azonban nem. Sőt ezzel egyidejüleg gyakorlatilag elzárta a felhasználót a géptől. A kötegelt feldolgozás tehát nemcsak, hogy nem oldotta meg az összes felmerült problémát, hanem ujakat is támasztott.

2.5.1.3. Kötegelt feldolgozás multiprogramozásos rendszerben

Az autonóm adatáviteli csatornák megjelenése adta az első döntő lökést az operációs rendszerek fejlődésének. Felébredt az érdeklődés az olyan módszerek és rendszerek kutatása iránt, amelyek képesek kezelni a működésükben egymás tevékenységeit átfedő I/O eszközöket. E kutatások eredményeként születtek meg a modern operációs rendszerek, amelyekben három fő részt különböztetünk meg:

- az input-output rendszert, (mely interface-ként szerepel a programok és az I/O eszközök között);
- a feldolgozó és kiszolgáló programokat (fordítók, assemblerek, rendszerkönyvtárba kapcsolt programok, stb.); és
- a supervisor rendszert (amely interface-ként szerepel a hardvare és a software között).

Az operációs rendszerekkel kapcsolatos kutatások eredményeként:

- általános I/O eljárásokat dolgoztak ki;
- lehetővé vált a programozó számára az I/O eszközök szimbolikus logikai kezelése (a fizikai eszközöket csak közvetlenül futás előtt kell rögziteni a rendszer számára);
- felismerték, hogy (az I/O tevékenységek miatti) természetes várakozások ideje felhasználhatóvá válik más munkák futtatására.

Az általános I/O eljárások és az I/O eszközök logikai kezelése rakta le az általános adatkezelő rendszerek kidolgozásának alapjait, a párhuzamosithatóság felismerése pedig elvezetett a multiprogramozásos rendszerekhez. Ezek alkalmazása a kötegelt feldolgozás folyamatában ujabb előrelépés volt. Szemben a korábbiakkal, a multiprogramozásos kötegelt feldolgozás esetén több munka áll egyidejüleg végrehajtás alatt ugy, hogy közöttük megosztódik a központi egység ideje (meghatározott prioritási rendszer alapján). Egyes feldolgozási részfolyamatok párhuzamosan futnak, egyidejüleg véve igénybe a számitógép különböző (I/O eszközök esetében esetleg több azonos tipusu) egységét. (Amennyiben a gép több processzorral rendelkezik és ezek működése is párhuzamositható, multiprocessingről beszélünk).

Az autonóm csatornák alkalmazásán tulmenően még a következő kiegészitő hardware-feltételek is szükségesek a multiprogramozásos kötegelt feldolgozáshoz:

- a munkák befejezése, vagy felfüggesztése (I/O tevékenységre, a gép által készitett hibalista alapján programjavitásra való várakozás, stb.) esetén a következő prioritásu számu program-

nak automatikusan át kell tudnia venni a vezérlést, tehát kiterjedt megszakitási rendszerre van szükség.

- a párhuzamos kezelhetőség céljából a konkurráló munkákat a rendszerbe való belépés után direkt elérésü tárolóban kell elhelyezni, s ugyanigy a rendszerből kilépő outputokat. Megjegyezzük, hogy ez elvileg elvégezhető mágnesszalag alkalmazásával is, de ebben az esetben a rendszer hatékonysága csekély lesz.

A multiprogramozásos rendszerek a munkák összességének gyors átfutását biztositják a számitógépen, miközben (a feldolgozások részleges párhuzamositásával) többé-kevésbé optimatizálják is a gép funkcionális egységeinek kihasználtságát.

A munkák végrehajtási sorrendjének meghatározásában a következő tényezők játszanak szerepet:

- a belépő munka a kivülről adott prioritási szám, az érkezési sorrend, a jelzett perifériahasználat, a becslés szerint szükséges gépi idő, stb. alapján belső prioritási számot kap (amely a hagyományos rendszerekben a feldolgozás végéig állandó marad, az ujabb rendszerekben azonban változhat attól függően, hogy a munka ténylegesen hogyan terheli a gép különböző egységeit);

- megtörténik a központi egység (időbeli) felosztása a programok között a következőképpen: egy adott programhoz van hozzárendelve ,ameddig /1/ a végrehajtás során olyan helyzet nem áll elő, hogy várakoznia kell valamely, a központi egységtől független tevékenység befejeződésére, vagy /2/ az operációs rendszer fel nem függeszti a futást abból a célból, hogy átadja a vezérlést egy futásra kész magasabb prioritásu programnak. /1/ esetén alacsonyabb prioritásu program kapja a vezérlést a feltartó tevékenység befejezéséig, /2/ esetén viszont magasabb prioritásu program egy ilyen tevékenység befejeztével.

Tekintettel arra, hogy gyors időközökben egymás után több program használja a központi egységet, igen fontos a memóriakezelés pontos megoldása. Általában a következő módszerek lehetségesek ennél a feldolgozási formánál [37]:

- fix felosztásos módszer (fixed-partition system): a dinamikus memória (a rendszer által le nem foglalt memória) előre meghatározott számu részre (particióra) oszlik, az inicializált munka a megadott helyigény szerint kerül valamely particióba, és a végrehajtás során ezen belül kell maradnia. Jellemző rá a báziscimzéses rendszer.

- változó felosztásos módszer (region allocation): változó számu

munka lépjen be egyidejűleg a rendszerbe, illetőleg ezek ... meghatározott memóriaigénnyel,

- roll in/roll out ("hol be, hol ki") módszer: abban tér el a változó felosztásos módszertől, hogy futás közben az egyes munkák ideiglenesen egymás memóriaterületeit is felhasználhatják. (Az utóbbi két változat realizálását nagyon megkönnyiti a lapszervezésü memóriák alkalmazása.)

A multiprogramozásos rendszerek bevezetésével megvalósult a gépen belüli tevékenységek párhuzamositása, ami a gép jobb kihasználásához vezetett. Ezzel párhuzamosan azonban megszünt a felhasználó és a gép közvetlen kapcsolatának a lehetősége, mivel a rendszerbe való belépés után már csak azon keresztül lehet kommunikálni a munkákkal. E mellett a prioritásos kezelésmód ujabb nehézségeket okozott: bizonyos körülmények szerencsétlen egybeesése esetén egy-egy munka napokig sem került végrehajtásra, annak ellenére, hogy az ujabb rendszerekben a prioritási számok kialakitásának összetettebb módjaival igyekeztek javitani a helyzeten.

2.5.1.4. Többszörös hozzáférésü rendszerek

Mint már emlitettük, kialakult az időosztásos rendszerek egy másik tipusa, ahol a munkák közötti vezérlésátadást, adott időintervallum leteltével megjelenő órajel (vagy egyéb megszakitási jelek mellett az is) vezérli (időszeletelés). Az igy kialakitott megszakitások segitségével a kezelő-rendszer egymást követően különböző munkákhoz (felhasználókhoz) rendeli hozzá az eszközöket, hardware-t és software-t egyaránt. Az időszeleteléses rendszerekben mind a memória egyes részeit, mind az egységek idejét ideiglenesen, dinamikusan rendelik hozzá az egyes munkákhoz. A programok egy része (a futó program mellett) rezidensként tartózkodhat az operativ tárban, más része közvetlen hozzáférésü külső tárolón helyezkedik el. A futó programtól rezidens program kapja meg a vezérlést, ha /1/ az időintervallum letelt, vagy ha /2/ a futó program valamilyen (pl. I/O) tevékenység miatt várakozik.

Ezekkel a rendszerekkel kapcsolatban vezették be a virtuális memória (majd később a virtuális processzor és a virtuális rendszer) fogalmát. Ennek az a lényege, hogy a programok valóságos hardware eszközök (pl. memória) helyett csak azok logikai képével dolgoznak, a fizikai hozzárendelés csak akkor történik meg, ha futáskor szükség van rá.

A memóriakezelés tipikus módjai a következők:

- valamennyi program az operativ tárban helyezkedik el (core resident); ennek az az előnye, hogy a munkák átkapcsolása gyorsan, a rendszer I/O tevékenységekre való várakozása nélkül valósul meg, hátránya viszont az, hogy a memóriaigény rendkivül nagy, tehát csak kevés munka egyidejü kezelésére van lehetőség;

- az időintervallum végén a felfüggesztett program külső tárolóba kerül, helyét pedig ujabb program foglalja el (swapping). A főtárolóba egyidejüleg tartozkodó programok számának nö-

velésével a rendszer I/O tevékenységgel lekötött ideje jelentősen csökkenthető, de ugyanakkor a tárolóigény megnövekszik;

- lapszervezésű rendszerekben (paging) a programoknak mindig csak a kezelés alatt lévő lapjai vannak az operativ memóriában. A logikai cimek alapján, kombinált hardware--software eszközök gondoskodnak az igényelt lap fizikai feltöltéséről (a rendszer által kijelölt helyen).

A fenti eszközök egyszerre két problémát is megoldanak. Kötegelt feldolgozás esetén a rendszerbe beléptetett programok belátható időn belül végrehajtásra kerülnek; ezzel tehát megszünik a kötegelt feldolgozás egyik komoly hiányossága. Másrészt -- gyors átkapcsolások segitségével -- lehetőség nyilik arra, hogy egyidejüleg több felhasználó is bekapcsolódjék a rendszerbe olymódon, hogy mindegyikben az az illuzió keletkezzék, mintha kizárólagosan ő használná a gépet. Ezzel a rendszer többszörös hozzáférésüvé válik, azaz több felhasználó rendelkezik egyidejüleg ugyanazokkal az előnyökkel, nevezetesen:

- on-line kapcsolódnak a géphez, közvetlenül vihetik feladataikat rá központi diszpécser és operátor közremüködése nélkül;
- gépközelben, a géppel való szoros kontaktusban végezhetik a program kezelését (összeállitás, belövés, módositás, futtatás).

(Megjegyezzük, hogy elvileg a prioritásos rendszerrel is megoldható több felhasználó bekapcsolása, de ugyanaz a nehézség lép fel, mint a kötegelt feldolgozásnál: nem garantálható, hogy a felhasználók belátható időn belül "szóhoz jutnak".)

A felsorolt nyereségek mellett nem kell lemondani arról az előnyről sem, hogy a központi rendszert optimálisan használjuk ki.

Érdemes összehasonlitani a multiprogramozásos kötegelt feldolgozást és a többszörös hozzáférésü feldolgozást a feladatmegoldás folyamatában. [38] szerint a többszörös hozzáférés esetén 16, mig multiprogramozásos kötegelt feldolgozás esetén 19,3 emberóra a felhasznált idő. A felhasznált központi egységidő-percben 5,75 illetve 1,25. Jól látható, hogy ez önmagában nem szól egyértelmüen egyik változat mellett sem. Átlagos (nyugati, 1970 évi) órákon számolva mégis a többszörös hozzáférésü rendszerek mutatkoznak gazdaságosabbaknak.

2.5.1.5. Kombinált feldolgozási módok

A gyakorlatban az időosztás két fentebb tárgyalt módja nem különült el mereven egymástól. Hatékony rendszereket, amelyek a hardware konfigurációjának és a konkrét felhasználói igényeknek egyaránt megfelelnek, gyakran a két módszer különböző kombinációival hoztak létre.

× × ×

A felsoroltak mellett még egy lényeges hardware feltételt emlitünk meg, amely elengedhetetlen a felhasználó kényelmes és hatékony munkájához. Tekintettel arra, hogy célszerü, ha egy nagy gépen egyidejüleg többen dolgoznak, a kezelő eszközöknek (a terminálok-

nak) különböző, gyakran egymástól távoleső helyeken kell lenniök, állandó, vagy ideiglenesen kapcsolatban a központi géppel. Ehhez viszont egyrészt több megfelelő végberendezés (adatvégállomás, terminál, programozható terminál, szatellit gép, stb.), másrészt nagy megbizhatóságu átviteli hálózat szükséges. A kihelyezett végberendezések kezdetlegesebb tipusai csupán az adatfelvétel és leadás céljait szolgálták. A fejlettebb tipusok viszont alkalmasak munkák irányitására is a központi gépen, mig a legfejlettebb változatot a perifériavezérlő és előfeldolgozó egységként használatos kisgépek jelentik.

A terminálok és a központi gép (rendszer) lehetőségeinek kihasználását a következő módokon szokták biztositani:

- adatvégállomás alkalmazása: a központban inicializált programok segitségével adatfile-ok kialakitását teszik lehetővé;
- Remote Job Entry (RJE): a terminál ugyanolyan kötegelt feldolgozást tesz lehetővé mint a központi gép, olymódon, hogy a terminálról bevitt kötegelt munkák beállnak a rendszer egyesitett várakozó sorába és prioritásuknak megfelelően kerülnek végrehajtásra (ez csupán a központi kezelőket kapcsolja ki a feldolgozásból, a feldolgozás folyamatában azonban nem jelent közvetlen kapcsolatot);
- konverzációs terminálhasználat: a feldolgozás a felhasználó számára (látszólag) real-time jellegü, de viszonylag szük lehetőségeket biztosit a felhasználható programozási nyelvek (pl. APL, BASIC) és általában a központi rendszerprogramok vonatkozásában;
- konverzációs Remote Job Entry (CRJE): egyesiti az előző feldolgozási módok előnyeit. Konverzációs üzemmódban lehet információkat bevinni, programot összeállitani, belőni stb., de emellett komplett munkát is elő lehet készitení és feldolgozásra átadni a központi munkafeldolgozó rendszer input sorának.

A fenti rendszerek vetették fel az adatvédelem (protection) és titkositás (privacy) kérdését. Nyilvánvaló, hogy nagy zavarokhoz vezetne, ha a felhasználók tetszés szerint hozzányulhatnának egymás, vagy a rendszerprogram- és adat-file-jaihoz. Ez vonatkozik mind a ráirásra, mind a kiolvasásra. Ezért a terminálról bejelentkező felhasználót azonositóval látják el, amelyhez egy listán lévő jelző szó tartozik, ami megmutatja, hogy az illető mire jogosult. Az adatvédelem és titkositás feladatait részben hardware-, részben pedig software-eszközökkel oldják meg.

A hardware-eszközök időosztása csak az egyik jellemzője a modern kezelőeszközöknek. A tárolás gazdaságossági szempontjai miatt vetődött fel az adatok, illetve a programok megosztásának kérdése. Az előbbin azt értjük, hogy a központilag tárolt adatokhoz

több felhasználó is hozzáférhet, tehát ezeket nem kell az egyes felhasználásokhoz többszörözve tárolni. Az utóbbinál különösen a rezidens programokra vonatkozóan fontos, hogy kevés helyet foglaljanak el, és mégis ki tudják szolgálni a különböző munkákat. Ennek még akkor is teljesülnie kell, ha a megszakítás ugy következik be, hogy a megszakitott programrészt a megszakitó munka is használni kívánja. Ezek azzal a következménnyel járnak, hogy a program:

- nem módosithatja önmagát (invariant, reentrant program); és
- nem kezelhet lokális adatokat.

Az emlitett feltételek igen erős megszoritásokat jelentenek, melyek feloldásához a lapszervezés nyujt nagy segitséget, ami lehetővé teszi, hogy számos logikai cimnek átkapcsolásakor más és más fizikai cimek feleljenek meg.

2.5.1.6. Multiprocesszoros feldolgozás

Az eddigiekben olyan rendszerekkel foglalkoztunk, melyek egy központi egységet tartalmaznak és csak az I/O egységekből (beleértve a háttértárolókat is) áll több rendelkezésre. Hatékonyabb feldolgozás várható az olyan rendszerektől, amelyekben vagy több, önállóan is működőképes gép dolgozik összehangoltan, vagy több ilyen gép kijelölhető, bár fizikailag ezek határai nem rögzitettek.

A legkezdetlegesebb ilyen rendszerekben egyetlen nagy gépet több kisebb gép szolgált ki oly módon, hogy I/O tevékenységeket végeztek (szerkesztéssel, ellenőrzéssel együtt), és a feldolgozott anyagot gyors periféria adathordozón továbbitották a központi géphez (géptől), emberi közreműködéssel. A következő fokozatot a szatellit gépek jelentik, amelyeknél az előzőkben leirt funkciók is teljesülnek, de az adatátvitelt már maga a rendszer szervezi elektronikus sebességgel. A legtöbb esetben a szatellit gépek a megszakitórendszer segitségével kezdeményezhetik a kapcsolatot a nagy géppel és hozzáférhetnek a központi memóriához is. Amennyiben nagyobb teljesitőképességü gépet (attached support processor) kapcsolnak a központihoz (közös hozzáféréssel a memóriához, vagy csatorna-csatorna adapterrel), az önálló gépként is működhet, "szabad" idejében (kisebb) jobokat kezelve (pl. B6500 - ILLIAC IV).

Multiprocesszoros feldolgozásról akkor beszélünk, ha a feldolgozás olyan rendszerrel történik, mely egynél több központi egységet (central processzort) tartalmaz. Ezek elemei a vezérlő egység (control unit) és a müveletvégző egység (arithmetic-logical unit).

Több processzor alkalmazásának céljai általában a következők [37]:

- a megbizhatóság fokozása, vagy

- az átbocsátóképesség növelése.

Az első esetben a többszörözés csupán két célt szolgál:

- meghibásodáskor a központi egységek átvehetik egymás szerepét, vagy
- mindegyik ugyanazt a tevékenységet végzi, amikor is (az eredmények összehasonlításával és az egyezőknek helyesként való elfogadásával) rendkivül nagy megbizhatóság érhető el. (Az ILLIAC IV rendszernél 2-4 óránként számitanak egy meghibásodásra! [26].)

E rendszerek azonban a munka-kezelés szempontjából nem vetnek fel lényeges uj kérdéseket.

A feldolgozó egységek müveletvégző sebességének lényeges növelése érhető el, ha az ALU-k számát növelik és müködésüket párhuzamositják. Az ilyen rendszerekben egyetlen vezérlő egységnek alárendelve (memóriával kiegészitett) ALU-k mátrixa dolgozik, párhuzamosan végezve ugyanazt a müveletet. Megkülönböztetésül az olyan rendszerektől, melyekben több CU is müködik (szimmetrikus processzoros gépek), az előbbieket párhuzamos processzoros gépeknek nevezzük. A párhuzamos processzorok igen hatékonyan alkalmazhatók mátrixmüveleteknél, közönséges és különösen perciális differenciálegyenletek megoldásánál stb. Ezek a rendszerek a korábbi gépektől eltérő algoritmusokat, a programozási nyelvekben uj elemeket és a gép uj struktúrájának megfelelő kezelésmódot igényelnek, de itt sincs lényeges változás a munkák folyamatának (job stream) kezelésében.

A szimmetrikus processzoros kezelőrendszereknél az alábbi két tipus legelterjedtebb [37]:

- ha a processzorok megosztva használják a központi tárolót és a rendszer vezérlőprogramját, akkor a munkát kereső processzor kiválasztja a következő végrehajtásra kész feladatot (taskot) és azzal kezd foglalkozni;
- ha a processzorok a direkt elérésü tárolókat használják megosztva, továbbá ha mindegyik processzor külön operativ tárolóval és kezelőrendszerrel rendelkezik, akkor a megosztott direkt elérésü tárban elhelyezett input sorból kerülnek elosztásra a munkák a processzorok között.

Természetesen mindkét esetben több szervezési lehetőség áll rendelkezésre mind a munkák ütemezése és a memóriaszervezés, mind az egyszeres és a többszörös hozzáférés szempontjából. Ezeknek a bonyolult rend-

szereknek a használatánál egyszerü prioritási sémákkal nem lehet megoldani még a hozzávetőlegesen optimális kihasználást sem, ehhez egyes esetekben matematikai programozási (optimalizálási) feladatok megoldására van szükség [20].

Szimmetrikus processzorok esetén több kezelőrendszerre van

szükség. Ezek számos virtuális gépet kezelnek egyidejüleg olymódon, hogy az egész géprendszer kihasználtsága közel optimális legyen. Ebben az esetben a munkák átfutása lényegesen gyorsabb, mint az egyprocesszoros gépeknél.

2.5.2. Az operációs rendszer részei és feladataik

Az operációs rendszereknek három fő részét jelöltük meg az előző részben. Az alábbiakban a felügyelő rendszerrel, az I/O rendszerrel, valamint a kiszolgáló programokkal foglalkozunk. A fordítóprogramokat a következő részben tárgyaljuk.

2.5.2.1. A felügyelő rendszer

A felügyelő rendszerben két fő részt különböztetünk meg: a rendszer-, valamint a munka-kezelő részt. A rendszer-kezelő rész az un. hardware-megszakításokat kezeli, amelyek felfüggesztik a programok futását. Ezek vagy valamely I/O eszközről, illetve az időszeletelő óráról, esetleg a konzolról érkeznek, vagy pedig csapdázott utasitások esetén fordulnak elő. A rendszerkezelő programok feladata éppen a megszakitások feldolgozása. Ennek során:

- első lépésben azonositódik a megszakitás, majd a
- második lépésben a megfelelő rutin aktivizálásával és végrehajtásával megtörténik a tényleges feldolgozás.

Az emlitett tevékenység során a rendszerkezelő program felügyeletet gyakorol a rendszer elemei (processzorok, központi tároló, I/O eszközök) felett, és kezeli azokat. Tekintettel arra, hogy a fent jelzett funkciókat hatékonyan kell teljesiteni (gyakran kerül rájuk sor), a rendszerkezelő rutinoknak állandóan a központi tárolóban kell lenniök, védelmük érdekében pedig nem fordulhat hozzájuk feldolgozó program, és csak supervisor-üzemmódban hajthatók végre.

A munka-kezelő rész tevékenysége két nagy csoportra osztható:

- az egyik a software-megszakitások kezelése (monitoring),
- a másik pedig a munka-vezérlő információk feldolgozása és kezelése.

A software-megszakitások kezelése során mindig valamilyen diagnosztikai rutin, illetve a felhasználó által megadtt rutin kapja meg

a vezérlést és ez kezeli a program, valamint az operátor közötti üzeneteket. A munka-vezérlő információk feldolgozásánál, miután megtörténik az input sorból való beolvasás és értelmezés, a vezérlés a job control utasitásban megjelölt tevékenységnek megfelelő rutinnak (egyes esetekben az I/O rendszernek) adódik át. A felügyelő rendszer emellett gondoskodik az input sor összeálli-

tásáról, a munkák taoben utemezeséről, a memória-hozzárendelésről és az output sor kiviteléről.

2.5.2.2. I/O rendszer

Az I/O rendszer (ujabb terminológia szerint: adatkezelő rendszer) az I/O tevékenységeket szervezi a feldolgozó programok számára, felügyel a különböző katalógusokra és kezeli a pufferterületeket. Rutinjait az alábbi csoportokra oszthatjuk:

- az I/O egységek és a központi tároló közötti átvitelt szervező rutinok (access routines);
- katalógus-kezelő rutinok (ezek segitségével valósitható meg a file-ok név szerinti hivatkozása);
- eszközkezelő rutinok (ezek végzik a fizikai I/O hozzárendelést);
- külső tároló kezelő rutinok (ezek határozzák meg a direkt hozzáférésü tároló felosztását a felhasználók között).

Annak ellenére, hogy az I/O tevékenység szervezését az I/O rendszer végzi, a fizikai I/O utasitások végrehajtása a felügyelő rendszer rutinjainak a feladata (I/O executive), minthogy ez a tevékenység a hardware rendszer bizonyos elemeinek kezelését jelenti, ami a felügyelőre (supervisor) tartozik.

2.5.2.3. Rendszerkönyvtár, kiszolgáló (utility) programok, programkönyvtár

Az eddig tárgyalt részek -- viszonylag egységes rendszerbe foglalhatóan -- a rendszerkezelés legfontosabb feladait látják el. Ezek mellett azonban számos olyan tevékenység is van, amely nagymértékben megkönnyiti a rendszer karbantartását és a felhasználók rendszeresen visszatérő igényeinek kielégitését. További követelmény, hogy a rendszernek bővithetőnek kell lennie, mégpedig oly módon, hogy az uj igények kielégitése és az ujabb funkciók beépitése ne vonja maga után az egész rendszer módositását. Célszerü ezért az egyes funkciónak megfelelő rutinokat változtatható könyvtárakban elhelyezni, ami

lehetővé teszi, hogy csak felhasználás előtt aktivizáljak őket. Ez az eddig előfordult rutinokra is vonatkozik, egy megjegyzéssel. Nem célszerü ugy szervezni a könyvtárakat, hogy bármely programból tetszés szerint hozzá lehessen férni bármelyikhez. Ezért meg szoktak különböztetni rendszerkönyvtárat, amelyhez a hozzáfé-

rés a kezelő rendszer joga, bár egyes esetekben ettől el lehet térni, továbbá felhasználói könyvtárakat, amelyekben a közhasznu kiszolgáló (utility) programok és az egyes felhasználók (esetleg védett) feldolgozó programjai kapnak helyet.

2.5.3. A feldolgozó rendszer megbizhatóságának kérdése

A számitógépek fejlődésével és tömeges elterjedésével szoros kapcsolatban, javitani kellett a gépek megbizhatóságát. Ma az elvégzett müveletszámra vetitve az előforduló hibák száma nagyságrendekkel kisebb, mint a kezdeti időszakban, bár még a legkorszerübb gépeknél is számitani kell időnként előforduló hibákra. A széleskörü felhasználás és a megadott feladatok fontossága egyaránt megköveteli az előforduló hibák gyors felderitését, hogy felhalmozódó következmények felszámolhatók legyenek.

A hardware megbizhatóságát különböző szinteken különböző eszközökkel (pl. párhuzamositással) oldják meg. A várható egyéb hibák ellen azonban software védelmet kell biztositani. Ez azt jelenti, hogy egyrészt időben kell felderiteni az előforduló hibákat, másrészt biztositani kell, hogy a felderitési állapothoz lehetőleg közeli állapotból legyen ujraindithatő a rendszer, visszavezetve a feldolgozást a helyes kerékvágásba.

Az ujraindithatóság érdekében bizonyos információkat (pl. a rendszer állapotát jelző táblázatokat, várakozó sorokat stb.) különleges védelemmel kell ellátni. Célszerü ha ezeket az információkat egyetlen rutin kezeli. Ismeretes, hogy a hibadetektálás és javitás alapja az információk redundanciája. Ez lehet implicit (pl. a listák szerkezetéből adódó) vagy olyan explicit formáju (pl. kontrollösszeg), amely kifejezetten a hibafelderités céljait szolgálja. Ujrainditáskor az a cél, hogy ne kelljen "hidegstart"-ot csinálni, hanem minél kevesebb felhasználóra terjedjen ki az ujrafuttatás, az egyes feladatok ismétlése. Ennek érdekében egy speciális program vizsgálja végig a rendszer állapotát jelző részeket és az ujratöltés során átmenti a sértetlen vagy javitható információkat, hogy ezek alapján folytatódjék a munka. Célszerü, ha a hiba időpontjában az érintett felhasználók is jelzést kapnak a meghibásodásról.

A hardware mellett a software-ben is előfordulhatnak hibák, amik nem véletlenszerüen jelentkeznek, hanem adott szituációkban rendszeresen. Ezen hibák felderitésében nagy segitséget nyujtanak a rendszer eseményeit nyomon követő és feljegyző programok. Ugyanez az elv alkalmazható a hardware "kórtörténetének" felderitésére is.

Amennyiben valamilyen hibát észlelnek, gondoskodni kell lehetőleg menet közbeni pontos lokalizálásáról és elháritásáról. Korszerü rendszereknél e célból on-line tesztek állnak rendelkezésre, melyek konverzációs formában teszik lehetővé a vizsgálatokat. A software-rendszer hibáinak kijavitására olyan programokat szokás

alkalmazni, amelyek a rendszer valamennyi eleméhez hozzátérhetnek. A hibás rutinokat időszakosan is ki lehet emelni, általában anélkül, hogy a kezelőrendszer üzemképtelenné válna, csupán néhány munka szenved késedelmet. Ilyen eljárás biztositása a hardware esetében még megoldásra vár.

2.6. A programozási nyelvek fejlődése és főbb jellemzői

A programozási nyelvek -- hasonlóan minden más nyelvhez -- az információátvitel célját szolgálják valamilyen adó és vevő között (természetesen kódolt formában). A programozási nyelvek esetében az adó a feladatot leiró ember, a vevő pedig a feldol-

gozást végző elektronikus számítógép. Ez a körülmény a programozási nyelvek számos sajátosságát meghatározza, és egyben korlátozza lehetőségeiket. Mivel adóként az ember szerepel, célszerű lenne az emberéhez minél közelebb álló nyelvet alkalmazni az információátvitelre. Ugyanakkor biztositani kell, hogy a vevő (a számitógép) képes legyen dekódolni és értelmezni minden számára szóló kódolt információt. Ez a két követelmény némiképpen ellentmond egymásnak, minthogy a gép számára közvetlenül érthető nyelv (saját belső kódrendszere) igen távol áll az emberi nyelvektől.

Az első számitógépek még csak a saját kódjukon irt információt "értették meg", ez pedig erősen akadályozta nemcsak hatékony alkalmazhatóságukat, hanem szélesebbkörű elterjedésüket is. Felhasználói oldalról felvetődött az igény olyan közvetitő nyelvek iránt, melyek kielégitik az alábbi követelményeket:

- közelebb állnak a természetes nyelvekhez (beleértve a matematikai formulanyelvet is);
- segitségükkel elég precizen fogalmazhatók meg a feladatok számitógép számára (a konkrét gép specifikációi nélkül is); és
- meggyorsitják a problémák előkészitését, programozását.

Két ut látszott járhatónak abban az irányban, hogy a számitógépet képessé tegyék az ilyen "magasabb szintű" nyelvek megértésére:

- vagy olyan gépet kell épiteni, melynek belső kódrendszere és utasitásrendszere megegyezik (izomorf) egy magasabb szintű bemenő nyelvvel (ezen irányzatot képviselik a formulavezérlésű számitógépek, melyekkel a tanulmány 3. fejezete foglalkozik);
- vagy olyan eszközökkel, programokkal kell ellátni a meglévő általánosan elterjedt gépeket, amelyek segitségével közvetlenül megérthetik a közölt információt, illetve amelyekkel az lefordítható, automatikusan átkódolható a gép belső nyelvére.
  (A paragrafus további részében ezzel az utóbbi lehetőséggel foglalkozunk.)

Történetileg a második elgondolás indult először fejlődésnek, a formulavezérlésű gépek csak később kerültek az érdeklődés közép-

pontjába. A programozási nyelveket feldolgozó rendszerek (Programming Language Processors-PLP) kialakitásához Turing munkássága szolgáltatott elvi alapot [39]. E szerint (kissé egyszerüsitve) minden számitógép, feltéve hogy rendelkezik bizonyos számu és megfelelő tulajdonságu utasitással, alkalmas arra, hogy bármilyen tág utasitás-rendszerű más gépet szimuláljon. (A magasabb szintű

programozási nyelvek is felfoghatók valamilyen (formulavezérlésü) gép belső utasitásrendszereként.)

Az első programozási nyelvek természetesen még igen közel álltak a gépi nyelvekhez. Ennek az a magyarázata, hogy ilyen jellegü nyelvek feldolgozása igényli a legegyszerübb PLP-t. A későbbiek folyamán erősödik meg az a tendencia, hogy a programozási nyelvek mind jobban közelednek a természetes nyelvekhez.

Azt, hogy milyen szintü és milyen elemekből álló nyelvet alkalmazhatunk, elsősorban a PLP-k fejlettségi szinvonala határozza meg. A PLP-k fejlődése egyre bonyolultabb strukturáju programozási nyelvek bevezetését teszi lehetővé. Eddigi ismereteink szerint azonban a természetes nyelvek szintjéhez való közlekedés korlátokba ütközik, ami indokolttá tette annak a következtetésnek a levonását, hogy a természetes nyelvek sohasem válhatnak programozási nyelvekké. (E nézet egyik legkiemelkedőbb képviselője Y.Bar-Hillel.)

A programozási nyelvek fejlődésének áttekintésénél mindenekelőtt az alábbi osztályozásra támaszkodunk (majd a későbbiekben más jellemzők szerinti felosztások is szóba kerülnek):

- gépi utasitásrendszer szintje,
- assembly szint,
- eljárás-orientált nyelvek,
- probléma-orientált nyelvek.

Az eljárás-orientált nyelvek között fogjuk tárgyalni a magasabb szintü (szükebb értelemben) gépre orientált nyelveket is, amelyeket nem tekintünk külön osztálynak, mivel az alapelvekben sok a közös vonás. (Tágabb értelemben véve ugyanis minden programozási nyelv gépre orientáltnak tekinthető, annak ellenére, hogy egy ideális programozási nyelv esetén elvárható lenne a dekódolótól, a számitástechnikai eszköztől való függetlenség. Az ilyen irányu igények kielégitése azonban nem valósult meg [40] és véleményünk szerint nem is valósulhat meg a közeli jövőben.) A természetes nyelvekkel szemben (melyeken problémáinkat különösebb nehézség nélkül meg tudjuk fogalmazni) minden programozási nyelv lényeges megszoritásokat tartalmaz. Ezeken igen jól nyomon követhető az a tervezőikkel szemben támasztott követelmény, hogy (adott) géppel feldolgozandó nyelvet kellett produkálniuk. Ha tágabb értelemben vett gépre orientáltságról van szó, természe-

tesen nem minden esetben kell valamilyen meghatározott géptipusra gondolunk (ekkor beszélnénk szükebb értelemben vett gépre orientáltságról), hanem csak arra, hogy a programozási nyelvek megalkotásánál mindig valamilyen gépkoncepciót tartanak szem előtt (memória-hozzárendelés, indexregiszter, szekvenciális utasitásszámláló, memóriától elkülönülő aritmetikai egység, szekvenciális müveletvégrehajtás, parallel feldolgozás stb.). Igen jól illusztrálja ezt a gondolatot, hogy a Neumann-i (rendkivül gyümölcsöző) szekven-

ciális müveletvégrehajtási koncepciót alapnak tekintő eljárás-orientált nyelvek alkotják a ma használatos programozási nyelvek zömét. A listafeldolgozó nyelvek, melyek a hagyományos memóriaszervezéstől eltérő szervezést igényelnek, kevésbé elterjedtek, a modulrendszerü "multicomputerek" lehetőségeit teljesen kihasználó nyelvek pedig még gyakorlatilag nem is léteznek. Ebben persze az is közrejátszhat, hogy a Neumann-i koncepciótól eltérő elveken alapuló rendszereken nagyon kevesen tudják áttekinteni [40].

Az eddigi elemzésből tehát arra a következtetésre juthatunk, hogy a programozási nyelvek alapelveit a mindenkori számitógépek felépitésében alkalmazott koncepciók határozzák meg, ezen belül pedig elsősorban a PLP-k fejlettségi szintje.

2.6.1. Gépi szintü programozás -- a szubrutinkönyvtárak kialakulása

A számitógépek programozása az első időszakban (megjelenésüktől kb. 1952-ig) közvetlenül belső kódrendszerükben, gépi kódjukban történt. Ebben a viszonylag hosszu "lappangási" időszakban a gépek felhasználói elsősorban a gépet épitő és tervező, valamint egy körülöttük kialakult szük gárdához tartozó emberek voltak, akik számára természetesnek tünt az általunk konstruált és igen jól ismert gépi kódrendszer használata.

1952-től kezdődően azonban lényeges változások következtek be; megindul a programozási nyelvek és PLP-k fejlődésének tulajdonképpeni története. A gépek elterjedésével a felhasználók köre jelentősen kibővült és számukra már egyáltalán nem volt kielégitő az egyedi programkészités, a kényelmetlen, elemi müveleteket tartalmazó kódrendszerek alapján.

Az első lépés a programozás megkönnyitése felé a szubrutinmódszer kidolgozása (amelyben a paraméter-átadás okozott eleinte problémát) és a szubrutin-könyvtárak kialakitása volt (1952-53: az EDSAC ; Wilkes, Wheeler, Booth és Gill munkássága nyomán, és az ILLIAC gépek szubrutin-könyvtárának megalkotása). Eleinte ezek a könyvtárak csak fix programokat, rögzitett utasitássorozatokat tartalmaztak, melyekre a felhasználók névvel hivatkozhattak, tehát ily módon kérhették végrehajtásukat. Ez a rendszer egyben a makroutasitások kezdetleges alkalmazási formája volt (igen szük makro-fogalommal). Azóta a makroutasitás fogalmának tartalma lényegesen kibővült.

2.6.2. Assembly szint -- makroutasitások

A könyvtárak használatának megkönnyitésére irányuló törekvések vezettek el -- többek között -- az assembly rendszerek kialakitásához. Az assembly rendszerek csirája abban található, hogy a programozók voltaképpen sohasem használtak szigoru értelemben vett (bináris) gépi kódot. A müveleti kód helyett elég korán a könnyebben megjegyezhető mnemonikus kódokat vezették be, a valódi cimek mellett pedig -- legalábbis a program egyes utasitásaihoz füzött ma-

gyarazatban -- szimbolikus jelöléseket is alkalmaztak a megfelelő fekeszek tartalmára abból a célból, hogy mások is megértsék a programot. Olyan utasitások megjelölésére pedig, amelyekre más (pl. vezérlésátadó, utasitásmódositó) utasitásokra hivatkoztak, lassanként megjelentek a cimkék, eleinte "nem hivatalosan", a program-ürlap bal szélén, majd az utasitás valódi cime helyett a rá hivatkozó utasitásban is. A relativ cimzés csirája pedig az volt, amikor a programozó előre megirta a programnak egy abba később majd beépitendő részét, az utasitások cimeit egyelőre (pl.) k, k+1, k+2,... alakban irva, azzal, hogy k értékét majd később adja meg.

Assembly rendszerről a szó valódi értelmében azonban csak azóta beszélhetünk, amióta megjelentek az assemblerek is -- amelyeket az első PLP-knek tekinthetünk -- az addig az adatelőkészitő által végzett átkódolási munka (mnemonikus kódok helyére bináris müveleti kódok, szimbolikus cimek és cimkék helyére valódi cimek beültetése stb.) automatizálására.

Az assembly rendszerek azonban csak igen egyszerü programozási nyelvek használatát tették lehetővé. Mégis lényeges előrelépést jelentettek, mivel a valóságos (bináris) cimek helyett lehetővé vált szimbolikus és relativ cimek használata is, anélkül, hogy a programozónak törődnie kellene ezek átkódolásával. A felhasználható müveletek köre azonban egyelőre nem bővült és az utasitások strukturája sem változott. Az assembly rendszereknél egy-egyértelmü megfeleltetéssel rendelték egymáshoz a hardware müveleteket és a szimbolikus utasitásokat, eltekintve a beépithető makroutasitásoktól és a forditás idején feldolgozásra kerülő pszeudoutasitásoktól (pl. memória-kijelölés). (Egyébként a makroutasitások csirája meg az volt, amikor a programozó -- eleinte csak a maga számára -- röviditéseket vezetett be gépikódutasitások gyakran előforduló sorozatai számára.)

Az assembly rendszerek a fejlődés fontos állomását jelentették. Szerepük megmaradt mind a mai napig, mert -- néhány kivételtől eltekintve -- a gépek hardware adta lehetőségeinek teljes és hatékony kihasználását ma is az assembly nyelvek biztositják, sőt a rendszer-programozásnak -- az ujabban használatba került compiler-iró és általánosabban rendszer-programozó nyelvektől eltekintve -- szinte kizárólagos eszközeivé váltak.

Ma az assembly nyelveknél -- figyelembe véve a korszerü assembly nyelveket is -- három szintet célszerü megkülönböztetni [41], az alább felsorolt jellemzőkkel:

1. abszolut assembly nyelvek

   - mnemonikus kódok használhatók a gépi utasitáskódok helyett,

   - betü vagy számkódok használhatók a valódi cimek helyett,

2. szimbolikus assembly nyelvek

- szimbolikus, relativ, literális és tömbcimzést engednek meg,
- pszeudoutasitások adnak kiegészitő információkat az assembler számára (pl. a különböző mezők elhelyezéséről a memóriában),

3. makro assembly nyelvek

- makroutasitások definiálhatók és aktivizálhatók,
- feltételes assembly utasitások használhatók (pl. nyomkövetelő utasitások).

(A fentiekben egyuttal áttekintettük a tipikus utasitásokat és cimzési módokat is az assembly szinten.)

Az egyes konkrét realizációknál természetesen a megadott szintek nem mindig különithetők el egyértelmüen, a jellemzők kombináltan is előfordulhatnak.

Kialakulásuk óta az assembly nyelvek és rendszérek igen jelentős fejlődésen mentek keresztül a cimzési lehetőségek és a makroutasitások alkalmazása, valamint a könyvtári rutinok és programmodulok használata tekintetében. A makroutasitások fejlődésének kiindulópontja a fix, gépi kódu nyitott szubrutin volt, alkalmazásuk pedig hosszu ideig az assembly nyelvekre korlátozódott. A fejlődés során két fontos tendencia figyelhető meg:

- a definiciós lehetőségek fejlődése, és
- a felhasználási terület szélesedése.

Eleinte csupán olyan makroutasitások voltak használhatók, melyeket korábban definiáltak (gépi kódban), és már a felhasználást megelőzően beépitettek a rendszer könyvtárába (nyitott vagy zárt szubrutin formájában). Ma e tipus mellett még megkülönböztetünk assemblerben definiált (abba beépitett) makrokat, továbbá programozó által definiált makrokat (amelyek jelenleg a legmagasabb szintet képviselik). Az utóbb emlitett tipusok első változataiban (az assemblerekben) csak a cim-

részek szerepelhettek formális (a hivásban aktuális) paraméterekként, a második változatban már az utasitások kódjai is; ilyen pl. az IBM MAP.

Strachey nevéhez füződik egy olyan általános makroprocesszor kidolgozása [42], mely tetszőleges szöveget enged meg a makrodefiniciókban, igy a paraméterek jellege sincs megkötve. Természetesen az ilyen tág értelmü makroutasitásokat magasabb szintü nyelveknél is igen jól lehet alkalmazni. (Makroutasitásokként foghatjuk fel pl. az ALGOL-60 eljárásait is.) Több olyan makroprocesszort is ismerünk, melyek magasabb szintü nyelveken irt célprogramokat produkálnak, bemenő nyelvük

[illegible] nyelvek pedig az adott magasabb szintű programozási nyelv, kiegészitve a szintaxis kibővitését eredményező makroutasitások lehetőségével. Ilyen pl. az MP/1 a FORTRAN-ra [43]. A gépi nyelvek egyik igen fontos jellemzője (ti. az operátorok körének kiterjeszthetősége szubrutinok segitségével) ily módon átvihetővé válik a magasabb szintű nyelvekre is.

A fejlődés következő lépése az volt, hogy (nem ragaszkodva a hardware-müveletekhez) a gépi nyelvekhez hasonló strukturáju nyelveket hoztak létre, melyek végrehajtása programozott interpreterekkel történt. Ezek közé tartozik a MIT-ben a Whirlwind gépre kifejlesztett "Comprehensive Algebraic Computer" ("általános algebrai számológép"), melyet Davis [40] az ALGOL egyik előfutárának tekintett. Ugyanerre az időszakra (1952-54) esik több automatikus kódoló rendszer kidolgozása is. Ilyen volt pl. az UNIVAC-ra készült A-2 compiler, mely háromcimes pszeudo-kódot forditott egycimes gépi kódra. Érdekes jelenség, hogy többnyire interpretereket alkalmaztak (melyek nem készitették el a végleges gépi-kód programot, hanem minden lépésben megvizsgálták a forrásprogram megfelelő elemeit és az általuk kijelölt müveleteket hajtották végre), bár compiler jellegü PLP-k is előfordulnak (ilyen az emlitett A-2 is). Ez utóbbiak gépi kódu célprogramot adnak eredményül, s azután ez kerül végrehajtásra. Az interpreterek tulsulya 1956-tól kezdve fokozatosan csökken, helyüket a kezdeti assemblerekből kinőtt, bonyolultabb strukturákat feldolgozó compilerek foglalják el. Megjegyezzük, hogy később -- a listafeldolgozó nyelvek kapcsán -- az interpreterek ismét előtérbe kerültek. Emellett fontos szerepet játszanak a gépek kompatibilitását célzó szimulációs rendszerekben, valamint uj, megtervezett de még meg nem épitett számológépek kipróbálásában, ill. software-jük elkészitésében is [19].

Már 1954-ben kidolgoztak egy univerzálisnak szánt kódolási rendszert is (pszeudo-kód szinten) az ENIAC, EDVAC és ORDVAC gépekre. (Ez volt az első müködő "géptől független" kódolási rendszer.)

2.6.3. Az eljárás-orientált nyelvek fejlődése

Az előzőekben tárgyalt nyelv-tipusok jelentős mértékben egyszerüsitették a feladatok gépre vitelének folyamatát. Ugyanakkor azonban

még mindig részletes ismereteket követeltek a felhasznált gépre vonatkozóan a programozótól, továbbá a feladatok megfogalmazásához használható irásmód is megtartotta a gépikód-programozás néhány kényelmetlen vonását. Fontos lépésnek tekinthető, hogy megjelentek a gépi utasitásstrukturától eltérő formákat is megengedő programozási nyelvek.

1953-ra Ljapunov és a vezetése alatt müködő csoport kidolgozta a Ljapunov-féle operátornyelvet, pontosabban egy olyan koncepciót, mely évekre meghatározta az automatikus programozás fejlődését a Szovjetunióban [45]. Ez a koncepció a programozási séma

definíciójan alapszik: egy adott problémára felirt programozási séma olyan operátorok és logikai feltételek program-realizációinak sorozata, amelyeket a szükséges adatokkal ellátottan számológépbe vive és azt elinditva megkapjuk a probléma megoldását. (Fel kell tételeznünk, hogy minden operátor számára a végrehajtáshoz szükséges program megtalálható a memóriában.) A megadott standard operátorok lehetővé teszik a feladatok egységes, géptől független leirását, bár egyes változatokban (elsősorban a vezérlésátadás leirását tartalmazókban) felismerhető, hogy egy ill. három cimes gépre készültek. (Ezzel egyidőben felmerültek és figyelemre méltó részeredményekhez vezettek a programozási munka és a célprogram optimalizálásának máig sem teljesen megoldott kérdései.) Az operátorséma fogalmát Janov fejlesztette tovább, elméleti eredményei a gyakorlat számára is igen hasznosak [46] e problémakör területén. 1954-ben egy aritmetikai kifejezéseket fordító program (Kurocskin, Koroljev), készült a BESZM-re, ugyanezen időben fejlesztették ki a PP-1 és PP-2 programokat (Sura-Bura, Ljubimszkij, Kaminyin és mások), melyek programozási sémákat dolgoztak fel [47]. A Ljapunov-féle nyelv továbbfejlesztett változatainak fordítására szolgáltak a Jersov által kidolgozott PLP-k, melyek a BEŞZM és a Sztrela-4 gépekre készültek (1954 és 58 között).

Ugyancsak 1954-ben indult el a FORTRAN fejlődése, ekkor dolgozta ki J. Backus az alapkoncepciókat. A későbbi IPL listafeldolgozó nyelvekkel kapcsolatos kutatások is ebben az időben kezdődtek el.

A következő években több nyelv (pl. a COBOL) és PLP kidolgozására is sor került, melyek nagy részét a szigoru értelemben vett gépre orientáltság jellemezte. Bár ekkorra esik az első folyamatvezérlő rendszer (APT--Whirlwind, MIT) megjelenése, mégis ezt az időszakot az eljárás-orientált nyelvek fejlesztése jellemzi.

Az eljárás-orientált nyelvek körébe azokat a nyelveket soroljuk, melyek végrehajtandó eljárásokat irnak le a feldolgozandó adatokra és az alkalmazandó eljárásokra vonatkozó fogalmak és eszközök segitségével. Főbb jellemzőik a következők:

- a felhasználónak tudnia kell specifikálni az alkalmazandó matematikai, logikai, stb. eljárásokat; a már leirt eljárásokra elnevezések vezethetők be, melyekre való hivatkozás az eljárás aktivizálását eredményezi;

- részletesen le kell irni a feldolgozandó adatokat.

Az eljárás-orientált nyelvek nagy sulya és elterjedtsége valószinüleg a digitális számológépekre vonatkozó Neumann-i koncencpió közvetlen következménye (a számológépipar e koncepcióin alapuló gépeket gyártott). Az eljárás-orientált nyelvek tervezői a szekvenciális müveletvégrehajtás általánositását -- a probléma megoldását produkáló eljárások szekvenciális végrehajtását -- fogadták el.

A tervezők csak lassan ismerték fel, hogy a programozási nyelvek tervezésénél és a PLP-kben célszerü matematikai nyelvészeti esz-

közöket is alkalmazni. Bár már az 1956-ig terjedő időszakban világossá vált, hogy a programozási nyelvek információtranszformátorok és ebből a szempontból az alkalmazott számitógépnek csupán másodlagos a jelentősége, mégis a hatvanas évekig a programozási nyelvek tervezésében többnyire konkrét gépekhez kapcsolódó, tehát szigoru értelemben vett gépi orientáltságu, programozási meggondolások domináltak. A nyelvészeti szempontok és eszközök használata esetleges és felszines volt. "Csak terminológiát kölcsönöztek a nyelvésztől" [40].

A programozási nyelv szabályait nagyrészt a PLP-k határozták meg, igen szoros összefüggésben a hardware-rel. Néhány kivételtől (pl. ALGOL) eltekintve a nyelvek és a PLP-k tervezése még ma sincs teljesen elválasztva. A fentieknek megfelelően az első FORTRAN, COBOL, stb. leirások terjedelmesek, nehezen kezelhetők, bonyolultak. Egyre nyilvánvalóbban jelentkezik az uj kezelésmódok kialakitásának szükséglete.

1957-ben állitották be az első FORTRAN-rendszert (IBM-704), amely az addigi legkomplexebb programozási rendszer volt. Ekkor kezdődött meg a COMIT (általános szimbolumkezelő) rendszer kidolgozása (IBM-704). A következő évben került sor a FORTRAN korszerüsitésére. Erre az időszakra tehető az első felhasználói csoportok kialakulása is. Ebben az időben már nagyon sok helyen foglalkoztak az automatikus programozás kérdéseivel. Számos automatikus kódolási rendszert dolgoztak ki, amelyek között a legtöbb szigoru értelemben vett gépre orientált programozási nyelv volt. Emiatt ezek sem válhattak általánosan elterjedt, univerzális programozási nyelvekké, annak ellenére, hogy az alábbi szempontokból már elég kényelmesek voltak programozásra:

- matematikai jelölésmódot alkalmaztak aritmetikai müveletek leirására;
- komolyabb megszoritások nélküli azonositókat használtak változók jelölésére;
- angol "vezényszavakat" alkalmaztak (transzput tevékenységekre, vezérlésátadásokra, stb.) ami jelentősen növelte a kényelmesebb felhasználás lehetőségeit.

Az előnyös tulajdonságuk mellett azonban a következő hátrányokkal is számolni kellett:

- nem bizonyultak eléggé általánosaknak, mivel összetett problémák leirásánál az egyes részek programozása igen kényelmetlen volt;
- az adatleirás igen nehézkes volt, amelyet kibővitése akadályokba ütközött, bár erre gyakran szükség lett volna a felismert uj igények és lehetőségek miatt.

A kialakult "bábeli zürzavar" ellen, a nyelv nagymértékben fokoztak az egyes kis számitógépgyártó cégek és e, mással nem kom-

patibilis gépek felhasználóinak csoportjai által bevezetett, gomba módra szaporodó "programozási argók", 1957-től kezdve szervezett mozgalmak indultak mind az USA-ban (ahol közben az IBM monopol-helyzetbe került), mind Európában, elsősorban a tudományos számításokra szolgáló nyelvek kifejlesztése céljából. A munkát a GAMM-on valamint a SHARE keretén belül alakult csoportok kezdték meg. Tevékenységüket elvileg az ACM kapcsolta össze. Az uj nyelvre, az ALGOL-ra, főleg a GAMM-csoport tevékenysége alapján, 1958-ban tették az első javaslatot. Ebben a nyelvnek már három "szintjét" különböztetik meg:

- a hivatkozási nyelvet,
- a publikációs nyelvet és
- a hardware nyelvet.

Még ugyanebben az évben az RRU M460 gépre beállitották a NELIAC fordítóját (a NELIAC az ALGOL 58, az Egyesült Államokban egyideig használt nevén IAL, egyik "dialektusa"). Egyidejüleg az IBM bejelentette, hogy elkészült az IPL-V fordítója a 650-es gépre.

A következő évben a legkiemelkedőbb fejlemény az ALGOL fejlődése és a többi nyelvre gyakorolt hatása volt. Ezzel párhuzamosan (1959-től) megindult az adatfeldolgozási nyelvek egységesitésére irányuló törekvés is.

Az ALGOL tekinthető az első PLP-től függetlenül tervezett nyelvnek, amelynek tervezése nyelvészeti szempontokat vesz figyelembe és nyelvészeti eszközöket használ. Nagy lépés volt a Backus által bevezetett formális leirási mód, az un. Backus-féle normálforma [48]. Később a segitségével leirható nyelvek osztályáról kimutatták, hogy bizonyos matematikai nyelvészeti kategóriákkal ekvivalens, nevezetesen a kontextus-független nyelvekkel [49]. Ezekből a munkákból igen gyümölcsöző kutatási irány ágazott ki, amely nagy segitséget jelentett a formalizálás és az egyértelműség kérdésének vizsgálatában is [50].

Az ALGOL fejlesztésébe később több ország is bekapcsolódott, pl. Anglia, Dánia (később igen jelentős szereppel!). A Szovjetunió-ban különállóan folyt a fejlesztés. (Ennek egyik eredménye a Kaminyin és Ljubinszkij munkássága nyomán megalkotott ALMO, géphez igazodó algoritmikus nyelv volt [51].) Érdemes megjegyezni, hogy a különálló fejlesztés ellenére az 1960-ra és később is elkészitett módositások, ill. javasolt fejlesztések lényegében megegyeztek a nyugateurópaiakkal. Egy másik figyelemre méltó jelenség az IBM ellenállása az ALGOL-lal szemben. A SHARE-en belül alakult ALGOL-lal foglalkozó csoport kérte az IBM-et fordítók kidolgozására. Ugyanekkor a FORTRAN-ban lényeges módositások

re, elsősorban az ALGOL hatására.

Az 1960-ban publikált "ALGOL report" olyan nyelvet ajánlott, amely a blokkstrukturával, az eljáráskezeléssel, a tömbök dinamikus kezelésével, a deklarációkkal és az összetett utasitással igen reménytkeltő volt, de leirása még számos hibát és többértelmüséget, maga a nyelv számos következetlenséget is tartalmazott. Még ez évben elkészült az első fordító is Hollandiában az Electrologica XI gépre. Az ALGOL elterjedése mégsem ment zökkenőmentesen. Az IBM — az első elutasitást követően — több kisérleti fordítót is produkált, de felhasználóinak nem adta át. Közben a SHARE-en belül is több izben lankadt az érdeklődés az ALGOL iránt. Európában sem volt olyan szerv, mely az elterjesztést kézben tartotta volna, vagy a fordítók kidolgozását irányitotta volna.

Az 1963-ban megjelent (1962-es eredetü) módositott ALGOL jelentős több korrekciót és pontositást tartalmaz. Ezt követően az ALGOL elterjedése meggyorsult, egymás után készültek a fordítóprogramok. Az IBM is változtatott álláspontján és hozzáférhetővé tette felhasználói számára az (egyébként nagyon kevéssé hatékony) ALGOL-fordítóit. A Szovjetunióban két rendszer készült az ALGOL "szibériai nyelvjárásnak" megfelelően [52]. 1962-ben az IFIP-en belül munkacsoportot alakitottak az ALGOL felügyeletére.

Az ALGOL példáján világossá vált, hogy bármennyire célszerü is egy programozási nyelv, elterjesztésében jelentős szerepe van a számológépeket gyártó cégeknek, mivel a felhasználók csoportjai nem tudnak (vagy nem hajlandók) megfelelő energiát forditani hatékony fordítók kidolgozására.

Az ALGOL kapcsán végzett munka hatására egységesitési törekvések mutatkoztak meg a programozási nyelvek leirása tekintetében, ha már magukban a nyelvekben ez nem sikerült. A Backus-féle normálforma segitségével több nyelv egységes leirását végezték el. Az előtérbe került nyelvészeti kezelésmód jelentős haladást hozott a hatékony fordítók készitésében (szintaxis-orientált fordítók). Több gondot fordítottak a szemantikai és pragmatikai kérdések vizsgálatára is, igy pl. kialakult a "bécsi leirási nyelv" [53], [54], [55], [56], [57], [58], amely szabatos módszert ad a nyelv szintaxisa és szemantikája egységes leirására. Megindultak és jelentős eredményeket mutattak fel a

a fordítástechnikai kérdésekkel kapcsolatos kutatások (pl. lengyel jelölés, optimalizálási kérdések stb.).

Az ALGOL processzorok kapcsán számos elv nyert hardware realizációt (pl. veremmemória, a név szerinti hivásnak megfelelő operandusz-kezelés stb.).

1964-től kezdve lanyhult az érdeklődés az ALGOL iránt, de hatása még ma is igen erős. Az IBM versenytársként a PL/1 nyelvet állitotta vele szembe, mely nagyon sok elemet tartalmaz az ALGOL-ból. Ugy tünik, hogy a PL/1 valóban sok korábbi nyelv előnyeit egyesiti magában, alapkoncepciójában pedig az ALGOL-ra támaszkodik. Pillanatnyilag ez a leginkább elterjedt fejlett eljárás-orientált nyelv.

Az IFIP-en belül már 1963-ban megkezdték az ALGOL tovább-

fejlesztését, s ennek eredménye volt az ALGOL-68. Ez nem egyszerüen kiterjesztése az ALGOL-60-nak, hanem az ott hasznosnak bizonyult elemeket olymódon tartalmazza, mint általánosabb kategóriák speciálisabb eseteit. Az ALGOL-68-ban azonban számos olyan kategória is szerepel, amelyeknek nincs korábbi megfelelőjük. Emiatt valóban lényegesen több lehetőséget tartalmaz, mint elődei, de tul általános volta miatt nem számithatunk gyors elterjedésére a gyakorlati használatban.

x x x

Az eddigiekben főleg a tudományos számitásokra tervezett eljárás-orientált nyelvek fejlődésével foglalkoztunk (elsősorban ugyanis ezek határozták meg hosszu időn át a fejlődés főbb vonalait), amelynek eredménye a FORTRAN, az ALGOL és a PL/1 volt. A fejlettebb változatokban, implementációkban a szekvenciális végrehajtás mellett megjelentek a parallel-szekvenciális feldolgozást lehetővé tevő speciális elágazó, csatlakoztató, hozzáférést tiltó, engedélyező, várakozó stb. utasitás tipusok is (PL/1, ALGOL-68).

Ettől a vonaltól viszonylag függetlenül fejlődtek a gazdasági alkalmazásokra szolgáló nyelvek, továbbá a listafeldolgozó nyelvek. Természetesen számos kölcsönhatás is érvényesült, a későbbiekben pedig lényeges közeledés figyelhető meg a három vonal között. A tudományos számitásokra szolgáló nyelvekben pl. lényegesebb szerepet kapott a I/O tevékenységek szervezése. Ez a tendencia indokolt, mivel a számitási feladatok nem skatulyázhatók be mereven egy tipusba, több területről származó elemeket tartalmazhatnak, ami a számukra hatékony nyelvekkel szemben egyre inkább az univerzalitás igényét veti fel [59].

A listafeldolgozó nyelvek fejlődésének kezdeteként az IPL sorozat megindulását tekinthetjük (1954), amelyet első izben 1958-ban implementáltak (IBM 650). E területen igen jelentős esemény volt a LISP megjelenése 1960-ban [60]. Bár elméleti szempontból igen elegáns ez a nyelv, praktikus programozási feladatokra mégis csak kevéssé bizonyult alkalmasnak [61]. Több kiegészités után (melyek a felhasználók igényeinek nyomására közelitést jelentettek az ALGOL-hoz) alakul ki a LISP 1.5., majd a LISP2, melyek a gyakorlati szükségleteket már jobban kielégitik. A LISP 2 már viszonylag végleges formának tekinthető a listafeldolgozó nyelvek területén.

A gépi forditás céljaira kidolgozott COMIT (1957-ben tervezték a Massachusetts Institut of Technology-ban az IBM 704-re), képezte a SNOBOL alapját (1964) mely később továbbfejlesztve ugyancsak elterjedt nyelvvé vált. Ezek mellett még számos listafeldolgozó nyelvet készitettek, melyek hatására listafeldolgozó elemek jelentek meg pl. az ALGOL-ban (FORMULA ALGOL) és a PL/1-ben is. A listafeldolgozó nyelvek lényeges eltérést mutatnak az általuk kezelt listák általánossága tekintetében (idézetek, karakterek, egyszintü listák, többszintü listák, speciális listák).

További lényeges különbségek találhatók a nyelvek szintjében (pl. az EOL assembly szintü, a LISP magasabb szintü), valamint a belső tárolásra vonatkozó információk megadhatóságában

(a LISP-ben a rendszer végzi a memória hozzárendelést, ezt az L6-ban a programozó adja meg [62], ami a kötetlenség ellenében természetesen programozási kényelmetlenségekkel jár).

Igen érdekes, hogy a listafeldolgozó nyelvek területén még nem jelentkezett számottevő egységesitési törekvés [61], ellentétben a gazdasági és a tudományos feladatokra orientált nyelvekkel, bár két nyelv -- a LISP és a SNOBOL -- szélesebb körben terjedt el, mint a többi.

Az eljárás-orientált nyelvek fejlődésének eredményeképp az ALGOL-68 lényegében egyesiti magában a legfontosabb lehetőségeket, kielégiti az igényeket az univerzalitás és részben a párhuzamositás tekintetében is. Ugyanez elmondható a PL/1-ről is, mely ugyan alacsonyabb szintet képvisel, használata azonban mégis elterjedtebb. Nem állithatjuk mégsem, hogy a gyakorlati fejlődés iránya az univerzális nyelvek felé mutat, mivel a felhasználók nem elsősorban az algoritmusokat, hanem a problémákat leiró nyelveket igénylik és olyan processzorokat, melyek ezeket képesek feldolgozni. Természetesen ez mellett (sőt ennek érdekében) szükség van az eljárás-orientált nyelvek fejlesztésére is, mivel -- elméleti szempontok mellett -- pl. a probléma-orientált nyelvek bonyolult processzorainak leirását is nagymértékben megkönnyitik.

Végül röviden foglalkozunk az un. dialógus nyelvekkel, melyek az eljárás-orientált nyelvek osztályába tartoznak, de feladataik és a gépen belüli kezelésmódjaik specifikussága miatt megkülönböztetett figyelmet érdemelnek. E területen legelterjedtebb az APL rendszer, mely az Iverson által konstruált programozási nyelvre épül [63], és amelyet Falkoff és Iverson implementáltak az IBM 360-as rendszerre (APL\360) [64]. Nyilvánvaló, hogy a konverzációs üzemmódot megengedő terminálok elterjedése olyan -- a korábbiaktól eltérő -- nyelvek igényét vetette fel, melyek segitségével a felhasználók által megszokott irásmódhoz közelálI, egyszerü, könnyen elsajátitható formában közölhetők a feladatok a géppel. Egyszerüsége mellett azonban lehetővé kell tennie összetettebb feladatok kényelmes gépreviteIét is. Azt kellett megoldani, hogy a számitógép asztali számitógépként és nagyteljesitményü rendszerként is használható legyen, egy ugyanazon nyelv alkalmazásával. E követelményeket elé-

giti ki az APL. A feladatok leirására szolgáló utasitások mellett az üzemmódot (végrehajtás, ill. definició) meghatározó utasitásokat is tartalmaz és tág lehetőséget biztosit a programok módositására, javitására, továbbá a programok interaktiv belövésére, de tekintettel előnyeire az APL mutatkozik a legperspektivikusabbnak.

2.6.4. A probléma-orientált nyelvek fejlődése

Eltérően az előzőekben tárgyalt nyelvektől, a probléma-orientált nyelvek a megoldandó problémát annak terminológiájával, kifejezéseivel és utasitásaival irják le, egyes esetekben pedig a kivánt eredménnyel kapcsolatos fogalmakat is felhasználnak. A probléma-orientált nyelvek magasabb absztrakciós szintet képviselnek

mint az eljárás-orientált nyelvek, ennek megfelelően a gépi nyelvektől igen távol állnak. Ezért processzoraikat többnyire ugy készitik, hogy első lépésben valamilyen a forrásnyelvről magasabb szintü nyelvre transzformálja az információt, és csak a második lépésben történik meg a gépre forditás. Ugyancsak processzoraik bonyolultsága magyarázza viszonylagos fejletlenségüket, és azt, hogy a növekvő igények ellenére sem terjedtek el kellően.

A fejlődésben ez első lépését az APT jelentette, melyet gépipari folyamatvezérlési alkalmazásokra fejlesztettek ki és a MIT-ben implementáltak a Whirlwindre (1955-56). Kezdetben elsősorban folyamatvezérlési feladatoknál használtak probléma orientált nyelveket. Alkalmazásuk csak a hatvanas évek elejétől vált szélesebb körüvé [40]. Ugyancsak ettől az időtől számottevő a dialógus-rendszerek fejlődése. Az utóbbiak szoros kapcsolatban állnak a probléma orientált nyelvek fejlődésével, mivel processzoraik többnyire dialógus üzemmódban dolgoznak.

A fejlődés mai fokán nehéz lenne egységes rendszerbe foglalni a probléma-orientált nyelveket, ezért a továbbiakban csupán néhány területről vett példákkal illusztráljuk a fejlődést.

Elsőként a szimuláció kérdésével foglalkozunk, melynek kezdeti formája az analóg szimuláció volt. Később -- a számitógépek felhasználásával párhuzamosan -- megjelent a digitális szimuláció is, de mai ismereteink szerint a hibrid rendszerek tekinthetők a leghatékonyabb eszközöknek. A szimulációs problémák nehezen fogalmazhatók meg eljárás-orientált nyelveken, ezért a felmerült igény nyomán megjelentek a szimulációs nyelvek. Ezek kezdetben magukon viselték az analóg szimuláció nyomait (szervezésükben, az idő figyelembevételében, mely itt jelentős szerepet játszik -- rendszeridő), később egy, a digitális számitógépek lehetőségeit jobban figyelembe vevő szemlélet alakult ki, melyet az jellemez, hogy diszkrét ese-

ménysorozatokkal dolgozik. Ez jelentős gyorsitást eredményezett a szimulációs futtatásokban, mivel itt a rendszeridő egy lépésben nem egy előre meghatározott kis időintervallummal változik (ami a folytonos rendszeridő elemi szakasza), hanem azt az időpontot veszi figyelembe, melyben a legközelebbi esemény bekövetkezik.

A leirandó modellek sajátosságai miatt az általános célu eljárás-orientált nyelvekhez viszonyitva számos uj elemmel találkozunk (pl. ilyen a párhuzamos folyamatok leirásának lehetősége). Mint általában a probléma-orientált nyelveké, a szimulációs nyelvek feldolgozása is ugy történik, hogy előbb valamilyen szintü eljárás-orientált nyelvre kell elvégezni a forditást, majd ez kerül végrehajtásra, egyes esetekben interpretációs technikával (pl. GPSS). A SIMSCRIPT a FORTRAN-ra, a SIMULA az ALGOL-

ra épülő, szimulációs nyelv, megengedve a szimulációs programban az alapnyelv használatát is.

A hibrid rendszerek fejlődésével olyan eszközhöz jutott a szimuláció, mely lehetővé teszi mind a diszkrét, mind a folytonos folyamatok hatékony vizsgálatát. Ennek természetesen software vonatkozásai is vannak. Jelentős előrehaladásról azonbannem beszélhetünk, inkább csak a jövőbeli fejlődés irányaként jelölhetjük meg.

A gazdasági, adatfeldolgozási területen is keletkeztek probléma-orientált nyelvek. 1962-ben tették közzé a DETAB-X-et, mely a problémák leirását döntési táblázatok formájában teszi lehetővé. Processzorának célnyelve a COBOL. Megjelenését több hasonló rendszer előzte meg: TABSOL (1960-61), LOGTAB (1961) stb. [65], és a későbbiekben is számos adatfeldolgozást elősegitő probléma-orientált rendszer született.

Végül pédaként megemlitünk néhány ujabb kisérletet, melyek a mesterséges intelligencia oldaláról közelednek a kérdéskör felé, felhasználva a matematikai nyelvészet eredményeit. 1971-ben számoltak be olyan dialógus-rendszerrel kapcsolatos kutatásokról és kisérletekről, melyek angol nyelven megadott elemi (főként kombinatórikai valószinüségszámitási problémák megoldására irányulnak [66]. Ezeknél a programrendszert LISP 1.5-ben irták fel IBM gépre, interpretációs technikát alkalmazva. Az ilyen jellegü rendszerek általános jellemzésére tesznek kisérletet [35] -ben, egy természetes nyelvü problémamegoldó rendszer kapcsán.

A legfontosabb jellemzők a következők:

- általános a heurisztikus elemek használata a PLP-kben;
- rekurziv rutinok és veremmemória alkalmazása az igényelt tevékenység megállapitására;
- a természeteshez közelálló nyelv alkalmazása bemenő nyelvként;

- a nyelvi elemek jelsorozatok halmazával való reprezentálása, amely alkalmas az általánosság biztositására;
- interaktiv üzemmódban való müködés.

Az eddigi eredmények alapján ezt az irányt reménytkeltőnek tekinthetjük, annak ellenére, hogy jelenleg még csak a kisérletek szintjén mozog és gyakorlati alkalmazására szélesebb körben még nem került sor.

A probléma-orientált nyelvek területén nehéz eldönteni, hogy mely esetekben beszélhetünk "nyelvről", hiszen minden alkalmazási programnak, programrendszernek van valamilyen, néha csak bemenő nyelve, és csak ezek szintjében van lényeges különbség. Ennek megfelelően igen sok e kategóriába tartozó rendszert lehetne felsorolni. E helyett megkisérlünk néhány általános érvényü észrevételt kiemelni.

- A felhasználó általában azt igényli, hogy problémáit saját terminológiája és logikája szerint közölhesse a számítógéppel (a felhasználón persze nemcsak a programtervezőt és programozót értjük, hanem általában a számítógépes feladatokat megfogalmazó embereket), tehát igen lényeges a probléma-orientált nyelvek és rendszerek területének fejlesztése, sőt ezen tulmenően a számítástechnikai kultura elterjesztése a potenciális felhasználók széles körében.
- A probléma-orientált nyelveknek célszerű magasabb szintű nyelvet választani, lehetőleg az emberi nyelvhez közel állót, hogy könnyen elsajátithatóak legyenek..

3. A KALMÁR-FÉLE FORMULAVEZÉRLÉSŰ SZÁMITOGÉPPEL KAPCSOLATOS KORÁBBI ELGONDOLÁSOK

3.1. Bevezetés

Az előző fejezet végén a programozási nyelvek fejlődésével kapcsolatban, áttekintettük a PLP-k fejlődését is. A PLP-k ("software PLP-k") egyik lehetséges, és ma általánosan használt megoldását szolgáltatják annak a 2.6. alfejezetben (a 2-54. oldalon) emlitett ellentmondásnak, amely a felhasználó számára kényelmes nyelvek és a számitógép által "közvetlenül" (hardware uton, nem záva ki természetesen a mikroprogramozott hardware-

műveletvégzőket) megérthető nyelv (gépi kód) között teremt. Ebben a fejezetben a másik megoldással, a formulavezérlésű számítógépekkel ("hardware PLP-kkel") foglalkozunk, tekintettel arra a lehetőségre, hogy a számítógépek további fejlődése ismét aktuálissá teszi majd az e téren végzett kutatások gyakorlati hasznosítását (ha nem is az eredeti formájukban, hanem sokkal valószínűbben, az említett ellentmondás software és hardware megoldásának valamilyen szintézise formájában).

Formulavezérlésű számítógépen e tanulmányban olyan digitális számítógépet értünk, amelynek belső nyelve (gépi kódja) izomorf valamely magasabb szintű, rendszerint eljárásra orientált programozási nyelvvel, amelyet a továbbiakban a gép nyelvének nevezünk. A formulavezérlésű számítógép tehát a gép nyelveként használt formulanyelven irt programokat forditóprogram nélkül dolgozza fel ugy, hogy a program minden egyes alapjelét (pl. egy-egy betűt, számjegyet, műveleti jelet, zárójelet, vagy pl. az ALGOL nyelv esetén az if, then, else, for, step, until, stb. alapjeleket), amely megfelelően kódolt formában kerül bele az utasításregiszterébe, egy-egy utasításnak tekinti és ezeknek az utasításoknak (a lineáris sorrendtől eltérést előíró (pl. vezérlésátadó, ciklus-) utasítások figyelembe vételével meghatározott dinamikus sorrendben tekintve azokat) az eredője mindig éppen a program által előirt gépi tevékenység.

A formulavezérlésű számítógépek gondolata az 1950-es évek vége felé merült fel, egymástól függetlenül több helyen. Igy Bauer és Samelson (München, majd Mainz) 1957-ben szabadalmi igényt jelentettek be a Német Szövetségi Köztársaságban (B 44 122 sz. alatt, 1957 március 30-án) hardware uton (a tolóregiszterek -- shifting registers -- elvén) megvalósitott veremmemóriára és annak alkalmazásaként létrehozható, aritmetikai és logikai formulákat közvetlenül feldolgozó digitális számítógépre. (Nyugatnémet szabadalmat nem kaptak rá, azonban a szabadalmi igénybejelentés biztositotta elsőbbség révén, a hiányzó előzmény-kutatás miatt,

1958-ban sikerült erre angol, olasz és francia szabadalmat kapniok l. [71]; ez a publikáció akkor jelent meg, software oldalról aknázva ki a gondolatot, amikor a hardware-megoldás nyugatnémet szabadalmaztatása meghiusult.) Kämmerer (Jena és Berlin) a Jenai Egyetemhez benyujtott habilitációs értekezésében vetette fel matematikai formulakép alapján programozható számítógép gondolatát [72]. Kalmár 1959-ben Varsóban [73], majd 1960-ban Freibergben a Gesselschaft für Angewandte Mathematik und Mechanik évi ülésszakán [74] és a Második Magyar Matematikai Kongresszuson [75] számolt be formulavezérlésű számítógép tervezésére vonatkozó kutatásairól. Pawlak (Varsó) 1960-ban [76], Gluskov (Kijev) és munkatársai 1963-ben kezdtek hozzá formulavezérlésű számítógépek megvalósitásához [77]. A kérdés jelenleg sem vesztette el aktualitását; igy pl. Lee (Amherst, Mass., USA) a University of Massachusetts

Computer Science Program-ja keretében jelenleg a FORTRAN eljárásra orientált programozási nyelv WATFOR változatán irt programokat fordítóprogram nélkül feldolgozni képes számítógép megvalósitásán dolgozik [78].

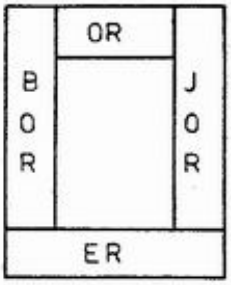
A formulavezérlésü számítógépek gondolatának felmerülésében nyilván azok a nehézségek játszottak szerepet, amelyek egyrészt a gépi kódban való programozásnál, másrészt a magasabb szintü programozási nyelvek implementációjához szükséges PLP-k szerkesztésénél, továbbá akár a gépi kódban irt felhasználói programokban, akár a PLP-kben a hibák megtalálásánál és kiküszöbölésénél jelentkeztek.

A továbbiakban Kalmár László korábbi elgondolásait ismertetjük egy formulavezérlésü számítógép megvalósitására nézve. Minthogy Kalmár László annak idején azt a tanácsot kapta a Matematikai Kutató Intézet igazgatójától és gazdasági vezetőjétől, hogy formulavezérlésü számítógépre vonatkozó előkisérlet céljából (amelynek során kellett volna tisztázni a végleges formulavezérlésü számítógép optimális tervét) minél olcsóbb, de máris a formulavezérlés elvén müködő számítógép megépitésére kérjen engedélyt (erre vonatkozó, a Matematikai Kutató Intézet igazgatóján keresztül az illetékes akadémiai szervekhez 1963 április 8-án benyujtott kérésére, amelyet az Intézet 1963 április 13-án továbbitott, nem jött válasz), ezért a formulavezérlésü számítógép nyelvéül a Ljapunov-féle oprátor-nyelvnek egy, az indexes változók és a ciklus-utasitások linearizálásában az ALGOL nyelvet követő, nagyjából assembly szintü programozási nyelvet választott. Ehhez hozzájárult az, hogy 1959-ben, amikor vizsgálatait elkezdte, az ALGOL nyelv még csak az 1958 évi előzetes változatában létezett és jelentősége még nem volt látható. Ezért kutatásaiban ilyen Ljapunov-szerü nyelvet vett alapul és később plauzibilis volt az előkisérletek céljából olcsón megépithető formulavezérlésü számítógép nyelveként az 1959-ben alapként tekintett nyelvet választani csekély módositással. Azonban elgondolásai a gép nyelvének választásától nagymértékben függetlenek.

3.2. Aritmetikai kifejezések feldolgozása; cim nélküli számítógépek

A szokásos (nem formulavezérlésü) számítógépek belső nyelve és a formulavezérlésü számítógépek nyelveként használható formulanyelvek között a legfeltünőbb különbség az, hogy az utóbbiakban egyetlen utasitással egy egész aritmetikai vagy logikai kifejezés értékének kiszámitása és (valamely változó értékeként való) tárolása előirható. Ezért mindenekelőtt a (Kalmár-féle elgondolás szerinti) formulavezérlésü számítógép "kifejezés-feldolgozó egységét" ismertetjük. Ez -- az elgondolás eredeti, régebbi változata szerint (ugyanis később egy bonyolultabb szerkezetü, de kevesebb regisztert igénylő ujabb változat is tervezésre került) -- egy regiszter-négyesekből felépitett hierarchikus struktura. A regiszter-négyesek egy-egy baloperandusz-regiszterből (BOR), operátor- (müveleti jele) regiszterből (OR)

ról- (műveleti jel-) regiszterből (OR), jobboperandusz-regiszterből (JOR) és eredmény-regiszterből (ER) állnak (lásd 1. ábra; az ábra közepe nem jelent regisztert). A bal- és jobboperandusz-regiszterek, valamint az eredmény-regiszterek hossza a gép szóhosszával egyezik meg, az operátor-regiszter hossza pedig a gép nyelve alapjeleinek (bináris) numerikus kódja hosszával. A hierarchikus struktura abban áll, hogy a regiszter-négyesek "emeletekbe" vannak rendezve, és a legalsó kivételével minden egyes emeleti ER egy-egy kapuáramkörrel össze van kötve az eggyel alacsonyabb emeleti BOR-rel és JOR-rel ugy, hogy e két kapu közül legfeljebb az egyik lehet nyitva. Az összekötés azt jelenti, hogy a kérdéses ER tartalma (amennyiben nem üres, l. később) automatikusan betöltődik az eggyel alacsonyabb emeleti BOR és JOR közül abba, amellyel nyitott kapuval van összekötve (ha van ilyen), majd a kérdéses ER emeletén mind a négy regiszter kiürül és az addig nyitott kapu bezárul. A regiszter-négyesek ugyanakkor a gép aritmetikai egységének müveletvégző részével is össze vannak kötve. Ezen összeköttetés abban áll, hogy valahányszor valamely JOR megtelik (vagyis az üres állapotból valamely számot tartalmazó állapotba megy át), a kérdéses emeleti BOR és JOR tartalma automatikusan betöltődik azon müveletet végző alegység megfelelő bemenő regisztereibe, amely müvelet jelének numerikus kódja megegyezik a kérdéses emeleti OR tartalmával, majd, miután ez az alegység végrehajtotta a kérdéses müveletet, annak eredménye automatikusan betöltődik a kérdéses emeleti ER-be (ahonnan azután, amennyiben ez az ER nyitott kapuval van összekötve az eggyel alacsonyabb emeleti BOR és JOR valamelyikével, a fentiek szerint abba töltődik be; amennyiben az eggyel alacsonyabban lévő



1. ábra

emeleti JOR-ba, akkor ez a folyamat eggyel alacsonyabb emeleten ismétlődik).

Ebből világos, hogy lényeges valamely regiszter üres (alap-) állapotát valamely számot (akár a 0-t) tartalmazó állapotától megkülönböztetni. Ezért az előjeles számok direkt kódos ábrázolását használjuk; az "aritmetikai" 0 negativ előjellel van ábrázolva (előjelbit 1, a többi bit 0), mig a pozitiv előjelü 0 (minden bit 0) az üres állapot jele.

A kifejezés-feldolgozó egység müködésének ismertetéséhez tekintsük először az olyan aritmetikai kifejezés esetét, amelyben csak skaláris változók fordulnak elő (indexes változók nem, és konstansok sem). Egyszerüség kedvéért tegyük fel, hogy minden skaláris változó jele egyetlen betü (egykarakteres azonositó), esetleg latin betük mellett görög és egyéb betüket is megengedve. Ez esetben célszerü a skaláris változók értékét az operativ memória kü-

lön tartományában, az un. skaláris memóriában elhelyezni, mégpedig minden skaláris változó értékét abba a rekeszbe, amelynek címe a kérdéses skaláris változó (betű) numerikus kódja. Egyszerűség kedvéért tegyük fel továbbá, hogy az aritmetikai kifejezések un. teljes zárójelezéssel vannak felirva a következő szintaxis szerint:

⟨aritmetikai kifejezés⟩ ::= ⟨operandusz⟩ | ⟨operandusz⟩ ⟨operátor⟩ ⟨operandusz⟩

⟨operandusz⟩ ::= ⟨skaláris változó⟩|(⟨operandusz⟩⟨operátor⟩ ⟨operandusz⟩)

(pl. ab + cd nem a x b + c x d, hanem (a x b) + (c x d) alakban van irva). Itt egyszerüség kedvéért csak kétoperanduszu operátorokat vettünk figyelembe. Tegyük fel továbbá, hogy az (egyszerü) értékadó utasitások nem

⟨skaláris változó⟩ := ⟨aritmetikai kifejezés⟩ , hanem

⟨aritmetikai kifejezés⟩ ⇒ ⟨skaláris változó⟩ alakban vannak felirva.

A gép nyelvén felirt program, amely a memóriában kódolt alakban ("programvektor" alakjában) ván tárolva, egyes alapjeleinek numerikus kódjai (a programvektor komponensei, vagy, ha egy gépi szóban több -- az eredeti elgondolás szerint négy -- alapjel numerikus kódja fér el, e komponensek megfelelő szeletei) dinamikus sorrendben egymás után mennek be a gép utasításregiszterébe (UR). A gép bizonyos regiszterei, többek között a BOR-ek, az OR-ek és a JOR-ek, sorszámmal vannak ellátva. A vezérlő egység egy speciális regisztere, az aktiváló regiszter (AR), mindig egy ilyen "sorszámozott" regiszter sorszámát tartalmazza; amelyikét, azt az (adott pillanatban) "aktiv regiszternek" nevezzük. Azt az utasitást, amelyet

a gép akkor végez, ha valamely alapjel numerikus kódja megy be az UR-be és ugyanakkor valamely regiszter aktiv, a kérdéses alapjelhez és aktiv regiszterhez tartozó utasitásnak nevezzük (vagy a kérdéses alapjelhez tartozó utasitásnak, amikor a kérdéses regiszter aktiv). Az utasitáshoz, más tevékenységen kivül, az is hozzátartozik, hogy mely regiszter "aktiválódjék", azaz melyiknek a sorszáma töltődjék be az AR-be. Kezdetben (a gép inditársakor), valamint minden egyes utasitás végrehajtása után a legalsó emeleti BOR az aktiv regiszter.

Előre megemlitjük még, hogy indexes változó feldolgozása folyamán az AR-nek az indexek értékének kiszámitása során van szerepe; ezért előbbi tartalmát az összes indexek értékének kiszámitásáig a vezérlő egység egy "várakozó regisztere" (VR) őrzi. A most tárgyalt esetben, amikor az aritmetikai kifejezésben nem szerepel indexes változó, VR üres marad.

A kifejezés-feldolgozó egység működésének ismertetéséhez

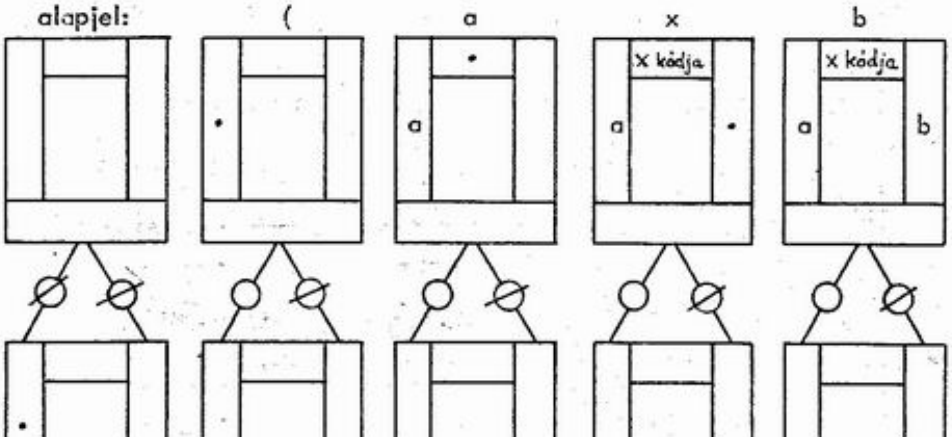
a választott esetben a gép utasításrendszerének következő részére van szükség (BOR_K, OR_K, JOR_K, ER_K a K-adik emeleti BOR, OR, JOR, ill. ER; a legalsó emelet 0-adiknak tekintjük):

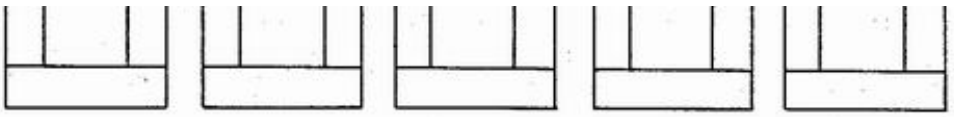
Alapjel	Aktiv regiszter	Egyéb feltétel	U t a s i t á s	Aktiválandó regiszter
(	BOR_K vagy JOR_K		ER_{K+1}-et az aktiv regiszterrel összekötő kapu kinyilik	BOR_{K+1}
⟨skaláris változó⟩	BOR_K		Azon memóriarekesz tartalma, amelynek cime UR tartalma, betöltődik a BOR_K-ba	OR_K
	JOR_K	VR üres	Azon memóriarekesz tartalma, amelynek cime UR tartalma, betöltődik JOR_K-ba	OR_λ, ahol λ a legnagyobb olyan K-nál kisebb index, amelyre OR_λ üres volt; ha ilyen nincs, akkor ER_0
	ER_0		ER_0 tartalma tárolódik azon memóriarekeszbe, amelynek cime UR tartalma, majd ER kiürül	BOR_0

Alapjel	Aktiv regiszter	Egyéb feltétel	U t a s i t á s	Aktiválandó regiszter
⟨operátor⟩	OR_K		UR tartalma betöltődik OR_K-ba	JOR_K
)	Bármelyik		Üres utasitás	A régi aktiv regiszter marad aktiv
⇒	ER_0		BOR_0, OR_0, JOR_0 kiürül	ER_0
	OR_0		BOR_0 tartalma betöltődik ER_0-ba, majd BOR_0 kiürül	ER_0

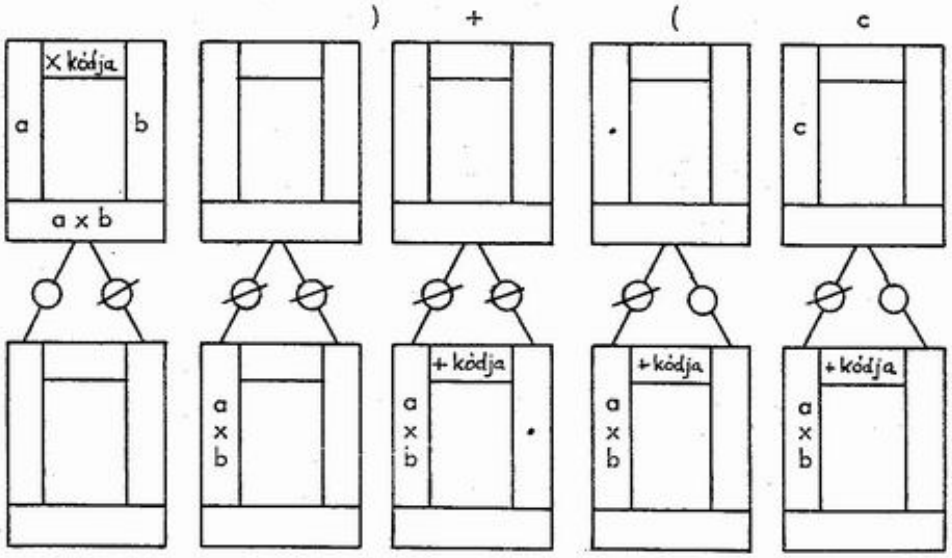
Természetesen a gép minden tényleges megvalósitása esetén csak véges számu (az eredeti tervek szerint 8) emelet van. Ezért, ha BOR_7 vagy JOR_7 aktiv és a programban kezdőzárójel következik, akkor megszakitás következik be és a gép "zárójelmélység-tulcsordulás" hibaüzenetet ad. Az ilyen programhibán ugy lehet segiteni, hogy az aritmetikai kifejezés egy alacsonyabb zárójelmélységü része számára "uj jelölést vezetünk be", azaz e részt egy addig nem szerepelt skaláris változónak adjuk érték gyanánt és az eredeti programban a kérdéses részt e skaláris változóval pótoljuk.

Az aritmetikai kifejezésnek a fenti szintaxis szerinti felépitését követő teljes indukcióval könnyen bebizonyitható, hogy ilyen utasitásokat tartalmazó utasitásrendszerü gép az ⟨aritmetikai kifejezés⟩ ⇒ ⟨skaláris változó⟩ alaku értékadó utasitásokat helyesen hajtja végre. A bizonyitás részletezése helyett ezt egy példán, az (a x b) + (c/d) ⇒ U utasitás példáján mutatjuk meg. A 2. ábrán az egyes oszlopokban a kifejezés-feldolgozó egység alsó két emeletének állapota látható az egyes alapjelek feldolgozása során; a pont az aktiv regisztert jelzi; O nyitott, Ø zárt kaput jelent.

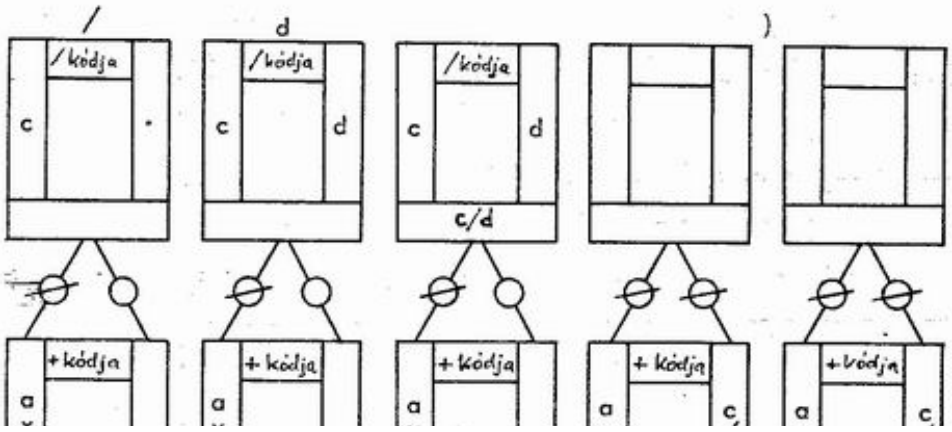


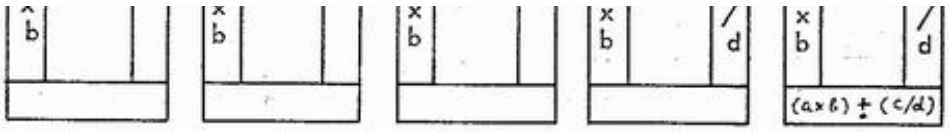


2. ábra (lásd folytatását is).

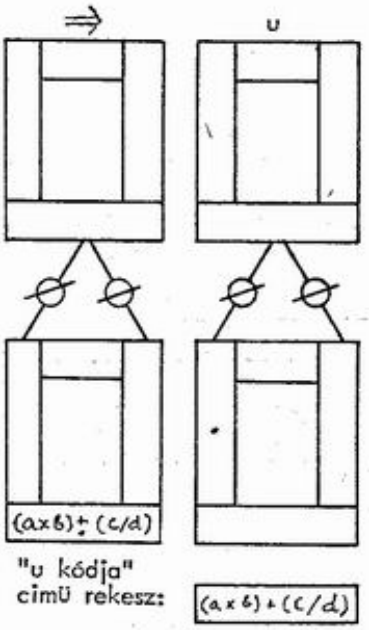


2. ábra, folytatás (lásd II. folytatását is).





2. ábra, II. folytatás (lásd III. folytatását is).



2. ábra, III. folytatás.

Már az eddigiekből látható, hogy a formulavezérléses számítógépek regiszter-igénye sokkal nagyobb, mint a hagyományos számítógépeké. Ez az első és második generációs számítógépek korszakában kételyeket támasztott a formulavezérlésü számítógépek realizálhatósága iránt. Épp ezért Kalmár László az előkisérletek céljából olcsón megépithető formulavezérlésü számítógép legtöbb regisztere gyanánt (az operativ memóriától, amely a kisérleti gépben szintén mágnesdob lett volna, független) mágnes dob rekeszeit (alkotóit) tervezte felhasználni. Ma már azonban, amikor viszonylag olcsón lehet kapni gyors MSI vagy LSI monolitikus regisztereket, nincs értelme ilyen "szükségmegoldásnak", amely nagymértékben rontotta volna a kisérleti gép müvelet-végző sebességét, minthogy egy-egy utasitás végrehajtása esetleg több regiszterhez, ilyen megoldás esetén tehát többszöri mágnesdobhoz fordulást igényel.

A fenti utasitástáblázatból látszik, hogy a skaláris változókhoz, mint alapjelekhez tartozó utasitásoktól (tehát az operandusz-regiszte-

rekbe töltő és az eredmény-regiszterekből tároló utasitásoktól) eltekintve a formulavezérlésü számitógép utasitásai 0-cimesek, vagyis nem hivatkoznak cimre. Ez a később ismertetendő utasitások esetén is igy lesz. A formulavezérlésü számitógép ebből a szempontból rokon a cimnélküli vagy 0-cimes számitógépekkel, amelyeknek gépi utasitásai (tulnyomórészt) 0-cimesek és amelyek első képviselői a KDF 9 és a Burroughs B 5000 tipusu gépek voltak [79].

3.3. Konstansok feldolgozása; egyoperanduszu operátorok, szabványos függvények

Ha az aritmetikai kifejezésben (decimális alakban irt, egyszerüség kedvéért rögzitett vesszős (fixpontos) pozitiv) konstans is szerepel (amely esetben természetesen az aritmetikai kifejezésekben szereplő operanduszok szintaxisa:

<operandusz> ::=<konstans>|<skaláris változó>|(<operandusz> <operátor> <operandusz>)

), akkor a gép konvertére is szükség van, amely a konstansokat decimális alakról gépi (bináris) alakra alakitja át. A konverter két, az aritmetikai egység összeadó és szorzó részével kapcsolatban lévő regisztert tartalmaz, a számregisztert (SzR) és a váltószámregisztert (VSzR). A konverter nyugalmi állapotában (a gép inditásakor, továbbá minden konvertálás befejezésekor) az SzR tartalma (aritmetikai) 0, a VSzR-é (bináris alakban irt)

10. Célszerű a számjegyek, mint alapjelek kódját értékükkel egyenlőnek választani. Ezt feltételezve, a gép utasításrendszerének megfelelő további része:

Alapjel	Aktiv regiszter	Egyéb feltétel	Utasítás	Aktivizálandó regiszter
⟨számjegy⟩	Nem UR	VSzR tartalma 10	VSzR tartalma és SzR tartalma betöltődnek a szorzómübe; a szorzat és UR' tartalma betöltődnek az összeadómübe; az összeg lesz SzR uj tartalma	A régi aktiv regiszter marad aktiv
		VSzR tartalma nem 10	VSzR tartalma és 1/10 betöltődnek a szorzómübe; a szorzat lesz VSzR uj tartalma; VSzR (uj) tartalma és UR tartalma betöltődnek a szorzómübe; a szorzat és SzR tartalma betöltődnek az összeadómübe; az összeg lesz SzR uj tartalma	A régi aktiv regiszter marad aktiv
⟨tizedesvessző⟩	nem UR		1 betöltődik VSzR-be	A régi aktiv regiszter marad aktiv

Alapjel	Aktiv regiszter	Egyéb feltétel	Utasítás	Aktiválandó regiszter
⟨konstansvégjel⟩	$BOR_{\varkappa}$		SzR tartalma betöltődik $BOR_{\varkappa}$ -ba; a konverter nyugalmi állapotába kerül	$OR_{\varkappa}$
	$JOR_{\varkappa}$	VR üres	SzR tartalma betöltődik $JOR_{\varkappa}$ -ba; a konverter nyugalmi állapotába kerül	OR_{λ}, ahol λ a legnagyobb olyan $\varkappa$ -nál kisebb index, amelyre OR_{λ} üres volt; ha ilyen nincs, akkor ER_0

Világos, hogy a konverzió a tizedes vesszőig a Horner-féle elrendezés szerint (és azután is helyesen) történik és eredménye a konstans-végjel hatására a kifejezés-feldolgozó egység megfelelő (aktiv) regiszterébe kerül. (Célszerű minden konstanst külön e célra szolgáló konstans-végjellel befejezni; ilyenek hiányában minden, a számjegyektől és a tizedes-vesszőtől különböző alapjelre az utasitásrendszerben egyébként is meglevő funkcióján kivül a konstans-végjel funkcióját is kell bizni.)

Egyoperanduszu operátor előfordulása esetén olyankor kerül sor ilyen operátorhoz, mint alapjelhez tartozó utasitásra, amikor az aktiv regiszter valamelyik BOR_{ic} (ez a jele annak, hogy az operátor, pl. a − jel, egyoperanduszunak tekintendő). Ilyenkor BOR_{ic} üres marad, UR tartalma (az operátor kódja) OR_{ic}-be kerül és a JOR_{ic}megtelésekor a megfelelő egyoperanduszu müvelet kerül (a müveletvégző alegységek segitségével) végrehajtásra.

Olyan operátor esetén, amelynek megfelelő müveletet végző mü nincs beépitve a gépbe (pl. ↑ mint a hatványozás jele esetén, vagy log, exp, sin és egyéb egyváltozós szabványos függvények, mint egyoperanduszu operátorok esetén) az aritmetikai egységbe be kell épitve ("huzalozva") lennie a megfelelő szubrutinnak. E szubrutinok ugyancsak a gép nyelvén készülnek, de ugy, hogy zárójelet tartalmazó kifejezés ne forduljon elő bennük, ez elérthető ugy, hogy minden müvelet eredményét munkaváltozó értékeként tároljuk. (Ez azt a célt szolgálja, hogy e szubrutinok ne használják a kifejezés-feldolgozó egységet és igy a megfelelő operátorok ne csökkentsék a megengedett zárójelmélységet.)

Célszerű a szubrutinok munkaváltozóinak külön, a skaláris memóriától idegen tartományt, un. munkamezőt kijelölni a gép operativ memóriájában. Ha valamelyik JOR_{ic} olyankor telik meg, amikor OR_{ic} tartalma olyan operátor, amelynek megfelelő müvelethez épitett szubrutin tartozik, akkor természetesen e szubrutin felhivása történik meg, BOR_{ic} és JOR_{ic} (ill. egyoperanduszu operátor esetén csak JOR_{ic}) tartalmával, mint paraméterekkel és a szubrutin szolgáltatta eredmény a szubrutin lefutása után ER_{ic} -ba töltődik be.

3.4. Indexes változók feldolgozása

Amennyiben az aritmetikai kifejezés indexes változókat is tartalmaz (amely esetben az

⟨operandusz⟩:: =⟨konstans⟩ |⟨változó⟩|(⟨operandusz⟩
⟨kétoperanduszu operátor⟩ ⟨operandusz⟩)|
⟨egyoperanduszu operátor⟩ ⟨operandusz⟩

⟨változó⟩:: = ⟨skaláris változó⟩|⟨indexes változó⟩

⟨skaláris változó⟩:: = ⟨kisbetü⟩

⟨indexes változó⟩:: = ⟨tömbnév⟩ [⟨indexlista⟩]

⟨tömbnév⟩::=⟨nagybetü⟩

⟨indexlista⟩:: = ⟨index⟩|⟨indexlista⟩ , ⟨index⟩

⟨index⟩::= ⟨aritmetikai kifejezés⟩

szintaxis érvényes, ismét egyszerüség kedvéért csak egykarakteres azonositókat engedve meg tömbnév gyanánt, de megkülönböztetve

a skaláris változónévként és tömbnévként használható, pl. kis- és nagy betüket), akkor a gép cimkiszámitó egysége is szerephez jut. Ennek feladata, hogy az indexes változók értékeit tartalmazó rekeszek cimét kiszámitsa a tömbnévhez tartozó tömbdeklaráció és az indexeknek ugyancsak a kifejezés-feldolgozó egység által kiszámitott értéke alapján.

A tömbelemeket cimezhető tömegtárolóba, un. tömbmemóriába helyezzük el, egy-egy tömb esetén az indexek szerinti lexikografikus elrendezésben. (Célszerü néhány tömb, köztük a programvektor számára helyet foglalni az operativ tárolóban; ezeket a helyeket is a tömbmemóriához számitjuk. Olyan tömböknek, amelyekkel később számolni akarunk, lehetőleg egyéb számitással egyidejü áttárolása a tömbmemóriának az operativ tárolóban lévő részébe, amiről a programkészitőnek, vagy előrelátó memória alkalmazásával, a formulavezérlésü számitógéphez készitendő operációs rendszernek kell gondoskodnia, lényegesen meggyorsithatja az indexes változókkal történő számitást.) Egyszerüség kedvéért szoritkozzunk sztatikus tömbelem-elhelyezésre. Ebben az esetben (egyszerüség kedvéért egyetlen tömböt deklaráló tömbdeklarációra szoritkozva) az

$$\underline{\text{array}}\ \Gamma[\alpha_1:\omega_1, \alpha_2:\omega_2, \cdots, \alpha_\nu:\omega_\nu]$$

tömbdeklarációval (ahol Γ helyett a tömbnév áll; ν numerikusan adott természetes szám, a tömb dimenziószáma; $\alpha_1, \alpha_2, \cdots, \alpha_\nu$ helyett az alsó indexkorlátok; $\omega_1, \omega_2, \cdots, \omega_\nu$ helyett a felső indexkorlátok értékét meghatározó egész értékü aritmetikai kifejezések állnak), amelyet a formulavezérlésü számitógép nyelvén

$$\omega_1 \leftarrow \alpha_1 \otimes \omega_2 \leftarrow \alpha_2 \otimes \cdots \otimes \omega_\nu \leftarrow \alpha_\nu \Longrightarrow \Gamma$$

alakban célszerü irni, deklarált tömb $\Gamma[\xi_1, \xi_2, \cdots \xi_\nu]$ elemének cimét, ahol $\xi_1, \xi_2, \cdots \xi_\nu$ helyett az indexkifejezések állnak, a

$$\sigma + \sum_{\kappa=1}^{\nu} \left((\xi_\kappa - \alpha_\kappa) \prod_{\lambda=\kappa+1}^{\nu} (\omega_\lambda - \alpha_\lambda + 1) \right) = \sigma - \sum_{\kappa=1}^{\nu} \left(\alpha_\kappa \prod_{\lambda=\kappa+1}^{\nu} (\omega_\lambda - \alpha_\lambda + 1) \right) + \sum_{\kappa=1}^{\nu} \left(\xi_\kappa \prod_{\lambda=\kappa+1}^{\nu} (\omega_\lambda - \alpha_\lambda + 1) \right)$$

lineáris kifejezés adja meg ($\mu > \nu$ esetén a $\prod_{\lambda=\mu}^{\nu} \alpha_\lambda = 1$ konvenciót alkalmazva), ahol σ a tömbmemória első szabad (előzőleg elhelyezett tömbök elemei számára le nem foglalt) rekeszének cime.

A cimkiszámitó egység regiszterei a következők:

(1) A szabadrekesz-regiszter (SzRR) a tömbmemória első szabad rekesze δ cimének tárolására.

(2) A cimregiszter (CR), amelyben valamely tömbdeklaráció feldolgozása során előbb a $\sum_{\kappa=1}^{\nu} \left(\alpha_\kappa \prod_{\lambda=\kappa+1}^{\nu} (\omega_\lambda - \alpha_\lambda + 1) \right)$ "cimeltolás", majd a $\delta - \sum_{\kappa=1}^{\nu} \left(\alpha_\kappa \prod_{\lambda=\kappa+1}^{\nu} (\omega_\lambda - \alpha_\lambda + 1) \right)$ "eltolt kezdő-

cim" képződik, valamely indexes változó feldolgozása során pedig annak értékét tartalmazó tömbmemóriarekesz cime képződik.

(3) Cimlépésregiszterek (CLR_0, CLR_1, CLR_2, ...), amelyek közül valamely tömbdeklaráció feldolgozása során CLR_0-ban a tömbelemek száma $\prod_{\lambda=1}^{\kappa} (\omega_\lambda - \alpha_\lambda + 1)$, $\kappa > 0$ esetén pedig CLR_κ-ban a $\prod_{\lambda=\kappa+1}^{\kappa} (\omega_\lambda - \alpha_\lambda + 1)$ "κ-adik cimlépés" képződik.
A gép minden tényleges megvalósitása esetén természetesen csak véges számu (az eredeti tervek szerint 6, ti. CLR_0, CLR_1..., CLR_5) cimlépésregiszter van; számuk a programban előforduló tömbök maximális dimenziószámát határozza meg (ti. 1-gyel nagyobb annál). E maximális dimenziószám tullépése megszakitást és "tömbdimenzió-tulcsordulás" hibaüzenetet okoz.

(4) Alsókorlát-regiszter (AKR), mely a "cimeltolás" kiszámitásához szükséges.

A cimkiszámitó egység nyugalmi állapotában (a gép inditásakor, továbbá minden tömbdeklaráció és minden indexes változó feldolgozásának befejezése után) CLR_0 tartalma 1, CLR_1, CLR_2,... és CR tartalma 0. Minthogy a programvektor deklaráció nélkül rendelkezésre áll (mint egydimenziós vektor 1-gyel mint egyetlen alsó indexkorláttal és a gépbe, egyedi feldolgozás esetén egyetlen programként, kötegelt feldolgozás esetén a köteg programjaiként, megszakitást és "program

Alapjel	Aktiv regiszter	Egyéb feltétel	Utasitás	Aktiválandó regiszter
		talma 1, $CLR_{\kappa+1}$ tartalma 0	SzRR és CR tartalmának különbsége kerül CR-be, majd SzRR tartalmához hozzáadódik CLR_0 tartalma	CR
⟨tömbnév⟩	CR		CR, CLR_1, CLR_2,... tartalma (gépi szavakba pakolva) tárolódik az operativ memórián belül e célra kijelölt tartomány ("tömbeligazitó") azon egymásutáni rekeszeibe, amelyek közül az elsőnek cime UR tartalma; a cimkiszámitó egység nyugalmi állapotába kerül	BOR,

	BOR_K, JOR_K vagy ER_K (az utóbbi csak $K=0$ esetén lehetséges)		A tömbeligazitó azon rekeszétől kezdődő tömöritett gépi szavak, amelynek cime UR tartalma, szétpakolva betöltődnek CR-be, CLR_1-be, CLR_2-be,...; AR tartalma betöltődik VR-be	BOR_{K+1}
[	Bármelyik		Üres utasitás	A régi aktiv regiszter marad aktiv
⟨index-közi vessző⟩	ER_K		CR tartalmához hozzáadódik CLR_1 tartalmának és ER_K (szükség esetén kerekitett) tartalmának szorzata; majd CLR_2, CLR_3,... tartalma 1-gyel kisebb indexü CLR-be tolódik át, s az utolsó CLR-be 0 tárolódik; végül BOR_K, OR_K, JOR_K és ER_K kiürül	BOR_K

Alapjel	Aktiv regiszter	Egyéb feltétel	U t a s i t á s	Aktiválandó regiszter
	OR_K		CR tartalmához hozzáadódik CLR_1 tartalmának és BOR_K (szükség esetén kerekitett) tartalmának szorzata; majd CLR_2, CLR_3,... tartalma 1-gyel kisebb indexü CLR-be tolódik át, s az utolsó CLR-be 0 tárolódik; végül BOR_K kiürül	BOR_K
]	ER_K		CR tartalmához hozzáadódik CLR_1 és ER_K (szükség szerint kerekitett) tartalmának szorzata; BOR_K, OR_K, JOR_K és ER_K kiürül ;	

			VR tartalma áttöltődik AR-be; VR kiürül; majd (az uj aktiv regisztertől függően) ugyanugy folytatódik, mint ⟨skaláris változó⟩ esetén (beleértve az aktiválandó regiszter kijelölését is), azonban UR tartalma helyett CR tartalmával; végül a cimkiszámitó egység nyugalmi állapotába kerül	
	OR_K		CR tartalmához hozzáadódik CLR_1 és BOR_K (szükség esetén kerekitett) tartalmának szorzata; BOR_K kiürül; VR tartalma áttöltődik AR-be; VR kiürül; majd (az uj aktiv regisztertől függően) ugyanugy folytatódik, mint ⟨skaláris változó⟩ esetén (beleértve az aktiválandó regiszter kijelölését is), azonban UR tartalma helyett CR tartalmával; végül a cimkiszámitó egység nyugalmi állapotába kerül	

Minthogy indexes változók feldolgozása során, amikor egy-egy indexkifejezés értékének kiszámitása történik, akkor VR nem üres, hanem AR-nek az indexes változó feldolgozásának megkezdése előtti tartalmát őrzi, szükséges az utasitásrendszer következő kiegészitése is:

Alapjel	Aktiv regiszter	Egyéb feltétel	Utasitás	Aktiválandó regiszter
⟨skaláris változó⟩	JOR_K	VR tartalma BOR_λ vagy JOR_λ sorszáma	Azon memóriarekesz tartalma, amelynek cime UR tartalma, betöltődik a JOR_K -ba	OR_μ, ahol μ a legnagyobb olyan index, amelyre $\lambda<\mu<K$ áll és OR_μ üres volt; ha ilyen nincs, akkor $ER_{\lambda+1}$

⟨konstans-végjel⟩	JOR_K	VR tartalma BOR_λ vagy JOR_λ sorszáma	SzR tartalma betöltődik JOR_K-ba; a konverter nyugalmi állapotába kerül	Mint ⟨skaláris változó⟩ esetén, ha ugyan azon feltétel teljesül

Röviden elmondva, tömbdeklaráció feldolgozása ugy történik, hogy a cimkiszámitó egység (a kifejezés-feldolgozó egység és a müveletvégző alegység segitségével) kiszámitja és, gépi szavakba pakolva, a tömbeligazitó megfelelő rekeszeibe (amelyeket célszerüen ugy választottunk, hogy közülük az elsőnek a cime megegyezzék a deklarált tömbnév numerikus kódjával) tárolja a tömb elemeinek a redukált kezdőcimét és cimlépéseit (a tömb elemei cimét indexeik függvényében megadó lineáris kifejezés konstans tagját és az indexek együtthatóit). Indexes változó feldolgozása pedig ugy történik, hogy a redukált kezdőcim és a cimlépések a tömbeligazitóból, szétpakolva, betöltődnek a cimkiszámitó egységbe, az indexkifejezések értékeit a kifejezés-feldolgozó egység (a müveletvégző alegységek segitségével) kiszámitja (az indexes változót tartalmazó kifejezés feldolgozásához az indexes változó előfordulását megelőzően használt emeleteinél magasabb emeleteken), majd az indexkifejezések éttéke, valamint a redukált kezdőcim és a cimlépések alapján a cimkiszámitó egység kiszámitja az indexes változó cimét.

3.5. Cimkék és vezérlésátadó utasitások feldolgozása

A formulavezérlésü számitógép nyelvén cimkék gyanánt célszerü speciális alaku cimkezárójelbe (pl. [1]) tett, decimális alakban irt természetes számokat használni. Minden megengedett cimkének megfelel a vezérlőegység egy ugráshely-regisztere (U^{HR}_0, U^{HR}_1,...), amely, attól kezdve, hogy a kérdéses cime (mint utasitás cimkéja, nem pedig mint vezérlésátadó utasitásban megadott cimke) a program alapjeleinek feldolgozása során sorrakerült, a neki megfelelő valódi cimet (a cimkézett utasitás első alapjele numerikus kódjának, mint a programvektor komponense szeletének cimét) tárolja. (Egyszerüség kedvéért feltesszük, hogy egy utasitásnak csak egy cimkéje van. Azt természetesen fel kell tennünk, hogy egy cimkével csak egy utasitás lehet cimkézve.) A cimke értékét e helyen is, a vezérlésátadó utasitás feldolgozása során is a konverter alakitja át gépi (bináris) alakra.

A vezérlésátadó utasitásokat a formulavezérlésü számitógép nyelvén

go to ⟨természetes szám⟩

helyett, ahol ⟨természetes szám⟩ a cimke (e helyen cimkezárójel

nélkül), célszerű csak magát a címkét kiírni (mint az ALGOL 68 nyelv esetén), a végén egy, a konstans-végjeltől különböző címke-végjellel.

Abban az esetben, ha a vezérlésátadó utasítás feldolgozásakor a konverter SzR-ében gépi alakban megjelenő címkének megfelelő ugráshely-regiszter nem üres (hátra ugrás), a címkevégjelhez tartozó utasítás abban áll, hogy az SzR tartalmának megfelelő ugráshely-regiszter tartalma töltődik be az utasításszámlálóba, majd a konverter nyugalmi állapotába kerül. (Természetesen a formulavezérlésű számítógép esetén az utasításszámláló (USz) tartalma a program azon alapjele numerikus kódjának címét adja meg, amely legközelebb betöltődik az UR utasításregiszterbe. Ez a cím a programvektor azon komponensének címéből áll, amelynek egyik szelete a kérdéses alapjel numerikus kódja, kiegészítve jobboldalt a szükséges számú bittel, amelyek azt határozzák meg, hogy a komponens hányadik szelete a kérdéses alapjel numerikus kódja. Vezérlésátadást nem okozó utasítás végrehajtása előtt az USz tartalma mindig 1-gyel nő, tehát legközelebb a programvektor ugyanazon komponensének következő szelete kerül az UR-be, vagy, ha az utolsó szelete volt benn, a következő komponensének legelső (csupa 0 bittel jelzett) szelete).

Egészen más a helyzet akkor, ha az SzR tartalmaként megjelenő címkék megfelelő ugráshely-regiszter üres, vagyis a kérdéses címke még nem került sorra, mint utasítás címkéje (előre ugrás). Ebben az esetben a gépnek meg kell keresnie ezt a címkét a program-

ban. Evégett a címke-végjelhez tartozó utasítás ebben az esetben az UR utasításregisztert aktiválja, vagyis ennek sorszámát tölti be AR-be, azon (üres) ugráshely-regiszter sorszámát pedig, amely SzR tartalmának (a keresett címkének) felel meg, VR-be tölti be, majd a konvertert nyugalmi állapotába viszi. Az utasításregiszter aktív volta a gép egy speciális "kereső" üzemmódjának jele. Ebben az üzemmódban a címke-kezdőzárójel kivételével minden alapjelhez tartozó utasítás üres, tehát minden, a címke-kezdőzárójeltől különböző alapjelen "átfut" a gép, úgy, hogy UR marad az aktív regiszter (tehát tovább is kereső üzemmódban marad). A címke-kezdőzárójelhez tartozó utasítás abban áll, hogy az az ugráshely-regiszter lesz aktív, amelynek sorszáma VR tartalma (anélkül, hogy VR tartalma megváltozna). Ezzel tehát felfüggesztődik a kereső üzemmód, a címke-kezdőzárójel után következő számjegyekhez tartozó utasítások (lásd fent az utasítás rendszernek a ⟨számjegy⟩-hez tartozó részét) végrehajtódnak, tehát a címke-zárójelben lévő címke gépi alakra konvertálva az SzR-ben jelenik meg. A címke-végzárójelhez tartozó utasítás, azonkívül, hogy mint mindig, betölti az USz aktuális tartalmát az SzR tartalmának megfelelő ugráshely-regiszterbe és a konvertert nyugalmi állapotába viszi, abban az esetben, ha valamelyik ugráshely-regiszter aktív, a konverternek nyugalmi állapotba vitele előtt megvizsgálja, hogy az aktív re-

giszter megegyezik-e az SzR tartalmának megfelelő ugráshely-regiszterrel (vagyis megtaláltuk-e a keresett cimkét). Ha igen, akkor VR-et kiüriti, BOR_0-t aktiválja (tehát a kereső üzemmód megszünik és az eddig keresett, most megtalált cimkével cimkézett utasitás végrehajtása kezdődik meg); ha nem, akkor ismét UR-et aktiválja (vagyis folytatja a kereső üzemmódot) és csak azután viszi (mindkét esetben) a konvertert nyugalmi állapotába.

Eszerint a cimke megkeresése során minden, időközben talált cimkének megfelelő valódi cim betöltődik a kérdéses cimkének megfelelő ugráshely-regiszterbe. Ez azt a célt szolgálja, hogy ha a program során később ilyen cimkére vezérlésátadó (visszaugró) utasitás kerül sorra, a megfelelő valódi cim már benne legyen a cimkének megfelelő ugráshely-regiszterben, tehát valóban csak előre ugrás esetén kelljen a cimkét megkeresni.

Ha a keresett cimkét a gép nem találja meg, mielőtt a "program vége" jelhez ér, akkor az ez utóbbi jel okozta megszakitást "hiányzó cimke" hibaüzenet kiséri. (VR tartalmából, az ugráshely-regiszterek sorszámának alkalmas választása esetén, a hiányzó cimke is könnyen rekonstruálható és a hibaüzenettel kinyomtatható.)

Amennyiben nem kereső üzemmódban kerül sor a cimkézett utasitásra a programban (amikor tehát nem UR, hanem, mint minden utasitás feldolgozása után, BOR_0 aktiv), akkor a cimke-kezdőzárójelhez tartozó utasitás üres (és BOR_0 marad az aktiv regiszter); a cimke-

végzárójelhez tartozó utasitás pedig, mint mondtuk, betölti az USz tartalmát az SzR tartalmának megfelelő ugráshely-regiszterbe, majd a konvertert nyugalmi állapotába viszi.

A cimkékkel és a vezérlésátadó utasitások feldolgozásával kapcsolatos gépi utasitások tehát, a fentiekhez hasonló táblázatban összefoglalva, a következők:

Alapjel	Aktiv regiszter	Egyéb feltétel	U t a s i t á s	Aktiválandó regiszter
Bármelyik cimke-kezdőzárójel kivételével	UR		Üres	UR marad akiv
⟨Cimke-kezdő-zárójel⟩	UR		Üres	Az a regiszter, amelynek sorszáma VR tartalma

	Nem UR		Üres	A régi aktiv regiszter marad aktiv
⟨Címke-végzárójel⟩	UHR_κ	SzR tartalma κ	USz tartalma betöltődik UHR_κ-ba; VR kiürül; a konverter nyugalmi állapotába kerül	BOR_0
		SzR tartalma $\lambda \neq \kappa$	USz tartalma betöltődik UHR_λ-ba; a konverter nyugalmi állapotába kerül	UR
	BOR_0	SzR tartalma λ	USz tartalma betöltődik UHR_λ-ba; a konverter nyugalmi állapotába kerül	BOR_0 marad aktiv
⟨címke-végjel⟩	Nem UR	UHR_κ, ahol κ az SzR tartalma, nem üres	UHR_κ tartalma betöltődik USz-be; a konverter nyugalmi állapotába kerül	BOR_0

Alapjel	Aktiv regiszter	Egyéb feltétel	U t a s í t á s	Aktiválandó regiszter
		UHR_κ, ahol κ az SzR tartalma, üres	UHR_κ sorszáma betöltődik VR-be; a konverter nyugalmi állapotába kerül	UR

3.6. Logikai kifejezések feldolgozása

A formulavezérlésü számitógépben az "igaz" logikai érték gépi ábrázolása bármely pozitiv szám (többek között 1 is) lehet, a "hamis" logikai értéké pedig bármely negativ szám lehet (többek között 0 is, amelynek, mint mondtuk, előjel-bitje a negativ előjelnek megfelelő 1). A reláció jelek ($<, \geq, >, \leq, =, \neq$) és a logikai müveleti jelek ($\neg, \wedge, \vee, \rightarrow, \leftarrow, \leftrightarrow, \overline{\wedge}, \overline{\vee}, \not\rightarrow, \not\leftarrow, \not\leftrightarrow$) operátornak ($\neg$ egyoperanduszu operátornak, a többi kétoperanduszu operátornak) tekintendő. Az aritmetikai egység müveletvégző részének megfelelő áramkörei a reláció-jelek esetén olyan aritmetikai müveletet végeznek, amelynek eredménye valamely pozitiv (0 előjelbittel ellátott, de nem csupa 0 bitből álló) szám, ha a reláció teljesül, ellenkező esetben valamely negativ (1 előjelbittel ellátott) szám. Logikai müveleti jelek esetén pedig olyan müveletet végeznek, amelynek eredménye tetszőleges pozitiv számokon végrehajtva pozitiv, ha a logikai müvelet eredménye az "igaz" logikai értéken végrehajtva "igaz", ellenkező esetben negativ; tetszőleges negativ számokon végrehajtva pozitiv, ha a logikai müvelet eredménye, a "hamis" logikai értékeken végrehajva "igaz", ellenkező esetben negativ; egy tetszőleges pozitiv és egy tetszőleges negativ operanduszon végrehajtva pozitiv, ha a logikai müvelet eredménye az "igaz" és a "hamis" logikai értékeken végrehajtva "igaz", ellenkező esetben negativ; végül egy tetszőleges negativ és egy tet-

kező esetben negatív, végül egy tetszőleges negatív és egy tetszőleges pozitív operanduszon végrehajtva pozitív, ha a logikai művelet eredménye a "hamis" és az "igaz" logikai értékeken végrehajtva "igaz", ellenkező esetben negatív.

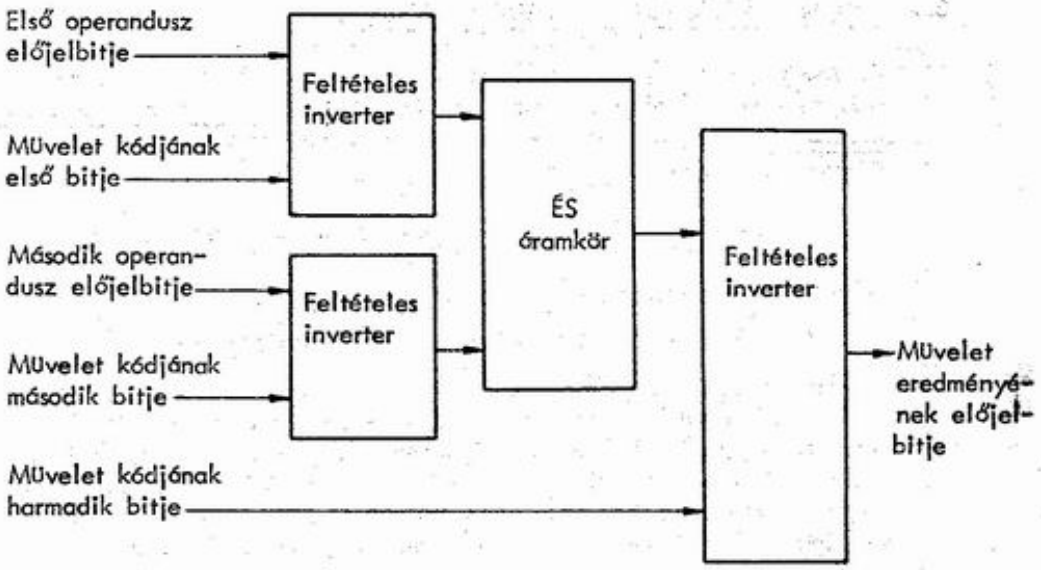
Így pl. a " $>$ " reláció-jel egyszerűen a " $-$ " (kétoperanduszu) műveleti jellel ekvivalensnek tekinthető, ugyanis $a - b$ akkor és csak akkor pozitív, ha $a > b$ áll. Hasonlóan, az $a<b$ reláció logikai értéke helyett $b - a$ értéke számítható ki. (Természetesen a kivonást is az összeadóműben végzi, úgy, hogy a kivonandó — ha nem 0 — megváltoztatott előjelbittel megy bele; $a < b$ logikai értékének kiszámításához tehát a megy megváltoztatott előjelbittel, b pedig változatlanul az összeadóműbe.) Az $a \geqq b$ és $a \leqq b$ relációk logikai értékének kiszámításához is $a - b$ ill. $b - a$ értékét számítjuk ki, azonban, ha ez az érték 0-nak adódik, előjelbitjét a + előjelnek megfelelő 0-ra, de ugyanakkor (hogy ne "üres" eredményt kapjunk) egy másik bitjét 1-re változtatja egy külön erre szolgáló áramkör. Az $a = b$ reláció logikai értékének kiszámításához is $a - b$-t számítjuk ki, de egy külön áramkör abban az esetben, ha 0 az eredmény, előjelbitjét 0-ra és egy másik bitjét 1-re, ellenkező esetben előjelbitjét 1-re változtatja. Az $a \neq b$ reláció logikai értékének kiszámításához, $a - b$ értékének kiszámítása után, ha 0 az eredmény, nem változtatunk rajta, ellenkező esetben előjelbitjét 0-ra változtatja egy külön áramkör.

Az $\wedge, \vee, \rightarrow, \leftarrow, \overline{\wedge}, \overline{\vee}, \not\rightarrow, \not\leftarrow$ kéttagu logikai műveletek elvégzésére egyszerű áramkörök szolgálnak, amelyekbe az operanduszoknak csak az előjelbitje megy be és az eredménynek is csak az előjelbitje jön ki, többi bitje pl. 0, egyiket, pl. az utolsót kivéve, amely 1. Ezek az áramkörök felépithetők egy ÉS-áramkörből és legfeljebb három inverterből. Ha az említett műveleti jeleket célszerűen kódoljuk, pl. úgy, hogy kódjuknak három (pl. szomszédos) bitje a következő táblázatból kiolvasható három bittel egyezik meg (a többi tetszőleges és e műveleti jeleknek más alapjelektől való megkülönböztetésére szolgálhat):

Műveleti jel	Kód három bitje
$\wedge$	000
$\vee$	111
$\rightarrow$	011
$\leftarrow$	101
$\overline{\wedge}$	001
$\overline{\vee}$	110
$\not\rightarrow$	010
$\not\leftarrow$	100

és a kód e bitjeit rövidség kedvéért első, második, ill. harmadik bit-

nek nevezzük, akkor e nyolc művelet egyetlen egy ÉS-áramkörből és három feltételes inverterből összetett áramkörrel elvégezhető. (Feltételes inverteren olyan kétbemenetű áramkört értünk, amely az egyik un. főbemenetére adott 0 vagy 1) jelet invertálja, ha a másik, un. vezérlő bemenetére adott jel 1, ellenkező esetben változatlanul hagyja. A feltételes inverter tulajdonképpen nem más, mint egy antivalencia-áramkör, amiből az is következik, hogy a két bemenete egyenértékű.) Ez az áramkör a 3. táblán látható.



3. ábra

Az ↚↛ (antivalencia) műveleti jelhez tartozó áramkör, mint említettük, éppen a feltételes inverter, annak mindkét bemenetét operandusz-bemenetnek használva. A ↚↛ műveleti jelhez tartozó áramkör ebből egy

(feltétlen) inverter utánkapcsolásával jöhet létre. A két áramkör megint egyesithető, amely esetben ez utóbbi inverter feltételes inverterrel pótolható, amelynek vezérlő bemenetére a müveleti jelek kódjának egy olyan bitje megy, amely az ↔ jel esetén 0, az ↔ jel esetén 1.

A 1 müveleti jelhez tartozó áramkör egy (feltétlen) inverter.

Az aritmetikai egység müveletvégző részét ilyen áramkörökkel kiegészitve, a kifejezés-feldolgozó egység alkalmassá válik logikai kifejezések feldolgozására is.

3.7. Feltételes kifejezések és utasitások feldolgozása

Az eredeti elgondolás szerint a formulavezérlésü számitógép nyelvének szintaxisa a feltételes utasitások közül csak a feltételes vezérlésátadó utasitásokat engedte meg, vagyis az olyanokat, amelyek alakja az ALGOL 60 nyelven

if ⟨logikai kifejezés⟩ then go to ⟨cimke 1⟩ else go to ⟨cimke 2⟩

volna; a formulavezérlésü számitógép nyelvén ezt

⟨logikai kifejezés⟩;⟨cimke 1⟩ ⟨cimkevégjel⟩;⟨cimke 2⟩ ⟨cimkevégjel⟩

alakban irjuk.

A feltételes vezérlésátadó utasitás elején álló (ugrás-feltételként szereplő) logikai kifejezés logikai értékét a fentiek szerint a kifejezés-feldolgozó egység számitja ki, éspedig e logikai kifejezés feldolgozása végén az ER_0 regiszter tartalmának előjelbitje reprezentálja (0 előjelbit az "igaz", 1 a "hamis" logikai értéknek felel meg). A cimkevégjelhez tartozó utasitás -- a feltétlen vezérlésátadó utasitásnál mondottaktól eltérően -- megvizsgálja ezt az előjelbitet, és csak akkor hajtja végre az ott mondottak szerint a vezérlésátadást (az SzR tartalmának megfelelő ugráshely-regiszter tartalmának betöltését SzR-be, vagy ha ez az ugráshely-regiszter üres, a "kereső üzemmódba" való áttérést), ha ez az előjelbit 0, ellenkező esetben szekvenciálisan továbbhalad a következő utasitásra (vagyis USz tartalmát 1-gyel növeli, BOR_0-t aktiválja és a

konvertert nyugalmi állapotába viszi). Eszerint a formulavezérlésü számitógép utasitásrendszerének a cimkevégjelhez, mint alapjelhez tartozó része a következőképpen módosul és egészül ki a pontosvesszőhöz (folytatás jeléhez) tartozó utasitással:

Alapjel	Aktiv regiszter	Egyéb feltétel	Utasitás	Aktiválandó regiszter
;	Bármelyik		Üres utasitás	A régi aktiv regiszter marad aktiv
⟨Cimke-végjel⟩	Nem UR	ER_0 előjelbitje 0; UHR_K, ahol K az SzR tartalma, nem üres	UHR_K tartalma betöltődik USz-be; a konverter nyugalmi állapotába kerül; BOR_0, Or_0, JOR_0, ER_0 kiürül	BOR_0

Alapjel	Aktiv regiszter	Egyéb feltétel	Utasitás	Aktiválandó regiszter
		ER_0 előjelbitje 0; UHR_K, ahol K az SzR tartalma, üres	UHR_K sorszáma betöltöltődik VR-be; a konverter nyugalmi állapotába kerül; BOR_0, OR_0, JOR_0, ER_0 kiürül	UR
		ER_0 előjelbitje 1	A konverter nyugalmi állapotába kerül; BOR_0, OR_0, JOR_0, ER_0 kiürül	BOR_0

A fentiek szerinti

⟨cimke⟩ ⟨cimkevégjel⟩

feltétlen vezérlésátadó utasitás is helyesen hajtódik végre; ti., ha a cimke előtt nem áll ugrás-feltétel, akkor (mint minden utasitás végrehajtása után, és a gép megindításakor is) ER_0 üres, vagyis tartalmának minden bitje 0,

így az előjelbitje is. Voltaképpen a fent említett

⟨logikai kifejezés⟩ ; ⟨cimke 1⟩ ⟨cimkevégjel⟩ ; ⟨cimke 2⟩ ⟨cimkevégjel⟩

feltételes vezérlésátadó utasitás két egymásután következő utasitásból áll:

⟨logikai kifejezés⟩ ; ⟨cimke 1⟩ ⟨cimkevégjel⟩

a ⟨cimke 1⟩ -hez ugrat, ha a ⟨logikai kifejezés⟩ logikai értéke "igaz", ellenkező esetben szekvenciálisan továbbhalad a

⟨cimke 2⟩ ⟨cimkevégjel⟩

feltétlen vezérlésátadó utasitáshoz, amely a ⟨cimke 2⟩ -höz ugrat.

A Ljapunov-féle operátor-nyelvnek megfelelően be lehet vezetni egy másik cimkevégjelet is, amelyhez tartozó utasitás akkor létesit vezérlésátadást, ha az ER_0 regiszter tartalmának előjelbitje 1. Az egyetlen cimkét és ilyen másik cimkevégjelet tartalmazó feltételes vezérlésátadó utasitás akkor ugrat a benne lévő cimkére, ha az elején álló logikai kifejezés logikai értéke "hamis", ellenkező esetben szekvenciálisan továbbhalad a következő utasitásra.

A pontosvessző természetesen használható utasitásoknak, vagy deklarációknak (vagy deklarációnak utasitástól való) elválasztására, mint az ALGOL-ban, azzal az (előnyös) eltéréssel, hogy sem elhagyása nem hiba, sem olyan helyre való kitétele, ahol a nyelv szintaxisa szerint nem kellene kitenni.

Mint ismeretes, az általános feltételes utasitás pótolható feltételes vezérlésátadó utasitással és nem feltételes utasitásokkal: az ALGOL 60-ban

if ⟨logikai kifejezés⟩ then ⟨feltétlen utasitás⟩

alakban irt feltételes utasitás a következővel:

⟨logikai kifejezés⟩ ; ⟨cimke⟩ ⟨cimkevégjel⟩ ; [⟨cimke⟩]
⟨feltétlen utasitás⟩

, ahol ⟨cimke⟩ a programban egyébként elő nem forduló cimkét jelent; az

if ⟨logikai kifejezés⟩ then ⟨feltétlen utasitás⟩ else ⟨utasitás⟩

pedig a következővel:

⟨logikai kifejezés⟩ ; ⟨cimke 1⟩ ⟨cimkevégjel⟩ ; ⟨cimke 2⟩ ⟨cimkevégjel⟩ ;
[⟨cimke 1⟩] ⟨feltétlen utasitás⟩ ; ⟨cimke 3⟩ ⟨cimkevégjel⟩ ;
[⟨cimke 2⟩] ⟨utasitás⟩ ; [⟨cimke 3⟩]

, ahol ⟨cimke 1⟩ , ⟨cimke 2⟩ , ⟨cimke 3⟩ a programban egyébként elő nem forduló cimkéket jelentenek.

Annak sincs azonban akadálya, hogy a formulavezérlésü számitógép nyelvében megengedjünk ilyen (ALGOL-szerü) általános feltételes utasitásokat. Célszerü ilyenkor az ALGOL 68 módján a feltételes utasitásokat az if "feltételes-utasitás-kezdőzárójel"-nek megfelelő, az ALGOL 68-

ban fi-nek irt "feltételes-utasitás-végzárójel"-lel befejezni. Ebben az esetben a feltételes utasitást kezdő if alapjel "párjaként" szereplő then alapjelhez tartozó utasitás megvizsgálja az ER_0 tartalmának előjel-bitjét és a gép attól függően, hogy az 0-e vagy 1, e then alapjel és a "párjaként" szereplő else alapjel közötti utasitás végrehajtására tér át, ennek befejezése után pedig átugrik a feltételes utasitást kezdő if alapjel "párjaként" szereplő fi alapjel utáni utasitásra, ill. e then alapjel "párjaként" szereplő else alapjel és a feltételes utasitást kezdő if alapjel "párjaként" szereplő fi alapjel közötti utasitás végrehajtására tér át, majd szekvenciálisan halad tovább. (Ha else nélküli feltételes utasitásról van szó, azaz a feltételes utasitást kezdő if alapjel "párjaként" szereplő then alapjeltől kezdve a feltételes utasitást kezdő if alapjel "párjaként" szereplő fi alapjelet megtalálja a gép, mielőtt e then alapjel "párjaként" szereplő else alapjelet megtalálná, akkor a gép attól függően, hogy az ER_0 tartalmának előjelbitje 0-e vagy 1, e then alapjel és a feltételes utasitást kezdő if alapjel "párjaként" szereplő fi alapjel közötti utasitást hajtja végre, majd szekvenciálisan halad tovább, ill. ezt az utasitást kihagyja és a feltételes utasitást kezdő if alapjel "párjaként" szereplő fi alapjel utáni utasitásra ugrik át. Azt, hogy valamely "baloldali" alapjelnek

melyik "jobboldali" a 'párja' (az if-nek melyik then és melyik fi, a then-nek melyik else), a már említettől annyiban eltérő "kereső üzemmódban" állapitja meg a gép, hogy egy oda-vissza számláló regiszter kezdeti tartalmaként 1-et állit be, e regiszter tartalmát a "baloldali" alapjel minden további előfordulásakor 1-gyel növeli, a "jobboldali" alapjel minden előfordulásakor 1-gyel csökkenti, minden más alapjelen "átfut" a gép, egészen addig, mig 0 lesz e számláló regiszter tartalma, amikoris megszünik ez a fajta kereső üzemmód. (Összesen három ilyen oda-vissza számláló regiszterre van szükség.)

Hasonlóan valósitható meg feltételes kifejezések feldolgozása is a formulavezérlésü számitógépen.

3.8. Ciklus-utasítások feldolgozása

A formulavezérlésü számitógép nyelvén célszerü a ciklus-utasitásnak nemcsak a kezdetét jelezni egy alapjellel (az ALGOL for alapjelének megfelelő cikluskezdet-jellel), hanem a végét is egy másik alapjellel (ciklusvég-jel; a Ljapunov-féle operátor-nyelv e célra kapcsos kezdő-, ill. végzárójelet használ).

Szoritkozzunk egyszerü (egy ciklusista-elemü) ciklus-utasitásokra, amelyeknek két fajtáját engedjük meg a következő szintaxis szerint:

⟨ciklus utasitás⟩ : := ⟨cikluskezdet-jel⟩ ⟨ciklus-paraméter⟩ ;
⟨kezdőérték-kifejezés⟩ step ⟨lépés-kifejezés⟩ until ⟨végérték-kifejezés⟩ do ⟨ciklusmag⟩ ⟨ciklusvég-jel⟩ | ⟨cikluskezdet-jel⟩ ⟨ciklus-paraméter⟩ ; ⟨kezdőérték-kifejezés⟩ step ⟨lépés-kifejezés⟩ until do ⟨ciklusmag⟩ ; ⟨ismétlés-feltétel⟩ ⟨ciklusvég-jel⟩

, ahol (az ALGOL 60-tól eltérően) az ismétlés-feltételes ciklus-utasitásnál is megengedünk lépés-kifejezést és az ismétlés-feltételt ilyen ciklus-utasitásokban a ciklusmag utján adjuk meg. Itt a ⟨ciklus-paraméter⟩ (egyszerüség kedvéért) skaláris (index nélküli) változó; a ⟨kezdőérték-kifejezés⟩, ⟨lépés-kifejezés⟩ és a ⟨végérték-kifejezés⟩ aritmetikai kifejezések; az ⟨ismétlés-feltétel⟩ logikai kifejezés; a ⟨ciklusmag⟩ pedig egy vagy több utasitás, ha több, akkor pontosvesszővel elválasztva.

A ciklus-utasitások feldolgozásához a vezérlő egységen belül a következő regiszterek szükségesek:

(1) Egy ciklusszámláló (CSz), amely egy oda-vissza számláló regiszter, amelynek tartalma kezdetben (a gép inditásakor, továbbá különálló ciklus-utasitásnak, vagy ciklus-utasitások egymásba skatulyázott rendszerének feldolgozása után) 0, minden ciklukezdet-jelhez, mint alapjelhez tartozó utasitás hatására 1-gyel nő, minden ciklusvég-jelhez, mint alapjelhez tartozó utasitás hatására pedig, amennyiben az ezen utasitás által végzett vizsgálat arra nézve, hogy kell-e ismét végrehajtani a ciklusmagot, arra az eredményre vezet, hogy nem kell, 1-gyel csökken. A CSz eszerint a mindenkori ciklus-mélységet mutatja; hosszát a maximális megengedett ciklus-mélységnek megfelelően kell megállapitani.

(2) Annyi paraméter-regiszter (PR_1, PR_2,...), amennyi a maximális megengedett ciklus-mélység. A cikluskezdet-jelhez, mint alapjelhez tartozó utasitás, miután a CSz tartalmát 1-gyel növelte, a CSz megnövelt tartalmához mint indexhez tartozó PR-et

aktiválja. Ha valamelyik PR_λ aktiv, akkor skaláris változóhoz, mint alapjelhez tartozó utasitás hatására az utasitás-regiszter tartalma (tehát a skaláris változó numerikus kódja, nem pedig az értéke, mint abban az esetben, ha valamelyik BOR_λ vagy JOR_λ aktiv) töltődik be az aktiv regiszterbe. Tehát PR_λ a λ mélységü ciklus-utasitás paraméterének numerikus kódját őrzi (mig e ciklus-utasitás végrehajtásra nem kerül).

(3) Annyi paraméterlépés-regiszter (PLR_1, PLR_2,...), amennyi a maximális megengedett ciklus-mélység. Ezek a megfelelő paraméter-lépéseknek, vagyis lépés-kifejezések értékének őrzésére szolgálnak (miután a gép kiszámitja, a kifejezés feldolgozó egység segitségével, a megfelelő lépés-kifejezés értékét, az until alapjelhez tartozó utasitás tölti be ezt az értéket a megfelelő PLR_κ -ba; ezért van szükség erre az alapjelre "üres végérték" esetén is).

(4) Annyi végérték-regiszter ($VÉR_1$, $VÉR_2$,...), amennyi a maximális megengedett ciklus-mélység. Ezek a megfelelő végértékeknek, vagyis végérték-kifejezések értékének őrzésére szolgálnak (miután a gép kiszámitja, a kifejezés-feldolgozó egység segitségével, a megfelelő végérték-kifejezés értékét, azt a do alapjelhez tartozó utasitás tölti be a megfelelő $VÉR_\kappa$ -ba).

(5) Annyi cikluskezdet-regiszter (CKR_1, CKR_2,...), amennyi a maximális megengedett ciklus-mélység. Ezek a megfelelő ciklusmag első alapjelének cimét őrzik (azt szintén a do alapjelhez tartozó utasitás tölti be a megfelelő CKR_κ -ba).

Kezdőérték-regiszterekre nincs szükség, a kezdőértékeket, vagyis a kezdőérték-kifejezések értékét, a kifejezés-feldolgozó egység segitségével történő kiszámitásuk után, a step alapjelhez tartozó utasitás adja értékül a megfelelő ciklusparaméternek, vagyis tárolja a skaláris memória azon rekeszében, amelynek cime a megfelelő PR_K tartalma.

A ciklusutasitások feldolgozásához szükséges utasitásokat a következő táblázatban foglaljuk össze, amelyben K mindig a CSz tartalmát jelenti:

Alapjel	Aktiv regiszter	Egyéb feltétel	Utasitás	Aktiválandó regiszter
⟨ciklus-kezdet--jel⟩	BOR_0		CSz tartalma 1-gyel nő (ha nem lehet már növelni, megszakitás következik be "ciklusmélység-tulcsordulás" hibaüzenettel)	PR_K, ahol K a CSz tartalma a növekedés után
⟨skaláris változó⟩	PR_K		UR tartalma betöltődik PR_K -ba	BOR_0
step	ER_0		ER_0 tartalma tárolódik azon memóriarekeszbe, amelynek cime PR_K tartalma, majd BOR_0, OR_0, JOR_0, ER_0 kiürül	BOR_0
	OR_0		BOR_0 tartalma tárolódik azon memóriarekeszbe, amelynek cime PR_K tartalma, majd BOR_0 kiürül	BOR_0

until	ER_0		ER_0 tartalma betöltődik, PLR_K-ba, majd BOR_0, OR_0, JOR_0, ER_0 kiürül	BOR_0
	OR_0		BOR_0 tartalma betöltődik PLR_K-ba, majd BOR_0 kiürül	BOR_0

Alapjel	Aktiv regiszter	Egyéb feltétel	U t a s i t á s	Aktiválandó regiszter
do	ER_0		ER_0 tartalma betöltődik $VÉR_K$-ba, USz tartalma betöltődik CKR_K-ba, majd BOR_0 OR_0, JOR_0, ER_0 kiürül	BOR_0
	OR_0		BOR_0 tartalma betöltődik $VÉR_K$-ba, USz tartalma betöltődik CKR_K-ba, majd BOR_0 kiürül	BOR_0
	BOR_0		Üres utasitás	BOR_0 marad aktiv
⟨ciklus-vég-jel⟩	BOR_0	Azon memóriarekesz tartalmának, amelynek címe PR_K tartalma, és PLR_K tartalmának összege nem nagyobb	Azon memóriarekesz tartalmához, amelynek címe PR_K tartalma, hozzáadódik PLR_K tartalma ; CKR_K tartalma betöltődik USz-be	BOR_0 marad aktiv

		$VÉR_K$ tartalmánál		
		Azon memóriarekesz tartalmának, amelynek címe PR_K tartalma, és PLR_K tartalmának összege nagyobb $VÉR_K$ tartalmánál	CSz tartalma 1-gyel csökken (ha 0 volt, megszakitás következik be "téves ciklusvégjel" hibaüzenettel)	BOR_0 marad aktiv
	ER_0	ER_0 tartalmának előjelbitje 0	Azon memóriarekesz tartalmához, amelynek címe PR_K tartalma, hozzáadódik PLR_K tartalma; CKR_K tartalma betöltődik az USz-be; BOR_0, OR_0, JOR_0, ER_0 kiürül	BOR_0

Alapjel	Aktiv regiszter	Egyéb feltétel	U t a s i t á s	Aktiválandó regiszter
		ER_0 tartalmának előjelbitje 1	CSz tartalma 1-gyel csökken (esetleges megszakitást lásd fent); BOR_0, OR_0, JOR_0, ER_0 kiürül	BOR_0

3.9. Befejező megjegyzések

Amennyiben most megépül egy, a fenti elgondolások szerinti formulavezérlésü számitógép, azt gazdasági és megbizhatósági szempontok miatt mindenképpen korszerü technológiával kell megépiteni (mikroprogramozás és LSI áramkörök), és operációs rendszert kell hozzá készitenì. Ki kell használni a gép programozhatóságát, vagyis azt, hogy a gép nyelve csak azt tartalmazza, amit a gép a nem-formulavezérlésü számitógépek gépikód-szintjének megfelelő szinten tud végezni (bár ez lényegesen magasabb, mint a nem-formulavezérlésü számitógépek gépikód-szintje). Megfelelő software-rel ellátva azonban ennél sokkal bonyolultabb feladatok elvégzésére is képes a gép. Ennek megkönnyitésére célszerü egyes regisztereket cimezhetővé tenni; pl. az SzRR cimezhetősége lehetővé teszi a dinamikus tömbelhelyezésnek, a PLR_K-k és a $VÉR_K$-k cimezhetősége lehetővé teszi a ciklus-paraméter lépése és végértéke változtathatóságának megvalósitását software segitségével. (A cimezhetőség azzal jár, hogy a kérdéses regiszterhez extra-karaktert rendelünk, mely a programban a tartalmat jelöli. Ugyancsak extra-karakter jeléli már az eredeti elgondolás szerint is, a programvektort, ez ahhoz szükséges, hogy a program a programvektor komponenseinek az eredetitől eltérő értéket adva önmagát módosithassa.)

Az operációs rendszer feladatainak a leirásához más jellegü utasitások szükségesek, mint a felhasználói feladatok leirásához. Ezért célszerünek látszik, hogy egy formulavezérlésü számitógépben a rendszervezérlést külön processzor lássa el, amely e célnak megfelelő utasitáskészlettel rendelkezik.

Ahol az előzőkben azt mondtuk, hogy "egyszerüség kedvéért" valamely speciális esetre szoritkozunk, meg kell gondolni, hogy

érdemes-e az általános esetet a regiszterek számának lényeges növelése árán megvalósitani, vagy pedig annak figyelembevételét megfelelő software utján célszerübb-e lehetővé tenni. (A regiszter-készlet hardware realizálásának költségeit, mint már emlitettük nagymértékben csökkentheti monolitikus integrált áramkörös gyorstárolók alkalmazása.) Ugyanez vonatkozik a blokkstruktura, a kapcsolók és az ALGOL-szerü eljárások megvalósitására is.

Minthogy a jövő várhatólag a sok programozási célnyelv implementálását lehetővé tevő számitógépeké (lásd e tanulmány 2. és 4. fejezetét), ezért a formulavezérlésü számitógép nyelvét célszerü ugy megválasztani, hogy programozási célnyelvek kompilátorának irása ne okozzon nehezebb problémákat, mint assemblerek irása a nem-formulavezérlésü gépekhez. (A formulavezérlésü számitógép egyik fő előnye, hogy nyelvéhez viszonyitva magasabb szintü programozási nyelvek assembly szintü nyelveknek tekinthetők.)

E végett nem célszerü a formulavezérlésü számitógép nyelvét valamely magasabb szintü programozási nyelvhez (pl. az ALGOL 60-hoz) tulságosan közelinek választani, mert ez azon előny mellett, hogy e programozási nyelv implementálása könnyü (ill., ha a formulavezérlésü számitógép nyelve épp a kérdéses magasabb szintü programozási nyelv, akkor szükségtelen), azzal a hátránnyal járhat, hogy más, ugyancsak magasabb szintü programozási nyelveket nehezebb lesz implementálni a formulavezérlésü számitógépen. Igy pl. nem célszerü az ALGOL 60 blokkstrukturáját hardware uton feldolgozhatóvá tenni a formulavezérlésü számitógépen, mert ez amellett, hogy nagyon sok regisztert igényelne, nem jelentene előnyt pl. a FORTRAN implementálása szempontjából.

Viszont, éppen a sok speciális nyelv implementálása szempontjából célszerü lehet a formulavezérlésü számitógép nyelvét (formulaszerüen irt) string-feldolgozó utasitásokkal kibőviteni.

Az is lehetségesnek látszik, hogy a számitógépek további fejlődése majd a fenti elgondolások szerinti formulavezérlésü számitógép egyes elemeinek a nem-formulavezérlésü számitógépekbe való beépitéséhez vezet. A veremmemóriát Bauer és Samelson egy másféle elgondolások alapján tervezett formulavezérlésü számitógép legfontosabb részeként kivánták szabadalmaztatni. Később a veremmemória software-szimulációja jutott fontos szerephez a szekvenciális kompilátorokban. Ma viszont beépitett hardware-veremmemóriát is tartalmazó számitógépek épülnek. Lehet, hogy hasonlóan célszerü lesz a kifejezés-feldolgozó egység beépitése nem-formulavezérlésü számitógépekbe, annak szerkezete ugyanis közelebb áll a szokásos zárójeles jelölés struktúrájához, mint a veremmemória, amelynek strukturája inkább a zárójel nélküli (Łukasiewicz-féle vagy lengyel, ill. a forditott lengyel) jelölésmódot szimulálja.

4. KÖVETKEZTETÉSEK

4.1. Általános következtetések

A számitógépek fejlődéstörténetének vázlatos áttekintése néhány általános következtetés levonását teszi lehetővé.

Első helyen áll ezek között az, hogy a számitógépekkel kapcsolatos gyors és szerteágazó fejlődés mozgatórugói a különböző szinteken érvényesülő ellentmondások, ill. a hatásukra kialakuló és a feloldásukra irányuló törekvések. Ilyen ellentmondások fellépése előtt, a társadalmi igények jelentkezését megelőzve, felmerülhettek ugyan kivételesen olyan előremutató ötletek, amelyek jelentősége csak később, a megfelelő ellentmondás, ill. társadalmi igény jelentkezésekor mutatkozott meg, de ezek megvalósitása, amit legtöbbször a technológiai feltételek még nem érett volta is hátráltatott, elszigetelt jelenség maradt és semmiképpen sem gyakorolt befolyást a számitógépek általános fejlődési tendenciáira. Amikor viszont a XX.szd. első felének vége táján a korábbiakhoz képest uj minőségi szinten jelentkező feladatok (részben a nagy volumenü számitást igénylő, vagy rövid időhöz kötött tudományos-müszaki és katonai problémák, részben a termelési, irányitási, szervezési információs bázis megnövekedésével előálló uj tipusu feladatok) ellentmondásba kerültek a megoldásukhoz objektive rendelkezésre álló idővel, kialakult a társadalmi igény kielégitésére lehetőséget teremtő számitógépes irányzat. A 40-es évek második felében felépült univerzális, digitális számitógépekkel uj minőség jelent meg a számitó- és infor-

macioreldolgozo eszkozok fejlodeseben. Ezt követően már nemcsak az előbb említett társadalmi igények, hanem a számítógépek fejlődésén belül jelentkező ellentmondások is alapvető szerepet játszanak a további haladásban.

Mint előzőleg már említettük a számítógépek fejlődésének korai szakaszában egy-egy gép fejlettségének megitélésében a hardware-jellemzők álltak az első helyen. Hamarosan kitünt azonban, hogy a számitógép univerzális jellegében rejlő lehetőségek hatékonyabb kihasználása érdekében -- párhuzamosan azzal, hogy növelik müködési sebességét -- alapvetően meg kell javitani tényleges felhasználásának lehetőségeit (könnyebbé és az ember számára megszokottabb eszközökkel hozzáférhetőbbé kell tenni a gépet). Ez fokozatosan elvezetett a software-eszközök kifejlesztésének igényéhez. Miután fejlődésnek indul és sulya gyors ütemben megnövekszik a számitógépes munkában, kiderül, hogy a software a legmozgékonyabb elem a fejlődési folyamatban, mivel szakadatlanul uj igényeket

támaszt mind a müszaki megoldások, mind a feldolgozási módok fejlesztése terén. Ez természetesen nem zárja ki azt, hogy esetenként az ujabb hardware-fejlesztések (mint pl. uj elem-bázis létrejötte) idéznek elő lényeges haladást.

A hardware- és software-fejlődésben egyre ujabb ellentmondások lépnek fel különböző szinteken, amelyek megoldása szabja meg a fejlődés általános menetét. Az a törekvés pl. hogy egy-egy feladat feldolgozásához a szükséges kiegészitő tevékenységet (pl. a felhasználó számára kényelmesebb, magasabb szintü programozási nyelvről a gépi nyelvre való forditást) magával a géppel végeztessék el, állandóan további sebesség-növelési szükségleteket táplál. Ez utóbbi viszont -- mivel szükségképpen a gép központi egységeiben kerül kielégitésre -- ellentmondásra jut a perifériák lassubb müködésével. Ennek az ellentmondásnak a feloldását először software-eszközökkel biztositják (multiprogramozás), ám ez ujabb ellentmondást szül. A felhasználót ugyanis nemcsak, hogy elzárja a gépen futó programjától, amennyiben arra kényszeriti, hogy passzivan várja ki — ha programja a gépben már sorra is került -- , amig a gép elvégzi a következő részteendőt, hanem esetenként még a sorrekerülésig is tetemes várakozásra kényszeriti. Az ujabb ellentmondás egy hardware-eszköz (timer) és egy sajátos feldolgozási mód (időszeletelés) oldja fel.

Nem kevésbé jelentős a következő, ezzel összefüggő tendencia, amely ugyancsak alkalmas az ellentmondások (fejlődési folyamatban játszott) szerepének illusztrálására. A gépre viendő feladatok megoldási menetének teljes automatizálása megoldotta ugyan azt az ellentmondást, amely a gyors megoldás szükséglete és a hagyományos (nem számitógépes) eljárások nehézkessége és lassusága között állt fenn, de egyben elzárta a felhasználót a megoldási menetbe történő (egyes feladat-tipusoknál szükséges) tetszés szerinti beavatkozások lehetőségétől. A dialógus üzemmód (a gép kihasználását biztositó időszeleteléses üzemmóddal együtt)

megoldja ezt az ellentmondást, mivel lehetővé teszi a programhoz való -- futás közbeni -- hozzáférést. A dialógus üzemmód azonban ellentmondásba kerül a meghonosodott és rendkivül hasznos forditási gyakorlattal, amely a felhasználók számára kényelmes, magasabb szintü programozási nyelvek és a futtatható gépi-kód programok közötti hatalmas távolságot képes áthidalni. Ugyanis a forditás tetemes gépi időt köt le, tehát a dialógus üzemmód, azon a kétségtelen előny mellett, hogy lehetővé tette inkrementális forditási módszerek alkalmazását [80], [81], [82], a forditóhoz való gyakori hozzáfordulás révén jelentékenyen rontja a gép hatásfokát. Ennek az ujabb ellentmondásnak a megoldására előrelátható an alkalmas lesz a formulavezérlésü számitógépek -- mind nagyobb jelentőségre szert tevő -- irányzata azzal, hogy lehetőséget teremt egy magasabb szintü programozási nyelvvel izomorf gépi nyelv megválasztására és ezzel a forrásnyelv és a futtatható gépi kód közötti távolság jelentős csökkentésére.

Ugyancsak általános tapasztalat, hogy a komplex fejlődési folyamatból valamely elem kiemelése és egyoldaluan gyorsabb fejlesztése (habár ennek egyes területeken rendkivül fontos eredményei lehetnek) ismét csak ellentmondásokhoz vezet, kiváltva ezzel más elemek fejlesztésének szükségleteit is. Amikor pl. a müködési sebesség (sok szempontból nagyon fontos) növelése ellentmondást idézett elő, a központi gép és a perifériák müködése között, akkor -- egyéb, már emlitett megoldási módok mellett -- napirendre került a perifériák fejlesztése (minőségileg ujabb perifériák kialakitása, a cserélhetőség és variálhatóság biztositása), valamint a gép alapstrukturájának módositása (az autonóm csatornák alkalmazása az átlapolt müködés biztositására) stb. Ez a tapasztalat amellett szól, hogy a számitógépek jellemzőinek arányos fejlesztése állandóan érvényesülő követelmény, ami azonban a gyakorlatban rendszerint ugy teljesül, hogy valamely jellemző időlegesen kiemelt fejlesztése ellentmondásba jut a többiek lassubb haladásával, s az ilyen ellentmondások feloldásának szükségessége lendíti előre az utóbbiak fejlesztését.

Végül ugyancsak általános tapasztalat, hogy egyes előnyökről való lemondás, más (nagyobb) előnyök érdekében, nem szokott végleges lenni, ellenkezőleg, elvesztett előnyeit magasabb szinten nyerheti vissza az ember. Jól illusztrálja ezt az a már emlitett példa, hogy a felhasználó a gép hatékonyságának biztositása érdekében lemondott arról az előnyéről, hogy közvetlen kontaktusban legyen a megoldás alatt lévő feladatával (tehát arról, hogy a megoldás menetébe beleszólhasson). Ez a lemondás azonban nem volt végleges, elveszitett előnyét a felhasználó lényegesen magasabb szinten (real-time módon) nyerte vissza az időszeleteléses, dialógus üzemmódu gépekkel. Sőt -- hozzátehetjük -- mindazon előnyök birtokában nyerte vissza a feladattal való közvetlen kontaktus lehetőségét, amelyek érdekében korábban lemondott róla.

Az általános tanulságok számbavétele azzal a kitűzött [illegible]

... általános tendenciák számbavétele azzal a következménnyel jár, hogy bármely, a számitógépek (hardware és software) fejlődésére vonatkozó prognózis felállitása esetén célszerü a fejlődési folyamat (néha csak csira-szerüen jelentkező) ellentmondásait szem előtt tartani, mert ezeknek (gyakran egyáltalán nem kézenfekvő) megoldási lehetőségei rajzolják meg a jövőbeli fejlődés utjait.

4.2. A főbb fejlődési tendenciákból levonható konkrétabb következtetések

A számitógépek fejlődéstörténetének és különösen a harmadik generációs gépek, ill. géprendszerek legfejlettebb csoportjánál érvényesülő tendenciáknak az elemzése lehetővé teszi az alábbi, főbb következtetések levonását:

- a közeljövőben kibontakozó fejlődés a rendszerjellemzőket helyezi előtérbe egy-egy uj számitógép (-rendszer) fejlettségének és jellegzetességeinek megitélésénél. (Ezek: a rendszer -- általában nagyfoku rugalmasságot biztositó -- struktúrája, a feldolgozási folyamat megszervezésének módja és a rendszer megbizhatósága),
- a fejlesztés egyik vezető szempontja a számitógépek (-rendszerek) használhatóságának alapvető megjavitása, aminek során lehetővé kell válnia, hogy gépi szolgáltatásokat a számitógéphez (-rendszerhez) nem értő, a jelenlegi előkészitési és kezelési módokban nem járatos felhasználók is közvetlenül és széleskörüen igénybe tudjanak venni,
- döntő jelentőségünek tünik a jövőre nézve a párhuzamositási törekvések térhóditása mind a számitógép (-rendszer) funkcionális egységeinek müködésére, mind a feldolgozandó feladatokra vonatkozóan. Ennek kapcsán biztositani kell a többszörös hozzáférésü rendszerek fejlesztésének és az interaktiv használatnak a lehetőségét,
- növekvő-félben van a kisgépek, valamint a számitógéphálózatok és számitórendszerek jelentősége.

Az emlitett szempontoknak megfelelő hardware és software-fejlődés részleteire vonatkozóan az alább felsorolt tendenciák kiemelését tartjuk szükségesnek:

1. A negyedik generációs gépek és számitórendszerek elembázisára (mint nem alapvető, de figyelmen kivül továbbra sem hagyható jellemzőre) várhatóan az LSI technika széleskörü elterjedése lesz jellemző, az ionimplantációs technológia alkalmazásával. A félvezető müködési elven alapuló eszközök mellett rohamosan tért hóditanak az optikai elven müködő eszközök (elsősorban nagykapacitásu háttértárolók formájában). Ezzel egyidejüleg olyan fizikai jelenségek felkutatása és számitógépekben való alkalmazása is várható, amelyek közvetlenül megvalósithatóvá tesznek olyan folyamatokat, amelyek szükségessége számitástechnikai oldalról merült fel, de amelyek a jelenlegi eszközökkel csak nehézkesen valósithatók meg (mint ahogy pl. a mágneses domainek kutatásától várható a felszabadult, nem folytonos memória-tartományok "összetolásának" közvetlen megoldása).

2. A funkcionális egységek (lásd alább) "finomstrukturájára" várhatóan a mikroprogramozás technikája lesz jellemző (összhangban a hardware és software részleges egymásba-olvadásának tendenciájával). A moduláris bővithetőség követelménye miatt ugyanis az uniformizált egységekből való felépités az áramköri integráció fejlődésével változatlanul jelentős törekvés marad. Az első állomás — még áramköri szinten -- az univerzális (Peirce- és Sheffer-féle) logikai müveletek alkalmazása mind a kapuáramkörökben (NOR- és NAND-kapuk), mind a regiszterek tároló-elemeiben. Nagyobb mértékü áramköri uniformalizálást tesz lehetővé a küszöblogika alkalmazása [44]. Fokozottabb elterjedése azonban csak a megfelelő szintézis-módszerek javulásával várható A küszöblogikai kapuáramkörök ECL-áramkörökkel is realizálhatók, igy késleltetési idejük nanosec-okra leszoritható. További előnyük, hogy univerzalitásukból fakadóan bonyolultabb (többszintü) logikai müveleteket is egyetlen fokozatban képesek realizálni, ami ismét a késleltetési idő csökkenését eredményezi. Pl. egy trigger-kapcsolást legalább két NOR- vagy NAND-kapuval, két szintben tudunk csak realizálni, mig a küszöb-logikában egyetlen visszacsatolt kapu-áramkörrel is realizálható, ami gyorsabb müködésü tárolóelemet eredményez. (Itt emlitjük meg, hogy a nem trigger-kapcsolásokat realizáló, hanem a kapacitiv töltéstárolás elvén müködő dinamikus monolitikus integrált MOSFET tárolók részben hasonló okok miatt, részben rendkivül alacsony teljesitmény-szükségletük miatt szintén elterjedőben vannak.) Az SSI és MSI áramköri készletek kialakitásánál is ügyeltek az univerzális alkalmazhatóságra, de itt már nem a belső áramköri uniformizálás volt a cél, hanem az integrált áramkörök és a belőlük összeállitott rendszerek egymáshoz illeszthetősége. Nemzetközi szinten sikerült uniformizálni (standardizálni) az interface-előirásokat, és a különböző cégek kompatibilis áramköri készleteket fejlesztettek ki. Az áramköri integráció növekedésével ismét magasabb szinten kellett biztositani az egységek univerzalitását és

kompatibilitását. Jelenleg erre a legjárhatóbb utnak a könnyen módosítható mikroprogramokkal vezérelt funkcionális egységek (pl. mikroprogramozható processzorok) létrehozása látszik.

3. A negyedik generációs gépek, ill. számítórendszerek strukturájában igen nagy változások bekövetkezése és fokozatos elterjedése várható. A fontosabb tendenciákat ezen a területen a modularitásra és a hálózatösszekapcsolásra való törekvés fogja jellemezni. A Neumann-féle klasszikus funkcionális egység-tagozódásnak egyrészt a további finomodása figyelhető meg, másrészt kiszélesedik a funkcionális egységek egymásbaolvadásának folyamata. (E szoros kölcsönhatásban végbemenő kétirányu fejlődés formája át a számítógépek, ill. rendszerek strukturáját.) Számitani lehet az "uj" funkcionális egységek (vezérlő processzorok, kommunikációs processzorok, perifériás processzorok stb.) széleskörü elterjedésére és változatos alkalmazására. Fejlődni fog a funkcionális egységek párhuzamos működésmódokra való alkalmassága. A feldolgozómü párhuzamos müködési készsége jelentősen javulhat az

aritmetikai-logikai müveletvégző egységek megsokszorozásával, amely utóbbiakba a már korábban is használt scratch-pad memóriáknál jóval nagyobb kapacitás fog beleolvadni, mint az operativ tároló szupergyors része. Igy gyakorlatilag magában a memóriában lehet (egyszerre sok rekesz tartalmán) müveletet végezni. Ugyanakkor jelentősen meg kell növelni a főtároló kapacitását, a hozzáférési idő romlása nélkül. Méginkább vonatkozik ez a háttértárolókra (ahol -- feltehetően -- még a 10^{15} bit tárolókapacitás sem irreális igény), hiszen a gép kényelmes használhatóságának döntő feltétele a kisegitő tevékenységek átruházása, ami rendkivül nagy tárolókapacitás-szükséglettel jár, és méginkább növeli ezt a szükségletet a tudakozó-informativ rendszerek megteremtésének irányzata. Előrelépés várható abban az irányban is, hogy a mikroprogram-tároló a szupergyors tárolóból tetszőlegesen kialakitható terület legyen. Ez felveti annak a tendenciának a fokozott érvényesitését, hogy a szupergyors tároló és a főtároló, valamint a főtároló és a háttértár közötti határ az előállók javára fokozatosan eltolódjék. Magának a főtárolónak a dinamikus kioszthatóság követelménye miatt célszerü lapszervezésünek lennie, esetleg valamilyen általánositott (többdimenziós) értelemben. A feldolgozó müben helyet kaphatnak hardware veremmemóriák, asszociativ memóriák és más szervezésü hardware- strukturák is (természetesen kisebb kapacitásuak), pl. a kifejezés-feldolgozások megkönnyitésére, laptáblázatokhoz stb.

4. A hálózatba-kapcsolódás adott feltétele a rendelkezésre álló kommunikációs (adatátviteli) hálózat, amelyre, mint egyik fontos objektiv bázisra támaszkodni kell. Ez megköveteli, hogy az adatátviteli hálózat megbizhatóságát és átviteli sebességét a lehetőségek szerint növeljék (pl. PCM modemek). Az általános hasznu kommunikációs hálózat fejlesztésénél azonban -- a távadatfeldolgozás kivételesen nagy jelentősége miatt -- már célszerü a számítórendszer fejlesztése révén kialakuló igényeket fokozottan figyelembe venni. A távlati cél ezen a területen a távirócsatornán való átviteltől fokozatosan fejlődni a multiplex te-

lefoncsatornák és speciális hirközlőcsatornák, valamint rádiócsatornák felhasználásával kialakitandó általános és komplex átviteli rendszer megteremtése felé.

5. Igen fontos fejlődési tendencia a fejlett számitógépek (-rendszerek) és az adatátviteli rendszerek szerves egybekapcsolódásának folyamata, amely nagy számitógéphálózatok kialakulása felé mutat.

6. A számitógépeken, -hálózatokon és -rendszereken belül rendkivül módon megnövekszik az átviteli egységek (kommunikációs processzorok) szerepe. A kommunikációs processzorokat oly módon célszerü fejleszteni, hogy elvégezzék az összeköttetési vonalakon mindkét irányban átvitelre kerülő információk szerkesztésének feladatát is.

7. Ki kell fejleszteni az ember-gép, folyamat-gép, gép-gép (általában objektum-gép) információs kapcsolatot magas szinten lehetővé tevő (általában real-time feldolgozáshoz alkalmas) periféria készleteket, amelyek sorában

fokozatosan növekszik a dialógus-üzemmód szükségleteit kielégitő berendezések (mindenekelőtt az univerzális display) szerepe. A távprognózisok nagy jelentőséget tulajdonitanak az akusztikus információ be- és kivitelére szolgáló komplexumok, valamint a stilizált kéziratot vagy a tetszőleges nyomtatott szöveget beolvasó berendezések fejlesztésének. Meg kell azonban jegyeznünk, hogy a közvetlen jövőben megmarad a hagyományos I/O berendezések tömeges használata javitott hatékonysági paraméterekkel.

8. A modularitási törekvések, valamint a rendkivül változatos perifériás berendezések növekvő alkalmazása miatt jelentősen megnövekszik a standard interface-módszerek szerepe. Nem kétséges, hogy igen fontos feladatokat kell megoldani az információfeldolgozáshoz szükséges berendezések szabványositásának és egységesitésének területén.

9. A hardware és a software eszközök szoros kölcsönhatásban végbemenő fejlesztésének az egyik legfőbb célja a felhasználó lehetőségeinek kiszélesitése, hogy könnyen, kényelmesen (és a szükségleteknek megfelelően real-time módon) igénybe vehessen minden, a rendszer által nyujtott lehetőséget (természetesen mások érdekeinek megsértése nélkül). Ez azt jelenti, hogy hozzáférést kell biztositani -- fokozatosan egyszerüsödő formában -- a rendszer egészéhez. Emiatt a felhasználóra orientált interaktiv rendszerek jelentőségének gyors növekedése várható. Ennek együtt kell járnia az időosztásos üzemmód legkülönfélébb változatainak (főleg a nagy rugalmasság lehetőségeit biztositó időszeletelésnek) a fejlesztésével mind hardware- mind software-eszközök utján, ami — egyebek között — fokozottan előtérbe állitja a perifériák kezelőrendszereinek fejlesztését. Igen fontos követelmény a dialógus üzemmód optimális feltételeinek megteremtése, amelytől lényeges előrelépés várható a számitógéppel eredményesen kezelhető feladatok bővitésének területén.

10. Az operációs rendszerekkel szemben már jelenleg érvényesülő nagy követelmények várhatóan fontos változásokhoz vezetnek. Közöttük igen lényeges az a tendencia, hogy a jelenlegi operációs rendszerek nagy részét hardware eszközökkel fogják realizálni, (ami — mint említettük — speciális processzort feltételez a rendszer vezérlésének szervezésére.) Döntő előrelépésnek tekintik azoknak a törekvéseknek a kifejlődését, amelyek a software modularizálására irányulnak, felvetve ezzel a standard interface módszereinek alkalmazására vonatkozó kérdést a software-re is. Várhatóan erőfeszitések történnek egyes software elemek tervezésének automatizálására is. Bár egyelőre nem látszik még jele az operációs rendszerek kezelése egyszerüsödésének, de a 4.1. rész végén kifejtettek értelmében számitani lehet arra, hogy az operációs rendszerek hasznossága és bonyolultsága közötti ellentmondás is előbb-utóbb megoldást követel. A megoldás egyik várható iránya

a különböző gépcsaládokhoz kifejlesztett operációs-rendszerek egységesitése lehet, amely a jelenleg még nagyon kusza terminológia szabványositásával indulhat meg. Egy további irányzat lehet univerzális munkavezérlő (job control) nyelv létrehozása, amelyre az első törekvések már láthatók.

11. Jelentős fejlődés várható a memóriakezelés nagy fontossággal biró területén. Ebben igen lényeges — s már a mostai szükségleteket is pontosan kifejezi — a virtuális memóriakezelés fejlődése. Várható a virtuális gép elvének fokozott érvényrejutása és elterjedése, amihez alapot biztositanak a számitógéphálózatok és számitórendszerek. Feltehetően előtérbe fognak kerülni a virtuális operációs rendszer kialakitásának követelményei is.

12. A programozási nyelvek területén kétirányu fejlődés várható: Az egyik fejlődési irány az univerzális nyelv fejlesztése felé mutat. Itt azonban meg kell jegyezni, hogy hasonlóan az eddigiekhez, a jövőben sem várható a minden terület átfogására törekvő univerzális programnyelvek (PL/1, Algol-68) jelentős mértékü térhóditása, annak ellenére, hogy van továbbmutató, elméleti jelentőségük. Ellenben lényeges előrehaladás várható az olyan univerzális nyelv-fejlesztési törekvésekben, amelyek céljai az egységes nyelvkezelésre és forditókezelésre, valamint a teljes rendszer leirására vonatkoznak. Ezeknek a törekvéseknek, egybevetve a formulavezérlés elvével, a magasszintü gépi bázisnyelv megteremtésének perspektivikus irányzatával, nagy jövőt lehet jósolni. A másik fejlődési irány a problémára- és ezzel a felhasználóra orientálás felé mutat. A programozási nyelvek fejlődésének ez az irányzata rendkivül fontos és fokozatosan növekvő tendenciát jelez. Ebben a tendenciában nagy jelentőséget kell tulajdonitani az interaktiv használat biztositásának.

13. A számitógépek és -rendszerek megbizhatóságának javitása várhatóan a már kidolgozott automatikus hibafelfedés és -javitás elemeinek

(amelyek mind az adatokra és a programokra, mind a rendszerekre, mind a műszaki berendezésekre vonatkoznak) széleskörű elterjedésében realizálódik. Ebben fontos eljárásnak minősíthető a feldolgozási történet pontos vezetése, mint a rendszer által megvalósítandó funkció.

14. A közeljövőben várható a legmodernebb hardware- és software-eszközökkel ellátott kisgépek jelentőségének lényeges megnövekedése, amelyeknél az önálló alkalmazástechnikájuk kidolgozása mellett a hálózatokba, vagy számítórendszerekbe való bekapcsolhatóság nélkülözhetetlen követelmény.

15. A számítógépek és rendszerek mai szintű fejlesztése és alkalmazása nemzetközi összefogást és koordinálást, az anyagi erők optimális és hatékony elosztását követeli. Hazai vonatkozásban az adottságainkkal összhangban álló irányok kiemelése, a fejlesztési programok irányításának centralizálása lenne fontos a rendelkezésre álló szellemi erők hatékonyságának fokozása érdekében.

JAVASLATOK

1. Kiindulva abból, hogy a negyedik generációs számítógépek és -rendszerek elembázisa várhatóan (egyes funkcionális egységekben döntően) LSI alapu lesz, célszerű fejleszteni az LSI-technológiákat, és azok alapjaként az ion-implantációs technológiát, elsősorban a következő területeken:

   - szupergyors előmemóriák (elsősorban stratch-pad memóriák és mikroprogram-tárolás céljára),
   - LSI blokkokból modulárisan bővíthető gyors főtárolók.

   Folytatni kell azokat a kutatásokat, amelyek a negyedik generációs számítógépekben és -rendszerekben előreláthatóan szükséges folyamatok közvetlen megvalósítására alkalmas fizikai jelenségek feltárására és törvényszerűségeik megállapitására irányulnak.

2. Hardwaregyártó-kapacitásunkat és az ESzR-ben már meglévő jelenlegi helyzetünket figyelembe véve indokolt folytatni a kisgépek fejlesztését és gyártását, valamint alkalmazástechnikájuk széleskörű kidolgozását, szigoruan szem előtt tartva -- különösen hardware-vonalon -- az ESzR programot. Elsődleges szempontként kell érvényesiteni a flexibilitást kettős értelemben is:

   - Alapvető követelmény a kisgépnek korszerű nagygéphez (számítórendszerhez) szatellitként való kapcsolhatósága, amiben a megfelelő ESzR program az irányadó. Mivel azonban a közvetlen közeljövőben várhatóan nem változik meg a helyzet, hogy nagygép-parkunk meglehetősen heterogén, ezért a fejlesztendő kisgépeknél biztositani kell a kapcsolódás lehetőségét egymástól jelentősen eltérő nagygépekhez is.
   - A kisgépeknek -- önálló egységekként dolgozva -- rendkivül sok fajta feladat megoldására kell alkalmasaknak lenniük

tehát ilyen értelemben is magasszintü flexibilitással kell rendelkezniök. A kisgép optimális alkalmazhatóságát, egy-egy területen, mikroprogram-rendszerének tág határok között való változtathatóságával is célszerü fokozni.

3. Kívánatos --hazánkban üzemelésre kerülő -- bármilyen számítógép esetén az ESzR perifériákkal való kapcsolás lehetőségét megteremteni, folytatva az ilyen irányu tevékenységet. Ugyancsak megfontolandó az ESzR perifériák gyártásának fejlesztése és kibővitése, főleg a többcélu vizuális megjelenitő berendezések irányában.

4. Országos számitóhálózat megteremtése egyelőre nem látszik reálisan tervezhető feladatnak. Várható viszont lokális centrumok kialakulása

(terminál-hálózattal és gép-gép kapcsolással). Ez a hazai fejlesztésben is napirendre tüzi a távadatfeldolgozással kapcsolatos feladatokat, amelyek közül kiemeljük:

- az adatátviteli hálózat felhasználását lehetővé tevő modemek kifejlesztését (itt felvetődhet esetleg a licencvásárlás lehetősége), és
- a hirközlő hálózat további fejlesztésénél a számitástechnikai igények fokozatos figyelembe vételét.

5. A software-fejlesztés legfontosabb irányát hazánkban a kisgépek alkalmazásával összefüggő követelmények jelölik ki. Ezért:

- Tekintettel arra, hogy a kisgépek alkalmazásában a sokirányuság célszerü, magának a rendszer-software-nek számos változatát kell kidolgozni különböző problémacsoportokra orientáltan. Ez felveti a különböző szakmáju felhasználók által, (alkalmazási módjukkal együtt) könnyen elsajátitható probléma-orientált nyelvek és rendszerek fokozott fejlesztésének és implementálásának követelményeit.
- A már emlitett alkalmazástechnikai kutatásokra támaszkodva ki kell dolgozni a hazánkban üzemeltetésre kerülő kisgépek széleskörü alkalmazási programcsomagjait.
- Kívánatos lenne, hogy egy-egy országos szerv fogja össze a különböző területekre vonatkozó rendszer-software kidolgozásának munkálatait, s ugyancsak az alkalmazási programcsomagok kidolgozását is.
- Feladatként jelentkezik a software interface kidolgozása, beleértve ide vonatkozó általános módszerek kidolgozását is, amelyeknél feltehetően felmerülhet az ESzR nagygépeken belüli kezelőrész (fogadórendszer) elkészitése is. Ennek mind a felhasználandó, mind az eladandó kisgépek szempontjából je-

lentősége van.

6. Célszerűnek látszana a számítógéptudományi és a számítástudományi kutatások és fejlesztések elvi irányításának centralizálása, a hazai szellemi kapacitások hatékonyságának növelése céljából.

Irodalomjegyzék:

1. Neumann J.: A számológép és az agy. Gondolat. Bp. (1964)
2. Sz.A.Lebegyev -- V.A.Melnyikov: Obscseje opiszányije BESZM i metodika vüpolnyenyija operacij. Moszkva. (1959)
3. E.F.Codd: Multiprogramming. Advances in Computers, 3.kötet, New-York. (1962)
4. Az NDK ESzR főkonstruktőre: A jelenleg nem ESzR nomenklaturához tartozó kis-, speciális és vezérlő számítógépek perspektivái. Tanulmányvázlat, (1972.március)
5. A.W.Burks -- H.H.Goldstine -- J.von Neumann: Preliminary discussion of the logical design of on electronic computing instrument. Inst. for Advanced Study, Princeton.(1946. junius 28)
6. W.N.Papian: The M.I.T. Magnetic Core Memory. Proc. Eastern Joint Computer Conference. (1953)
7. J.A.Rajchman -- A.W. Lo: The Transfluxor.Proceedings of IRE. (1956)
8. Chu, Yoohan: Digital Computer Design Fundamentals. McGraw-Hill, New-York. (1964)
9. C.J.Dakin-- C.E.G.Cooke: Circuits for Digital Equipment. Illife Books Ltd., London. (1969)
10. W.F.Chow -- J.Cubert: A Key to Nanosecond Switching. Electronics. (1963 okt.)
11. L.M.Spandorfer: Large Scale Integration -- an Appraisal. Advances in Computers, 7. (1966)
12. W.J.Poppelbaum: What Next in Computer Technology? Advances in Computers, 9 (1968)

13. L.L. Vadasz -- J.T. Chua -- A.S. Grove: Semiconductor random-access memories. IEEE Spectrum. (1972 május)

14. G.A. Fedde: Plated Wire Memories: Univac's Bet to Replace Toroidal Ferrite Cores. Electronics. (1967 május)

15. A.H. Bobeck -- H.E.D. Scovil: Magnetic Bubbles Scientific American. (1971 julius)

16. J.A. Rajchman: Holografic Optical Memory. Applied Optics, 10. (1970)

17. D.R. Hill: Man-Machine Interaction Using Speech. Advances in Computers, 11. (1971)

18. J.C. Murtha: Highly Parallel Information Processig Systems. Acvances in Computers, 7. (1966)

19. W.J. Bouknight et al: The ILLIAC-IV System. Proc.of the IEEE. Vol.60. No.4. (1972) pp.369-388.

20. D.A. Poszpelov: Vvegyenyije v tyeoriju vücsiszlityelnüh szisztyem. Izd. Szovj. Radio. Moszkva (1972)

21. D.W. Barron: Computer Operating Systems. Chapman and Hall LTD, London. (1971)

22. H.W. Lawson Jr.-- B.K. Smith: Functional Characteristics of a Multilingual Processor. IEEE Transaction on Computers, 7. (1971)

23. A.M. Johnson: The Microdiagnostics for the IBM System 360 Model 30. IEEE Transactions on Computers, 7. (1971)

24. M.J. Flynn— A. Podvin: An Unconventiodal Computer Architecture: Shared Resource Multiprocessing. Computer. (1972 márc.-ápr.)

25. D.E. McIntyre: An Introduction to the ILLIAC IV. Computer Datamation, (1970 április)

26. C.G. Bell -- R.C. Chen-- S.L. Rege: Effect of Technolgy on Near Term Computer Structures. Computer. (1972. márc. - ápr.)

27. S.G. Tucker: Microprogram Control for System/360. IBM Systems J. 4. (1960)

28. E.L. Braun: Digital Computer, Design. Academic Press, New-York. (1963)

29. M.V. Wilkes: Slave Memories and Dynamic Storage Allocation. IEEE Transactions on Electronic Computers, 4. (1965)

30. R.W. Watson: Timesharing System Design Concepts. McGraw-Hill, New York (1970)

31. The SHARE 709 System (Cikksorozat). J. of ACM, 6, No.2. (1959)

32. IFIP Guide to concepts and terms in data processing, Ed. by I.H.

Gould. North Holland Publ. Co., Amsterdam, London (1971)

33. ORION programming manual. International Computers Ltd.,

34. T. Kilburn et al.: The Manchester University Atlas Operating System. The Computer J. 4. No.3 (1961)

35. S.Ramani: A language based problem-solver. Second Int. Joint Conference on Artifical Intelligence, London. (1971) pp.463-473.

36. D.N.Freeman: Pseudo-devices in OS/360.Proc.of the 1968 Southeastern ACM regional Conference.

37. H.Katzan, Jr.: APL programming and computer techniques. Van Nostrand Reinhold Co., New York. (1970)

38. M.M.Golds: Time sharing and batch processing: An experimental comparison. Comm of ACM., 12. No.5 (1969) pp.249-259.

39. A.M.Turing: On computable numbers, with an application to the entscheidungsproblem, Proc. of London Math. Soc. (2), 42. (1936-37), Reprinted in: Annual Review in Automatic Programming. 1. (1960)

40. R.M.Davis: Programming Language Processors . Advances in Computers, 7. (1966)

41. I.Flores: Computer Software. Prentice-Hall, Englewood Cliffs, N.J. (1965)

42. C.Strachey: A general purpose macrogeneretor. The Computer J. 8, No.3. (1965) , pp.225-241.

43. I.A.MacLeod: MP/1 - a FORTRAN macroprocessor. . The Computer J. 14, No.3. (1971) pp. 229-231.

44. S.Muroga; Threshold Logic and Its Applications. Wiley et Sons, inc., New York, (1971)

45. A.A.Ljapunov: O logicseszkih Szhemah program. ProblemÜ kibernetiki 1.(1958) pp.46-74.

46. Ju.I.Janov: O ravnoszilnosztyi i preobrazovanyijah szhem programm, DokladÜ akad. Nauk SzSzSzR. 113,(1957) pp.39-42.

47. A.P.Jersov: Programmiruscsaja programma dlja BESZM. Izd. Akad. Nauk, Moszkva, (1958)

48. J.W.Backus -- F.L.Bauer et al: Report on the algorithmic language ALGOL 60. Annual review in Automatic Programming, 2.(1961)

49. Ginsburg, Rice: Two families of languages to ALGOL. J.of ACM, 9, No.3 (1962) pp. 350-371.

50. S.Gorn: Detection of generative ambiguities in context free mechanical languages. J.of ACM, 10. (1963) pp. 196-208.

51. Sz.Sz. Kaminyin -- E.Z.Ljubimszkij: ALMO. Esztergom. (1958)

52. A.P.Jersov -- G.I.Kozsuhin -- Ju.M.Volosin: Vhodnoj jazik szisztyemÜ avtomaticseszkovo programmirovanyija. Moszkva (1961)

53. M. Fleck -- E. Neuhold: Formal Definition of the PL/I. Compile Time Facilities. IBM Laboratory Vienna, Techn. Report TR 25.080. (1968 jun. 28.)

54. K. Walk -- K. Alber -- K. Bandat et al: Abstract Syntax and Interpretation of PL(I. IBM Laboratory Vienna, Techn. Report TR 25082. (1968 jun. 28)

55. P. Lucas et al.: Informal Introduction to the Abstract Syntax and Interpretation of PL/I. IBM Laboratory Vienna Techn. Report TR 25.083. (1968 jun. 28.)

56. K. Alber et al.: Concrete Syntax of PL/I. IBM Laboratory Vienna, Techn. Report TR 25.084. (1968 jun. 28.)

57. K. Alber -- P. Oliva: Translation of PL/I into Abstract Text. IBM Laboratory, Vienna, Techn. Report TR 25.086. (1968 jun. 28)

58. P. Lucas et al.: Method and Notation for the Formal Definition of Programming Languages. IBM Laboratory Vienna, Techn. Report TR 25.087. (1968 jun. 28.)

59. L. Bolliet: Compiler Writing Techniques. Programming Languages, ed. by F. Gennys. Academic Press, London, New York. (1968)

60. J. McCarthy: Recursive funtions of symbolic expressions and their computation by machine. Comm. of ACM, 3(1960)

61. P. W, Abrahams: Symbol manipulation languages. Advances in computers, 9. (1968) pp. 51-113.

62. K. C. Knowlton: A programmers description of L6. Comm. of ACM, 9. (1966) pp. 616-625.

63. K. E. Iverson: A Programming Language, New York, John Wiley et Sons, Inc., (1962)

64. A. D. Falkoff -- K. E. Iverson: APL\360: User's Manual, Yorktown Heights, New York, IBM Corporation, Watson Research Center, (1968)

65. W. C. McGee: The formulation of data processing problems for computers. Advances in computers, 4. (1966) pp. 1-53.

66. J. P. Gelb: Experiments with a natural language problem solving system. Second Int. Joint Conference on Artificial Intelligence, London. (1971) pp. 455-462.

67. M. Kay: Rules of interpretation: An approach to the problem of computation in the semantics of natural languages. Proc. Munich Congr. IFIP. (1962) pp. 79-83.

68. IBm 7040/7044 Operating System, Macro Assembly Program Language, Systems Reference Library File no. 7040-211, Form. C28-6335-1.

69. A. Zamorin -- Sz. Szolovjov.: IV. generációs ESzR koncepciók.

Az egységes dokumentum tézisei, (1972. május (OMFB anyag).

70. C.C.Foster.: The Next Three Generations. Computer. (1972 márc.-ápr.)

71. K.Samelson -- F.L.Bauer.: Sequential Formula translation. Com. ACM3. (1960) pp.76-83

72. W.Kämmerer: Ziffern-Rechenautomat mit Programmierung nach mathematischem Formelbild. Jenaer Jahrbuch II. (1959) pp.396-440; Ziffernrechenautomat mit Programmierung nach mathematischem Formelbild. ZAMM 40. (1960)

73. Kalmár L.: A practical infinistic computer, Infinistic Methods. Proceedings of the Symposium on Foundations of Mathamtics, Warsaw 1959, (Warszawa 1961) pp.347-362.

74. Kalmár L.: Über einen Rechenautomaten, der eine mathematische Sprache versteht. ZAMM, 40.(1960)

75. Kalmár L.: On a digital computer which can be programmed in a mathematical formula language, II Magyar Matematikai Kongresszus, előadáskivonatok II., V.(a matematika alapjai és a matematikai gépek elmélete). Bp.(1960) pp.V-3 -- V-16.; O cifrovoj vůcsiszlityelnoj masine sz progranyiroványiem poszredsztvom formalnovo matematicseszkovo jazika. Kibernetics eszkij Szbornyik, novaja szerija, 1. (1965) pp.215-226.

76. Z. Pawlak: The organization of a digital computer and computable funcions. Bull. Acad. Polon. Sci Sér. Sci. Techn. 8 (1960) pp.41-51. Organization of the addressfree digital computer for calculating simple arithmetical expressions. Ugyanott 8. (1960) 685; Organization of address-free computer B-100. Ugyanott 9. (1961) pp.229-234.

77. V.M. Guskov: Vhodnůj jazi k vůcsiszlityelnoj masina "MIR". Kibernetika, 1.(1965)

78. J.A.N. Lee levélbeli közlése.

79. P.Wegner: Zero address computers.The computer Journal, 5, No.1. (1962)

80. K.Lock: Structuring Programs for Multiprogram time-sharing On-Line Applications, Proceedings of the Fall Joint Computer Conference (1965)

81. J.L.Ryan -- R.L.Crandall -- M.C.Medwedeff: A Comversational system for Incremental Compilation and Execution in a Time-Sharing Environment, Proceedings of the Fall Joint Computer Conference (1966)

82. H.Katzan: Batch Conversational, and Incremental Compilers, Proceedings of the Spring Joint Computer Conference (1969)

Tartalommutató