NJSZT
MŰSZAKI ÉS TERMÉSZETTUDOMÁNYI EGYESÜLETEK SZÖVETSÉGE
NEUMANN JÁNOS SZÁMÍTÓGÉPTUDOMÁNYI TÁRSASÁG

PROGRAMOZÁSI RENDSZEREK '72

SZEGED

1972. augusztus 28-31

I

PROGRAMOZÁSI RENDSZEREK '72
TALÁLKOZÓ

Szeged, 1972. augusztus 28-31

A BEKÜLDÖTT ELŐADÁSOK

MTESZ

NEUMANN JÁNOS SZÁMITÓGÉPTUDOMÁNYI TÁRSASÁG

A SZERKESZTŐBIZOTTSÁG:

TARTALOMJEGYZÉK

A MINSZK-22 SZÁMITOGÉP EGY OPERÁCIOS RENDSZERE

Makay Árpád - Máté Eörs - Matievics István
JATE Kibernetikai Laboratórium

Az operációs rendszer feladata

- a feladatok közötti folyamatosság biztositása a számitógép idejének jobb kihasználása céljából,
- a programok futtatását végző operátorok tevékenységének egységesitése,
- a heterogén fordítók rendszerbe foglalása kezelés és célprogram tekintetében,
- a programfutás felfüggesztésének, illetve a program szegmentálásának biztositása,
- a programkönyvtár kialakitása,
- a feladatcsoportok végrehajtása előre programozott módon.

Számitógépünk fő input-perifériája egy FACIT lyukszalagolvasó, amelyet a start-olvasó helyére illesztünk. Az operációs rendszer vezérlő-irógépe a teletype (ASR-33 az eredeti helyett), ezért a FACIT és teletype csatornákat regiszterek és utasitások szempontjából hardware uton szét kellett választani. Egy másik hardware átalakitás biztosit irásvédelmet a memória felső 400 (nyolcas számrendszerben) rekeszének, amelyből 300 állandó tartalmu, 100 pedig felülirható. A megszakitó-rekeszek szerepét is ugyanennek a tartománynak egy szelete vette át.

A rendszerbe illeszkedő programokra csupán a következő korlátozások érvényesek:

- a védett tartományt (400 rekesz) sem program, sem változók tárolására nem használhatják,

- a feladat végrehajtása után a vezérlést a védett tartomány egy fix helyére adják át (a futás nem stop-utasitással fejeződik be).

Az operációs rendszernek állandóan a védett tartományban levő része a SUPER. A programok futása során tartja a kapcsolatot a gépkezelővel, a program futásához szükséges információkat, a szegmensek közötti kommunikációt biztositja, azonkivül a munkaprogramokból is hivható, standard tevékenységet végző szubrutinokat tartalmazza. A SUPER váltja ki a monitort, amely a program, illetve feladatcsoport inditásához szükséges adminisztrációt végzi, kitölti a kommunikációs mezőt, programfutás felfüggesztést, újrainditást hajt végre. A MONITOR a munkaprogram futása alatt nincs a memóriában, sőt virtuálisan nincs helyigénye (munkája befejeztével visszaáll a kiváltását megelőző memóriaállapot), de bármilyen rendszerprogram számára hozzáférhető. Az operációs rendszer részei, a programkönyvtár, a forditók (ALGOL, INZSENYER, szimbólikus kódrendszer, MISI) a 0-ás mechanizmus mágnesszalagján helyezkednek el.
A védett terület kommunikációs mezeje a feladatnevet, programnevet, mágnesszalag-mechanizmusok logikai táblázatát, kulcsszót, dátumot tartalmazza. A futó program számára bármikor hozzáférhetők, illetve tartalmuk megváltoztatható a SUPER egy szubrutinjának hivásával.

A gép dinamikus várakozó állapotában, valamint a program futása során a teletype-on utasitásokat gépelhetünk a SUPER számára. Ezek az utasitások a vezérlőpulton kézzel elvégezhető tevékenységeket interpretálják (memória adott cimére történő irás, olvasás, az aktuális utasitásszámláló lekérdezése, adott cimről, illetve stop utasitásról történő továbbinditás). Az utasitások karaktereinek gépelése között, illetve az utasitás végrehajtása után a gép tevékenysége folytatódik. SUPER-utasitással érhető el a program futásának fel-

függesztése, aminek hatására a rendszer-mágnesszalagról a MONITOR a memóriába kerül, biztositva a korábbi memória-tartalom megőrzését. A gép további tevékenységét a MONITOR-nak szóló üzenetek vezérlik.
Mind a futó program, mind a MONITOR a SUPER-től kérheti egy program vagy szegmens programkönyvtárból való beolvasását és elinditását, egyben biztositja a program végrehajtása után a vezérlés visszaadását a hivónak vagy a SUPER-nek. Ugyancsak a védett tartomány egy szubrutinja közvetiti a teletype-ról a futó program számára szóló üzeneteket.
A MONITOR tevékenységeit a teletype-on gépelt utasitásoknak megfelelően végzi. Az utasitások vezérszóból és paraméterlistából állnak.
A kommunikációs mező kitöltésére szolgálnak a JOB, ASSGN, UPSI, DATA utasitások. A feladatnév 6 karakteres, a logikai táblázat kilenc szimbólikus mágnesszalagnévhez, két input és öt output perifériához rendel abszolut egységszámot. A feldolgozáshoz 36 kulcs állitható be.
Az INTER utasitás a MONITOR kiváltásakor megszakadt program megőrzését biztositja. Mágnesszalagon tárolja a memóriatartalmat, a regiszterek állapotát, a megszakitó-rendszerről információkat. Egyszerre négy program lehet a felfüggesztés állapotában. Korábban megszakitott program futásának folytatását a CONT utasitással érhetjük el. Külön utasitás szolgál a MONITOR kiváltásakor megszakadt, de fel nem függesztett program folytatására. (A megszakitás oka valamilyen rendszerprogram menetközbeni lefuttatása lehetett: memóriakiirás lyukszalagra, szélesnyomtatóra stb.)

A forditóprogramok aktivizálására (gyakori használatuk miatt) egy-egy utasitás szolgál. Paraméterlistájukban a forditás módjára vonatkozó információkat közlünk: a forditók rendszerbe illesztésének egyik feladata volt a kulcsállásokkal, pultszó bitjeivel közölt paraméterek helyettesitése a kommunikációs mező megfelelő részeivel. A célprogram programkönyvtárba való ujrabetöltését

szintén MONITOR-utasitással kérjük.

Programkönyvtárbeli program memóriába töltését és a programhoz rendelt cimtől való elinditását az EXEC utasitással kérjük. A tényleges betöltést és inditást a SUPER végzi. A MONITOR feladata egyrészt információk leolvasása a programkönyvtár katalógusából, másrészt a program futásának befejeztével a megfelelő visszatérés biztositása. (A vezérlést visza kaphatja a MONITOR, a SUPER, egy futó program, illetve egy automatikusan működő programcsomag-vezérlő rendszerprogram.)

Speciális rendszerprogramok inditására külön vezérszavak adhatók. A MONITOR tevékenységeinek sora nem lezárt, állandó továbbfejlesztése folyik.

A rendszer-mágnesszalagon találhatók a fordítók és a programkönyvtár.
A MINSZK-22 mágnesszalagjainak információtárolási módja szükségessé tette egy programkönyvtár-katalógus létrehozását. A katalógus tartalmazza a program azonositására szolgáló nevet, a program inditási cimét (relativ cimes programok esetében a kezdőcimet), a szegmensek számát, illetve az első szegmens mágnesszalagos elhelyezési cimét. Maguk a szegmensek a könyvtár szövegrészében folyamatosan egymás után találhatók. A szegmenscimkék további szükséges információkat közölnek (hossz, memóriabeli kezdőcim, kontrollösszeg).
A rendszer mágnesszalag programkönyvtára a szolgáltató programokat tartalmazza. Külön szalagon helyezkedik el a felhasználói programkönyvtár, e mellett egyéni programkönyvtárak is kialakithatók. A felhasználói programkönyvtár felhasznált programjait a MONITOR jelzéssel lát el, ez alapján rendszerprogram gondoskodik a nem használt programok törléséről.
A karbantartást (program törlése, csatolása, a katalógus listázása stb.) rendszerprogram végzi.

Nemcsak közvetlenül a teletype-ról indithatunk el egy feladatot, hanem előre megadhatjuk valamely programcsomag összes információit. Off-line üzemmódban lyukszalagra gépeljük mindazokat az utasitásokat, kommentárokat, amelyek a MONITOR-nak, gépkezelőnek, rendszerprogramoknak, munkaprogramoknak szólnak, majd egy EXEC nevü vezérlőprogramot inditunk, amely a szalagot beolvassa, tárolja a rendszerszalag ecélra fenntartott intervallumán. A védett tartomány egy szegmense gyakorolja a felügyeletet a programcsomag elemei felett, igy automatikusan folyik a feladatok végrehajtása.

Az operációs rendszer másfél éves üzemelési tapasztalatai alapján sikeresnek mondhatjuk. Nem korlátozza lényegesen a munkaprogramok memóriahasználatát, időben jelentős megtakaritást jelent. Kiszolgáló programokból olyan rendszert épitettünk ki, hogy a programpróbák nagy része is az operátorokra bizható, felszabaditva a programozókat egy időigényes tevékenységtől.

Az operációs rendszer hardware és software dokumentációja a József Attila Tudományegyetem Kibernetikai Laboratóriumában hozzáférhető.

A VEIKI SZÁMITÓKÖZPONTJÁBAN KIFEJLESZTETT REX EXECUTIVE ÉS OPERÁCIÓS RENDSZER

Draskóczy Judit, Horniák Gábor, Luka Irén, Pásztor Zoltán

Villamosenergiaipari Kutató Intézet

A RAZDAN-3 számitógép eredetileg nem rendelkezett operációs rendszerrel. Az executive programok kezelése - a laborálás - nehézkes és időigényes volt. Az executive programokat összefogó és kezelésüket egységesitő első operációs rendszer - a REX - 1969-ben készült el és 3 éven keresztül megbizhatóan üzemelt. A számitógép hardware bővitései /megszakitási rendszer, konzolirógép, start-stop rendszerü kártyaolvasó, mágnesszalagos off-line berendezések, stb./ lehetővé és ugyanakkor szükségessé tették az operációs rendszer továbbfejlesztését, korszerüsitését. A REX-2 rendszer kifejlesztése 1971-ben kezdődött és ebben az évben fejeződött be.

A REX - az akkori hardware lehetőségeknek megfelelően - csak rendkivül leegyszerüsitett ember-gép kapcsolatot biztositott. Az executive programok aktivizálása, ill. az executiv programoktól a szolgáltatások kérése a gép kezelőpultján található kulcsok, az ezeken beállitott sorszám-kódok segitségével történt. Az üzemmód egyértelmü kiválasztásához szükséges /általában "igen-nem" tipusu/ kiegészitő információkat a 48 bites pultregiszteren lehetett beállitani. Bizonyos kényelmetlenséget okozott a laborálás során az, hogy az operátor program, valamint az executive programok a különböző közleményeket és hibajelzéseket a pulttól távolabb elhelyezett sornyomtatóra adták ki.

A REX tartalmazott bizonyos védelmet a laborálási hibák ellen is: regisztrálta a kiadott sorszámkódok sorozatát és a nem megfelelő helyen kiadott sorszámkódot nem akceptálta. Az operátor program kezelte az ALGOL és az assembler fordítóprogramokat, az általános könyvtári programokat, a könyvtár gondozására szolgáló programokat, valamint a teszteket.

A program "nucleus" része tartalmazta az i/o rutinokat, amelyek a lyukszalagolvasót és a sornyomtatót kezelték. Megjegyezzük, hogy Számítóközpontunkban, valamint felhasználóinknál kezdettől fogva többféle adatelőkészitő berendezés üzemel. Ennek megfelelően az input rutinok is többféle kód kezelésére vannak felkészitve. A telex, tele, Eichner, ISO kódban lyukasztott információt egységesen 6 bites belső kóddá képezik le. A megfelelő átkódóló kiválasztása automatikusan, a lyukszalagra lyukasztott első értékes karakter /kötelezően mindig "cr"/ alapján történik.

A jelenleg kifejlesztett REX-2 executive és operációs rendszer nucleus része a kezelői utasitásokat tartalmazó jobleirások interpretálását végző és a megszakitási rendszert kezelő operátor programból, az egységes i/o rendszerből és az executive programokat aktivizáló rutinból áll. A rendszer egyidejüleg egy információs és egy vezérlő input-output csatorna-párt kezel.

A kezelői utasitások a funkciót megjelölő utasitásnévből és az utasitáshoz tartozó paraméter-listából állnak. Egy utasitásnak legfeljebb 15 paramétere lehet; az utasitás paramétereinek megadási sorrendje tetszőleges. A listán a paramétereket a "sp" karakter határolja be. A paraméterek a paraméternévből és az előbbivel egybeirt "vál-

tozó" részből állnak. A paraméterek lehetnek logikai, numerikus /oktális vagy decimális/ és string tipusuak. A "változó" rész a logikai paraméterek esetében "üres", a numerikus paramétereknél előjel nélküli egész, a string tipusuaknál pedig tetszőleges alfanumerikus jelsorozat. A paraméterek megadásának szükségességét figyelembe vevő másik osztályozás szerint a paraméterek lehetnek kötelező és nem kötelező jellegüek. Az azonos logikai sorszámu helyen egymást helyettesitő és kölcsönösen kizáró paraméterek szerepelhetnek. Ezeket alternativ paramétereknek nevezzük.

A teljes jobleirást a kezelői utasitások "=" akcept-jellel lezárt sorozata alkotja. Az egy jobleirásban szereplő kezelői utasitások mennyiségére gyakorlatilag nincs korlátozás.

Mivel a kezelői utasitások által inditott müködések, következésképpen maguk a kezelői utasitások is csak meghatározott sorrendben követhetik egymást, a jobleirások helyességének ellenőrzésére a teljes rendszer müködését szintekre osztottuk. Egy-egy szint a hozzátartozó müködések "elinditásának" szükséges és elégséges feltételeit reprezentálja. A hibátlan müködés a rendszert egy ujabb, a korábbinál nem alacsonyabb szintre viszi át, ahonnan azután ujabb kezelői utasitásokkal ujabb müködések indithatók. Összesen 7 rendszer-szintet definiáltunk. A legalacsonyabb szint az "1", ezt az executive programok lehivására szolgáló CALL és a jobot záró END JOB utasitások állitják be. Az operátor program szintaktikusan ellenőrzi a kezelői utasitásokat, ezek sorozatait - a jobleirásokat -, valamint a jobleirások sorozatait és ezzel bizonyos mértékig kizárja a szemantikusan is helytelen müködés-sorozatokat. A nem megfelelő formában vagy helyen

kiadott kezelői utasitásokat hibásnak minősiti, hibajelzést ad és - az aktuális vezérlő input csatornáktól függetlenül - lehetőséget ad a kezelőnek a konzolirógép segitségével történő javitásra.

Az operátor program kezeli az egyszintes megszakitási rendszert. Bármely job futása megszakitható, és a megszakitás alatt ujabb job vagy jobok indithatók. A megszakitás törlése történhet a GO ON utasitással - ekkor a megszakitott job futása folytatódik, vagy történhet a STOP utasitással - ekkor a megszakitott job törlődik. A megszakitás állapotban inditott jobok is megszakithatók, azonban ezek folytatására a továbbiakban már nincsen mód.

A teljes utasitásrendszer az operátor program saját utasitásaiból és az executive programok utasitásaiból áll. A saját utasitásokat az operátor program mindig értelmezi. Ezek közé tartoznak többek között a vezérlő csatorna váltására, a megszakitások vezérlésére és az executive programok lehivására szolgáló utasitások. Valamely executive program utasitásait az operátor program csak az adott executive lehivása után értelmezi.

Az egységes i/o rendszer tartalmazza a PT, PC, TW és MT input, valamint a PT, LP, TW és MT output rutinokat. Az információ belső tárolása és kezelése - a külső input és output kódoktól függetlenül - 64 alfanumerikus jel ábrázolását biztositó 6 bites belső kóddal történik. Az input rutinok bizonyos adatelőkészitő berendezéseknél előforduló "kettős jelek" komprimálását - 6 bites kóddá alakitását - is elvégzik.

Az operátor program egyidőben, egymástól függetlenül, egy vezérlő és egy információs i/o csatorna-párt kezel. Az információs és a vezérlő input-csatornák szétválasztása szekvenciális. Alaphelyzetben a vezérlőcsatorna-párt a TW/TW rutinok, az információs csatorna-párt pedig - a korábbi gyakorlatnak megfelelően - a PT/LP rutinok alkotják A PC/TW vagy MT/TW vezérlőcsatorna választással lyukkártyáról vagy mágnesszalagról vezérelt, irógépen "bizonylatolt" folytonos "batch" rendszerü üzem is megvalósitható.

MÁGNESLEMEZ OPERÁCIÓS RENDSZER FEJLESZTÉSE A FACOM-R KISSZÁMITÓGÉPRE

VARRÓ LÁSZLÓ

Budapesti Müszaki Egyetem Hiradástechnikai Elektronika Intézet

1. Bevezetés

A Budapesti Müszaki Egyetem Hiradástechnikai Elektronika Intézetében 1971 őszén üzembe állitottak egy japán, FUJITSU gyártmányu FACOM-R kisszámitógépet. A központi egység a 16 kbyte-os memóriával és a 6 /usec-os müveletvégzési idővel a hazánkban ismert kis gépeknek /VT-1010B; TPA/ felel meg. A géphez a következő perifériák csatlakoznak:

- irógép, lassu lyukszalagolvasóval és lyukasztóval /20 kar/sec/
- lyukszalagolvasó /200 kar/sec/
- lyukszalaglyukasztó /50 kar/sec/
- lyukkártyaolvasó /90 kártya/perc/
- lyukkártyalyukasztó /30 kártya/perc/
- sornyomtató /120 sor/perc/
- mágneslemez /128 kbyte; 50 kbyte/sec/
- iker mágnesszalag /800 bpi; 9 sáv; 20 kbyte/sec/

A gép napi egy müszakban a hiradástechnika szakos villamosmérnök hallgatók hardware és software képzését szolgálja. Ez a cikk a hallgatók által diplomatervként, illetve diákköri munkaként készitett összefüggő programrendszerről számol be.

2. A gép lyukszalag operációs rendszere

A gyártó cég által a géphez szállitott lyukszalag operációs rendszer a következő programokat, illetve programcsomagokat tartalmazza:

- FASP nevü assembler forditóprogram, amely lyukszalagról két menetben fordit, és relokálható tárgyprogramot szolgáltat lyukszalagon.
- Relativcimes betöltő, amely a forditóprogram által szolgáltatott tárgyszalagot a memória bármely részére képes bevinni. Ez a program tartalmazza a megszakitás analizátor programot, amely a perifériák és a központi egység programmegszakitásait kezeli.
- Segédprogramok:
  - forrásszalag javitó program
  - forrásnyelvre visszafordító program
  - memória kiiratás sornyomtatóra
  - memória kiiratás lyukszalagra
  - memória kiiratás lyukkártyára
  - mágneslemez kiiratás sornyomtatóra
  - mágneslemez inicializáló program
  - stb.
- Szubrutinok:
  - fixpontos müveletek
  - lebegőpontos müveletek
  - lebegőpontos függvények
  - periféria működtető szubrutinok
  - kód és formátum konvertáló szubrutinok
- Egyéb programok:
  - MINI FORTRAN fordító és végrehajtó program
  - CALCULATOR program /beszélgető program, amely azonnal szolgáltatja az eredményeket/

3. A mágneslemez operációs rendszer

A mágneslemez operációs rendszert a meglévő lyukszalag operációs rendszer és a gép konfigurációjának figyelembevételével alakitottuk ki. A munka lényege egy monitor program készitése a FASP fordítóprogram és a relativcimes betöltő átirása, s néhány igen fontos segédprogram és szubrutin elkészitése volt. A géppel kapott többi programot csak illesztettük a rendszerünkhöz.

3.1. A monitor program

A monitor program egy a memória utolsó rekeszeiben elhelyezkedő rezidens részből és átlapolásos módszerrel müködő fázisokból áll. Az egyes fázisok a mágneslemezen vannak. A monitort önálló programként irógépről hivhatjuk és vehetjük igénybe szolgáltatásait, de hivhatjuk programból is, igy módot ad például egy program egyszerü átlapolásos - overlay-futtatására.

A monitor program funkciói:
- file generálás /felvétel a mágneslemezen lévő könyvtárba/
- törlés a könyvtárból /egy könyvtárelem törlése, és a könyvtár süritése/
- program behivás a könyvtárból
- tartalomjegyzék készités
- szabad terület kezdőcimének lekérdezése
- katalógus lekérdezés
- katalógus lap kitöltés /egy más program által előre felvitt könyvtárelem esetén/

3.2. A FASP fordító program

Ez a fordítóprogram annyiban különbözik a lyukszalagos rendszer fordítóprogramjától, hogy több input-output lehe-

tősége van, és második menetben a mágneslemezről veszi a forrásprogramot. A forrásprogramot veheti lassu és gyors lyukszalagolvasóról, lyukkártyáról, mágnesszalag adott nevü file-jából, mágneslemez adott nevü file-jából. A tárgyprogram készülhet lassu és gyors lyukszalaglyukasztón, lyukkártyalyukasztón és mágneslemezen.

3.3. Relativcimes betöltő

A tárgyprogramokat beviszi a memória kijelölt részére és végrehajtható programokká füzi össze őket. A tárgyprogramot lyukszalagról, lyukkártyáról és mágneslemezről veheti. Mód van egy bevitel során különböző perifériák használatára, például az összes rendszer szubrutin a könyvtárban egy elemként /SUBR/ van tárolva.

3.4. Felhasználói segédprogramok

A meglévő programokat átalakitottuk ugy, hogy a monitor hozza be, inditja el őket, müködés végén pedig a monitornak adják át a vezérlést. Néhány uj, nagyon fontos segédprogramot készitettünk, ezek a következők:

- Nyomkövető program. Az irógéppel megadható ellenőrzési pontokon kiiródik az összes általános regiszter tartalma, valamint kivánságra néhány összefüggő memóriaterület. Futás közben bármikor megszakitható a nyomozás, és ujra definiálhatók a nyomkövetési helyek és a kiiratandó területek.
- Kivitel mágnesszalagra - behozás mágnesszalagról. E két program lehetőséget ad a teljes memóriának adott névvel a mágnesszalagra irására, s onnan történő behozására. Igy tároljuk a ritkábban használt futtatásra kész prog-

ramokat

- Mágnesszalag kiszolgáló programok. Ezek a mágnesszalagok kényelmes használatát segitik elő. Mágnesszalag inicializáló, mágnesszalag másoló, mágnesszalag pozicionáló, mágnesszalag kiirató programok készültek.
- Forrásprogram javitó program. Az input perifériáról az output perifériára viszi a forrásprogramot, miközben módot ad sorok beszurására, törlésére, cseréjére. Input- és output perifériaként lyukszalag, lyukkártya, mágnesszalag és mágneslemez használható.
- Rendszer generátor program. Ennek a programnak a segitségével az összes rendszer programot tartalmazó MASTER TAPE-ről egy mágneslemez tartalom állitható össze ugy, hogy a felhasználó azokat a programokat veheti fel a konyvtárba, amelyekre szüksége van.

3.5. Szubrutinok

A meglévő szubrutinokon tul néhány uj szubrutin könnyiti a felhasználók munkáját:

- BCD aritmetika /a gyártó cég szállitotta/
- Általános perifériakezelő szubrutin. Az irógép, lyukszalag, lyukkártya perifériák közül három együttes müködésére ad lehetőséget.
- Mágnesszalag kezelő szubrutin. Az IBM kompatibilis gépeken használatos szabványos mágnesszalag file-ok kezelését teszi lehetővé, logikai szinten /OPEN-GET-PUT-CLOSE hivások/.

Befejezésül köszönetet mondok SARKADI NAGY ISTVÁN, SZLANKÓ JÁNOS és BITE PÁL, volt hallgatóinknak, szorgalmas munkájukért, amellyel résztvettek a rendszer kialakitásában.

A VT 1010/B OPERÁCIÓS RENDSZERE: A VIDOS

Dömölki Bálint - Havass Miklós

Történeti áttekintés

Az INFELOR és a KFKI Számitástechnikai Osztálya 1971 tavaszán megbizást kaptak a VT 1010/B elektronikus számológépre egy olyan programozási rendszer kidolgozására, amely az alábbi tulajdonságokkal rendelkezik:

- elsősorban az adatfeldolgozási feladatok megoldását biztositja, de figyelembe veszi kisebb igényü müszaki-tudományos feladatok megoldásának lehetőségét is.

- moduláris felépitése biztositja azt, hogy egyéb feladatokhoz készülő program-modulok a rendszer egy korlátozott változatához különösebb nehézség nélkül legyenek illeszthetők.

- egy megszabott összetételü alapkonfigurációra készül, amit azt jelenti, hogy a rendszer ezen müködik optimálisan, de az általa készitett programok a kisebb (ill. nagyobb) konfigurációkon is használhatók.

A feladat tervezéséhez 1971 áprilisában láttunk hozzá. Kezdetben egy 10 főnyi csoport a kisgépek software-jét tanulmányozta, s következtetéseket

vont le a tapasztalatokból a készitendő program-rendszerre vonatkozóan. Májusban kezdtünk hozzá a rendszerterv kidolgozásához. A rendszerterv - mely a rendszer kiépitését három lépcsőben tüzte ki - alapján szeptemberben inditottuk el a programozást.

A megirt programokat nagyrészt szimulátorokon teszteltük.
A rendszer első változata, a VIDOS-1 (VIdeoton Data Oriented System) f. év. áprilisában készült bemutatásra el. A feladat menetközbeni változása következtében a rendszer kifejlesztésének további két lépcsőjét összevontuk, s ennek megfelelően a rendszer második változatát a VIDOS-2-t f. év decemberére tervezzük megvalósitani.

Korlátozások a gép oldaláról

A fenti célkitüzések megvalósitását megvalósitó rendszer kidolgozásánál alapvetően figyelembe kellett venni a VT 1010/B speciális architekturáját, amelyben főképpen az alábbiak voltak meghatározóak:

- a gép lapszervezésü
- a gépnek csak aritmetikai regiszterei vannak, bázis-, index- és állapotregiszterek nincsenek
- négy cimzési módja van, de tetszés szerinti cim csak indirekt módon (akkumulátoron, vagy speciális lapon keresztül) érhető el.
- a szubrutinhivást szervező (branch and link) utasitás hiányzik,
- az utasitáskészlet byte (8 bit) orientált. Ezen felül a cimkidolgozáskor szó-aritmetika van
- több-szintü megszakitás rendszer van.

A Szubruintkezelő Rendszer

Az előző pontban felsorolt jellemző vonások alapján kimondható, hogy a hardware utasításkészletét általános felhasználói szempontból nézve mikroutasitás rendszernek tekintjük, amely szubrutinok (ha ugy tetszik mikroprogramok) segitségével teszi lehetővé az olyan szintü tevékenységek elvégzését, amelyeket nagyobb gépeken egy-egy utasitás szokott elvégezni. Igy a gép egyes felhasználói területeinek megfelelően kialakitható egy-egy szubrutin készlet, amely tulajdonképpen az adott felhasználási terület céljaira való programozás utasitás-készletének is tekinthető. Az igy megalkotott utasitások segitségével megirt program a gépben a megfelelő szubrutinokat aktiváló hivósorokból és a hozzájuk tartozó paraméter-sorokból áll.

Miután a hardware utasitás-készlet nem tartalmaz szubrutinhivást szervező utasitást, kézenfekvő volt, hogy a szubrutinok hivását egy "értelmező rendszer", az SKR (Szubrutin Kezelő Rendszer) végezze.

A felhasználó az esetek tulnyomó részében értelmező szinten fordul a géphez. Ilyenkor egy-egy szubrutin nem-csak gépikódu utasitásokból álló mikroprogram lehet, hanem egy értelmező-nyelven megirt utasitássorozat, sőt esetleg egy másik értelmező-nyelven megirt program részlet. Ilyenkor az SKR -nek egyrészt biztositani kellett az uj értelmezési módra való áttérést. Másrészt a visszatérés megszervezése érdekében biztositani kellett annak megjegyzését, hogy milyen értelmezési módról tértünk át. Harmadrészt biztositani kellett az információcserét - paraméterek

átadása - a hivó és hivott programrész között.

Az egyik szintről a másikra való áttéréssel kapcsolatban gyakran merül fel az egymásba skatulyázott szubrutin hivások igénye, ami indokolttá tette a hivó programrészekre vonatkozó információk megörzésével kapcsolatban a vermelési technika alkalmazását. Miután azonban a hivott szubrutinok az átadott információt csak akkor használják fel, ha gépikódban iródtak, ezért - a felhasználó számára előnyösen - azt a megoldást választottuk, hogy ha a hivott szubrutin értelmező szintü, akkor a paramétereket a verembe az SKR helyezi át, és csak a gépikód-szintü szubrutin fog valójában hozzáférni az átadott paraméterekhez, megtartva annak lehetőségét, hogy a hivó felülbirálhassa a hivott szubrutinban definiált paramétereket.

A megvalósitott programrendszer

Az SKR -rel kiegészitett gépre a következő programkészlet létrehozását tüztük ki.

A futtatandó felhasználói programok felügyeletét a supervisor látja el, amely biztositja mindazon funkciókat, amelyek - programozó számára a rendszer használatához segitséget nyujtanak.

Igy biztositja:
- az SKR igénybevételét
- vezérli az operátorral való kommunikációt.
- Szervezi a tárkezelést olymódon, hogy az adott tevékenységek ellátásához szükséges programok mindig bent legyenek a rendszer rezidens területen,

vagy bekerüljenek a háttértárról (mágnes lemez) a rendszer tranziens területére.

A háttértár igénybevételét és karbantartását a Könyvtárkezelő Rendszer végzi. A rendszer a mágneslemezt öt könyvtárra osztja fel. A könyvtárak közötti információcserét, valamint karbantartásukat a könyvtárkezelő végzi el.

A tárolt információk ábrázolásában a hagyományos file strukturát követtük.

A rendszerhez program-szegmenseket fordító assembler, ezeket programmá összeállító loader, FORTRAN fordító és egyéb programok csatlakoznak.

Az assembler célja, hogy mind a gépkód-szintü, mind az értelmező szintü programozást segitse, az utóbbit különböző célu extrakódok hozzácsatolásával.

A VIDOS egyes komponenseinek részletesebb ismertetésére a konferencia más anyagaiban kerül sor.

A VT 1010/B ELEKTRONIKUS SZÁMITÓGÉPRE IRT VIDOS SUPERVISORA

Simon István
INFELOR

BEVEZETÉS

Intézetünk a VIDEOTON FEJLESZTÉSI INTÉZET megbizása alapján elkészitett egy operációs rendszert az általuk gyártott VT 1010/B elektronikus számitógépekre. A rendszer neve VIDOS /VIdeoton Data Oriented System/
Előadásom e rendszer supervisorának ismertetésével foglalkozik.

A Supervisor általános ismertetése

A Supervisor programok, szubrutinok, táblázatok, és könyvtárak összefüggő rendszere.
A Supervisor, a VIDOS részeként, teljes egészében a gép fix mágneslemez háttér tárján helyezkedik el. E mágneslemeztár /minidiszk/ kapacitása 400-800K byte. A rendszer aktivizálásakor a supervisornak csak egy része kerül az operativtárba. A supervisornak ez a része a rezidens területen helyezkedik el. E terület nagysága függ a géphez illesztett perifériális berendezések számától. A rezidens terület hossza 2,5-4K byte között változhat. /A belső tár kapacitása 4K-32K byte-os modulonként, a rendszer legalább 16K byte nagyságu belső tárt tételez fel./
A supervisor a rezidens területén felül még további egy kilóbyte területet köt le a belső tárból. Ez a supervisor tranziens területe.

A tranziens terület segitségével a supervisor méretét a belső tár kapacitásának sokszorosává képes növelni. A rendszer müködése során mindig azt a programot vagy szubrutint hozza ide be és inditja automatikusan, amelyre éppen szükség van. Igy lehetővé tettük, hogy a rendszer viszonylag kevés helyet

foglaljon el. Ez kis gépeknél rendkivül fontos, mert igy a felhasználói programok egyrészt hosszabbak lehetnek, illetve nagyobb munkaterülettel rendelkezhetnek, másrészt a rendszer belső szolgáltatásu szubrutinjainak igénybevételével, amelyek nem terhelik a felhasználó tár területét, további programbővitést érhetnek el.

3. A supervisor feladatkörei

A supervisor feladatai két nagy csoportba oszthatók:
- a futó programok felügyelete és kiszolgálása
- kapcsolat a gép és kezelője között.

Az első csoportba három szubrutin rendszer tartozik:
- szubrutin-kezelő rendszer /SKR/
- fizikai input/output rendszer /FIOCS/
- belső hivásu szubrutinok rendszere

A szubrutin-kezelő rendszerrel ezen előadás 4. pontja a FIOCS-sel pedig egy külön előadás foglalkozik. A belsőhivásu rutinok széles skálájáról röviden annyit jegyzek csak meg, hogy ezek végzik többek között a könyvtárakkal kapcsolatos alapfunkciókat, a konzolirógép kezelését, az overlay-kezelést, konvertálásokat stb.
A gép és kezelője közötti kapcsolatot ezen előadás 5. pontja tárgyalja.

4. Szubrutin-kezelő rendszer /SKR/

Az SKR alapvető feladata, hogy a VIDOS előirásai szerint megirt programok felett felügyeljen, ill. azokat kiszolgálja.

Az SKR lehetőséget ad különböző módon értelmezendő szubrutinok egységes kezelésére, biztositva eközben
- a szubrutinokból való visszatérések és értelmezési módok vezérlését többszintü hivási lánc esetén is.
- a paraméterek átvételének automatizálását, megengedve a paraméter jellegének meghatározását mind a hivott szubrutinokból, mind a hivási helyről.

4.1. Programok és szubrutinok szerkezete

Minden VIDOS-ban irt program a végrehajtást tekintve két részből áll:
- a feldolgozás sorrendjét irányitó főprogram részből és
- a feldolgozást végrehajtó szubrutinokból.

A főprogram szubrutin hivások sorozata. Egy szubrutinhivás a szubrutin címének és aktuális paramétereinek megadásával történik.

A szubrutinok maguk irják elő értelmezési módjukat, paramétereik számát és tipusát.
Az SKR-ben három alapvető értelmezés van:
- gépikódu szubrutinok értelmezése
- szubrutinhivásos /normál/ értelmezés
- ugrótáblás értelmezés

Ezen felül a rendszer lehetővé teszi, hogy az SKR-be egyéb céloknak megfelelő, tetszőleges értelmezési módokat épithessenek be.
Az SKR nem tesz különbséget program és szubrutin között. Ezért a programok és szubrutinok szerkezetileg megegyeznek. Minden szubrutin három fő részből áll:
- szubrutin fej
- szubrutin törzs
- szubrutin logikai vége.

A szubrutin fej egy vagy több parancsszóból /S direktiva/ áll. A szubrutin ezekkel utasitja az SKR-t, egyrészt, hogy az az aktuális paramétereit milyennek tekintse és azokat hová helyezze, másrészt pedig bejelenti értelmezési módját.
A szubrutin törzsét a fejben definiált szintü utasitások sorozata alkotja.
Gépikódu értelmezés esetén az utasitások megegyeznek a hardware elemi utasitás-készletével.
Normál értelmezésnél az utasitás egy szubrutinhivásból és az aktuális paraméterek felsorolásából áll.
Az aktuális paramétereket a hivási helyen T direktivákkal felül lehet birálni. T direktivák segitségével lehet hivatkozni az előző hivási hely k-adik paraméterére, ill. indirekt cimzés és indexelés valósitható meg. Az indexelést különösen kiemelem, mert a gép nem rendelkezik index regiszterrel, igy az indexelés csak software eszközökkel oldható meg.
A szubrutintörzs bármely helyén ki lehet a szubrutinból lépni. Ezeket a kilépési pontokat a szubrutin logikai végének nevezzük. Minden szubrutinnak legalább egy, de tetszőlegesen sok logikai vége lehet.
A szubrutinok kilépési helyein maguk dönthetnek arról, hogy a soronkövetkező hivási helyre vagy a hivások természetes sorrendjét megváltoztatva a főprogram tetszőleges helyére térjen vissza. Ezáltal az elágazások rugalmas szervezése valósitható meg.

5. Kapcsolat a gép és kezelője között

A supervisor egy figyelő rutinjának /ATTN/ segitségével a konzolirógépen keresztül állandó kapcsolatot tart fenn a gép kezelőjével. Ez a kapcsolat párbeszédes formáju. Az operátor előirt szintaktikáju parancsot gépel-

het be. Az ATTN ezután a könyvtárak belső szolgáltatásához fordul megkeresteti az aktuális parancsot végrehajtó program katalógus bejegyzését, melynek alapján a belső tárba hozza és elinditja. Az operátori parancs végrehajtása után az ATTN egy ujabb parancs fogadására kész, melyet READY üzenettel jelez. Ha valamelyik operátori parancs szintaktikusan hibás, vagy az aktuális program müködése során valamilyen hibát észlel, azt megfelelő hibaüzenettel a konzolirógépen jelzi. Az operátori parancsokat itt az előadás rövidsége miatt nem részletezem, csupán azok egyes csoportjait vázolom.

- forrásnyelvü programok forditását végző programok aktivizálása. /VIDAS, FORTRAN, ASTROL/
- tárgyprogramok kiirását, ill. betöltését végző szolgáltatások /DUMP, LOAD, LOAD1, LOAD2, ez utóbbi relokálható szegmenseket szerkeszt össze és tölt a tárba/
- tárban lévő programok javitására szolgálnak a PDUMP és ALTER operátori parancsok
- a belső tár és a lemeztár között átvitelt végeznek az OVIN1 és OVOU1 operátori parancsok
- felhasználói programok inditását assembly nyelvü programok esetén az EXEC, Fortran tárgyprogramok esetén az FTNEX végzi
- a rendszernek információkat közölnek a DATE, JOB, és OPTION operátori parancsok
- könyvtár szolgáltatást végeznek a CTLST, DELET és TRPLT operátori parancsok.

Az előadás rövidsége miatt a supervisorról csak igen vázlatos képet adhattam. A mélyebben érdeklődőknek ajánlom a VIDOS 8-F jelü felhasználói kézikönyvet. E kézikönyv a VIDOS-szal foglalkozó INFELOR kiadványok keretében jelent meg.

AZ MTA KFKI SZÁMOLÓKÖZPONTJÁBAN MÜKÖDÖ Y OPERÁCIÓS RENDSZER ÉS ANNAK MEGVALÓSITÁSÁHOZ SZÜKSÉGES VÁLTOZTATÁSOK AZ EXECUTIVE VEZÉRLÖPROGRAMBAN

Ivanyos Lajosné
MTA Központi Fizikai Kutató Intézet

1. Bevezetés

A KFKI Számitástechnikai Osztálya egy ICL 1905-ös közepes teljesitményü és egy TPA 1001 kisteljesitményü számológépből álló rendszert üzemeltet.

Az előadás olyan több éves munkáról számol be, amelyben kisebb-nagyobb mértékben a Számitástechnikai Osztály dolgozóinak nagy része résztvett.

Az ICL 1905-ös számológépet az ICL cég 1966-ban helyezte üzembe, 32K központi memóriával, 1 sornyomtatoval, 2-2 lyukszalag olvasóval, ill. lyukasztóval. 1967-ben a konfiguráció 1 db, 4 deckből és 1 db 2 deckből álló magnesszalagos egységgel, valamint egy kártyaolvasóval bővült. A TPA 1001 1969 óta üzemel a számolóközpontban, s a lyukszalagos perifériális berendezéseken kivül rajzológép és egyéb adatátviteli berendezések is illeszkednek hozzá, és 1970-re a két számológép összekapcsolásának hardware és software munkái is befejeződtek.

A felhasználók kutatók, akiknek nagy része saját maga készit az ICT 1905-ös gépre ALGOL és FORTRAN programokat problémái megoldására, igy az ICT 1905-ös gép gépidejének jelentős része kipróbálás alatt álló programokra fordítódik. Az évek során a számológép használatára vonatkozó igények állandóan növekedtek, s azok kielégitésére hamarosan csak egy mód volt: a számológép jobb kihasználásának megvalósitása. A tapasztalat azt mutatta, hogy a már kipróbált programok futtatásánál nem jelentékeny a veszteség-idő, a kipróbá-

lás alatt álló programok kezelésénél már igen. A veszteség időnek lényegében két oka van:

a) a számológépen futó program perifériális átvitelre várakozik,
b) a program gépkezelői beavatkozásra várakozik.

A probléma megoldása: a gépkezelői beavatkozások számának csökkentése, valamint a futás közbeni lassu perifériális átvitel kiküszöbölése, vagyis operációs rendszer használata.

Miért saját operációs rendszer?

Az 1900-as számológépekre manapság rendelkezésre álló operációs rendszereket és egyéb segédprogramokat nem használhattuk részben azért, mert még nem készültek el, részben pedig azért, mert helyi sajátosságainkhoz nem alkalmazkodtak eléggé. Mivel közvetlen hozzáférésü háttérmemóriánk nincs, az ICL mágnesszalagra orientált operációs rendszerét használnatnánk. Kipróbálásakor azonban ugy láttuk, hogy egyrészt komoly veszteség-idővel kell számolnunk a mágnesszalagos mozgatások miatt, másrészt pedig a felhasználók rendelkezésére álló memória csökken, s ez erzékenyen érintette volna felhasználóinkat, hiszen akkor pl. ALGOL programok még egyáltalán nem használhattak OVERLAY technikát.

Olyan operációs rendszert kellett megvalósitani, amely messzemenően alkalmazkodik a KFKI számolóközpontjában folyó munka adottságaihoz. Igy készült el az Y operációs rendszer a kipróbálás alatt álló programok kezelésére. A különböző operációs rendszer modulok önálló programok, amelyek elvégzik, ill. vezérlik a szükséges müveleteket, s mivel viszonylag kis helyfoglalásuak, elősegitik a multiprogramozás lehetőségeinek kihasználását.

Az Y operációs rendszer felépitése

A rendszer által kezelt programokat mágnesszalagon tároljuk forrásnyelvü formában, s mivel a mágnesszalagon tárolt információk gazdaságosan sorozatban dolgozhatók fel, ezek javitása, majd forditása és

végül futtatása sorozatban történik a rendszer különböző moduljainak vezérlésével. Az Y rendszer a sornyomtató és a graph plotter mágnesszalagon való szimulálásával kapcsolatos feladatokat is megoldja. Az eredmények kiiratását külön rendszer modul végzi, a rajz információk esetén a TPA kisszámológéppel együttmüködve, szintén sorozatban. Ugyancsak a TPA-ICT kettős rendszer használatos a rendszer és a felhasználói programok input adatainak mágnesszalagra vitelére, hogy az ICT 1905 gépet mentesitsük a lassu perifériális átviteltől.

Az Y operációs rendszer moduljai:

a) Editor modul (YKA1, YKAT, YKA2)
   A forrásnyelvü programok és adatok tárolását, javitását és forditáshoz, illetve adatok esetén futtatáshoz való előkészitését végzi.
b) Sorozat-forditást vezérlő modul (YFRD)
   Trusted program, a compilerek vezérlését, a leforditott programok futtatásra előkészitését végzi. Operátori beavatkozás nélkül dolgozik, a mágnesszalagos mozgatást minimálisra csökkenti.
c) Sorozat futtatást vezérlő modul (YFUT)
   Trusted program, sorozatban elvégzi az előző modul által előkészitett programok futtatását, a gépkezelői tevékenység nagy részét kiküszöböli.
d) ICT-TPA kettős rendszerben adatfelvitelt, illetve kiiratást, rajzolást végző programok (YKTØ, YKTT, YAP2, YDUA)
e) A sornyomtatót szimuláló mágnesszalag tartalmát kiirató program (YLPO)

4. Az EXECUTIVE vezérlőprogram módositásai

Az ICT 1905-ös számológépeken az EXECUTIVE program oldja meg a multiprogramozással kapcsolatos problémákat, a perifériális egységek kezelését, konzol üzenetek formájában tartja a kétoldalu kapcsolatot a gépkezelőkkel, s extrakódok formájában a programok számára nyujt szolgáltatásokat.

Ezen a programon biznyos módositásokat, bővitéseket végeztünk; a számológéphez kapcsolt uj berendezések (mill timer és TPA 1001) szubrutin csomagjait hozzáillesztettük az EXECUTIVE-hoz, növeltük a programok, ill. gépkezelők számára nyujtott szolgáltatásokat. A módositások nagy részét az Y operációs rendszer használja, a többi rész a gépkezelői és az adminisztrativ munka megkönnyitésére készült.

A módositások a következők:

a) Szolgáltatások a GIVE extrakód kiterjesztésével:
   1. a futó program neve
   2. a futó programra számlázott idő (mill time)
   3. a szabad memória kapacitás
   4. az EXECUTIVE bevitele óta eltelt idő (binárisan)

b) Mill-timer package
   A multiprogramozásu üzemmódban a számlazáshoz szükséges óra, az un. mill timer a KFKI-ban készült, némileg eltér a standard ICL mill timer-től. A használatához készült rutincsomag nyujtja a számlázáshoz szükséges információkat is.

c) Az előzőhöz kapcsolódik a számlázás adminisztrativ részének megkönnyitésére készült módositás, amely azt a célt szolgálja, hogy a számlazáshoz szükséges információról azonnal lyukszalag készüljön.

d) Az előző két módositáshoz tartozó konzol üzenetek:

ÜZENET	HATÁS
MI# programnév	- a programra számlázott idő kiiródik a konzolon
EF	- kiiródik az EXEC betöltése óta eltelt idő és a programokra számlázott idő
TP8 vagy TP9	- a számlázási információk lyukasztása a 8-as vagy a 9-es számu szalaglyukasztón megkezdődik

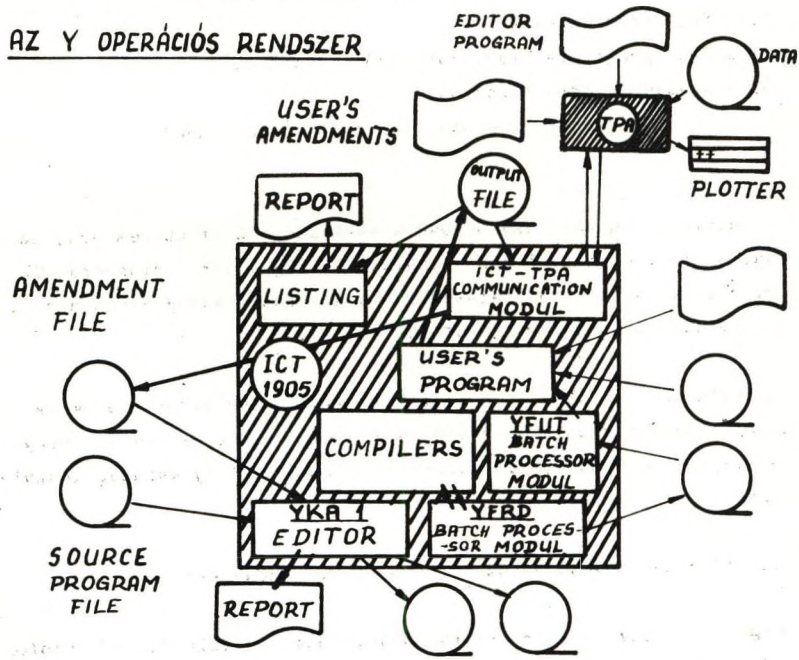
TPO - a számlázási információk lyukasztása megszünik

SZ - az utána kiirt üzenet a számla-szalagra kerül

e) TPA package

TPA 1001, mint az ICT standard interface-s perifériája kezeléséhez szükséges szubrutincsomag. A két gépből álló számológéprendszer alapja.

f) Programnév kiirás a sornyomtatón a kiirt eredmények szétválogatásának megkönnyitésére.



A rendszer megtervezésében és megvalósitásában nagyobb volumenü munkát a következő kutatók végeztek: Ivanyos Lajosné, Krammer Gergely, Lovas Istvánné, Varga László.

AZ ICL 1900-AS SOROZATU SZÁMOLÓGÉPEK OPERÁCIÓS RENDSZEREINEK ÉRTÉKELÉSE ÜZEMELTETÉSI TAPASZTALATOK ALAPJÁN

Kertész Miklós
NIM IGÜSZI Számológközpont
Software-fejlesztési Osztály

Egy elektronikus számológép üzemeltetését kiszolgáló software-rendszer alapja az alkalmazott operációs rendszer. /továbbiakban: OR/ Az installálás előtt lényeges az OR megismerése, mert az OR által nem biztositott software rendszerelemeket létre kell hozni. Különös gondossággal kell eljárni az OR kiválasztása szempontjából abban a esetben, ha a számológépet gyártó cég több OR-t biztosit.

Az ICL az 1900 sorozatu számológépeire 4 OR-t alkalmazott, amelyeket GEORGE-nak /GENERAL ORGANIZATIONAL ENVIROMENT/ neveznek. A GEORGE 1&2 a batch processing lehetőségeit biztositja, a GEORGE 3&4-ben a multi access /time-sharing/ valósitható meg.

A GEORGE 1 OR-t - amely egy program - a gép executive-ja a "bizalmas" státuszáról ismeri fel, és engedélyezi számára, hogy programokat betölthessen, futtathasson, ill. futásukat ellenőrizhesse. Az ellenőrzés az extrakódok felülbirálásának joga - kivéve a már lekötött perifériákra történő átvitelt - valamint a számológépbe épitett óra /megszakitójel/ alapján a programok memóriában való aktiv tartózkodásának állandó vizsgálata. A GEORGE 1 lehetőséget teremt a lassu perifériákon történő adatfolyamok dokumentumnév-, valamint bizonyos mágnesszalagok fejcimke ellenőrzésére /opcionális/. Az OR funkcióit a programozó vezérli a jobleirás segitségével: a program eseményeit figyeltetve - az operátor beavatkozását mellőzve - továbbindithat, törölhet, uj programot tölthet be /programbetöltő utasitást felülbirálhat!/

Ezzel a GEORGE 1 megadja a lehetőséget arra, hogy egymásra nem hivatkozó /vagy másként hivatkozó/ programokat programrendszerként futtathassunk.

A GEORGE 2 OR tartalmazza a GEORGE 1 lehetőségeit, kibővül az input-output off-line technikával /opcionális/, valamint a multiplexer csatornán keresztül kommunikáló távállomások adatainak kezelésével. Az off-line adatfolyam - csak lassu perifériákra hivatkozó - mágnesszalag ill. mágneslemez közbeiktatásával kerül feldolgozásra. Az input-output programok és a központi modul egymásutániságát nézve megkülönböztethető folyamatos - csak mágneslemezes rendszernél - és nem folyamatos üzemmód. A folyamatos üzemmód biztositja a lassu perifériák maximális sebességü használatát, az input-output programok egyidejü futtatását a központi modullal, valamint annak folyamatos munkáját.

A GEORGE 3 OR az előzőektől eltérő szerkezetü. Saját executive-val rendelkezik. A lassu perifériákat csak off-line üzemmódban használja - mágneslemezen, ill. mágnesdobon keresztül - ezzel lehetővé teszi olyan bináris programok futtatását is, amelyek az adott konfigurációtól eltérő periféria-tipusokon keresztül kommunikálnak! Mágnesszalagok és mágneslemezek használhatók mind on-line /csak ellenőrzéssel/, mind off-line /szimulált is!/ üzemmódban. A jobleiráson keresztül vezérelhető több job egyidejü futtatása az eseményektől függően, de az eseményt szolgáltató job törlése nélkül! A multiplexer csatornára kötött távállomások jobleirást is adhatnak, közvetlen háttér hozzáféréssel.

A GEORGE 4 OR a GEORGE 3 alkalmazása lapozási technikával rendelkező /pageing/ számológépekre. A rendszerrel kapcsolatos üzemeltetési tapasztalatokkal nem rendelkezünk.

A rendszerek rövid ismertetése után vizsgáljuk meg az összehasonlításukat elősegitő paramétereket, amelyeket két csoportba oszthatunk:

1/ az OR által nyujtott szolgáltatásokat irják le
/hivatkozva a bevezetőben emlitett rendszerelemek problémájára/

2/ az OR hatékonyságát irják le
/adminisztrációs idők, OR-executive kapcsolat/.

Az első csoportba a költségvetés, számlázás, ütemezés és háttérmemoria-kezelés megszervezését soroljuk.

A költségvetés a bérmunkát is végző számolóközpontoknál lényeges: a számolóközpont vezetője által jóváhagyott témakódra kiutalt keretösszeg tulhaladását /esetleg egyes futásokra megállapitott időkorlátokat/ ellenőrzi.

A GEORGE 1&2 csak időellenőrzést képes ellátni - a programozó ezt a jobleirásban figyeltetheti is - a keretösszeg ellenőrzést segédprogrammal kell végeztetni.

A GEORGE 3&4 a témakód szerinti ellenőrzést, a job-idő, és a program-idő ellenőrzést automatikusan elvégzi. A témakódok, munkák költségvetése a konzolon is lekérdezhető.

A számlázás a gép használatának bizonylata, a napi-, heti-, vagy havi elszámolás alapja.

A GEORGE 1 jobonként gyüjti a kezdés és befejezés időpontját, valamint a használt processzor időt az azonositáshoz szükséges információval együtt.

A GEORGE 2 kiegészíti ezeket az input-output off-line rekordok számával.

Mindkét rendszernél a rendező és nyomtató programot külön kell kidolgozni.

A GEORGE 3&4 a költségvetésre támaszkodó számlarendszerrel rendelkezik, amelyet makróutasitás aktivizál.

Az ütemezés a számológép hatékony kihasználásának alapfeltétele.

- A GEORGE 1&2 rendszer nem rendelkezik automatikus ütemezéssel: e munka a diszpécserekre hárul - ez emberi munka beiktatása egy gépi rendszerbe, de biztositja annak flexibilitását -.

  A GEORGE 3&4 automatikusan ütemez a számológép vezetője által engedélyezett sürgősség kód /urgency code/ a program nagysága, valamint periféria használata szerint. A time-sharing technika biztositja az egyenrangu programok egységes kezelését.

A háttérmemória-kezelés a gyorsperiférиák védelmét hivatott biztositani a multiprogramozott gépeknél.

A GEORGE 1&2 csak mágnesszalagok azonositását teszi lehetővé, amely csak akkor nyujt teljes védelmet, ha az összes felhasználó él a lehetőséggel. Mágneslemezek védelmét az executive által ellenőrzött megőrzési idő és integritási kód biztositja.

A GEORGE 3&4 magasszintü védelmet ad a gyorsperifériák azonositási kötelezettsége, témakódhoz való hozzárendelése, valamint a file-ok szimulálásának lehetőségével.

Egy OR hatékonyságát a program v. job futásakor fellépő adminisztrációs idők nagyságán is mérhetjük. Az executive és az OR együttmüködését vizsgálva azt tapasztaljuk, hogy a GEORGE 1&2 rendszer felügyete alatt futó program akcióit mind az executive, mind az OR adminisztrálja.

A GEORGE 3&4 "saját" executive-val rendelkezik, igy az eseményeket csak egyszer jegyzi-.

A time-sharing technika, az OR utasitásnyelvezetében megengedett összetett makró-hivási lehetőségek, valamint a szimulált perifériák listázása lassitják ugyan a rendszert, de megkönnyitik a használatát.

Az u.n. nullprogram futási ideje, a makrók aktivizálásának ideje összevetve az 1. pontban leirt szolgáltatásokkal megfelelően irányithatják a próbaüzem időszakának helyes kihasználását.

Az összehasonlitást természetesen befolyásolja a gép konfigurációja és felhasználási területe, amelyek az egyes paramétereket sulyozzák.

KISSZÁMOLÓGÉPEKEN ALKALMAZOTT MEGOLDÁSI MÓDSZER PROGRAMOK ÖSSZEÁLLITÁSÁRA ÉS BETÖLTÉSÉRE

Varga László

MTA Központi Fizikai Kutató Intézet

A programokat ma általában modulokból épitjük fel. A modulok olyan névvel ellátott programegységek, amelyeknek egyes utasitásait, adatait más modulban is érvényes névvel látjuk el. Az ilyen programszervezésnek az az előnye, hogy az egyes modulok önállóan megirhatók és kipróbálhatók, majd a program kész modulokból szerkeszthető össze. Ennél a módszernél a forditóprogramok a programmodulokat olyan formára konvertálják, amely még nem rögziti a memóriában a modul helyét, ugyanakkor a forrásnyelvtől független, annál tömörebb ábrázolási forma. Mivel ez a forma a programot bináris alakban ábrázolja, ezért ezt a "relocatable binary" elnevezés alapján röviden r/b formának nevezzük. Azt a programot pedig, amely a modulokból a programot összeszerkeszti és a memóriába betölti, szerkesztő-betöltő programnak nevezzük.

A tár egyes területei a hozzáférés szempontjából nem egyenértékűek. Ezért a tárat rendszerint mezőkre osztjuk. Egy-egy modul több mezőben igényelhet helyet, és az r/b formával ezekbe a mezőkbe való betöltését vezéreljük.

Az r/b forma legfontosabb utasitásai ma már géptől függetlenül kialakultak és közismertek. Mi ezeknek az utasitásoknak a végrehajtásával kapcsolatos problémákat foglaljuk össze, és ismertetjük azok egy megoldását a TPA-70 kisgép esetében.

A tárat a program betöltése szempontjából két mezőre osztottuk: az egyik mező a csatolólapi mező, a másik a programmező. A csatolólapi mezőbe azok a tárolócimek kerülnek, amelyek segitségével a program

különböző területei között megteremtjük a kapcsolatot, mivel a tároló-referenciás utasitásokkal a megadott bázis környezetében egy lap nagyságu területet foghatunk csak át, és a csatolólap segitségével lényegében ilyen bázisokat adhatunk meg.

Az r/b forma utasitásait ugy választottuk meg, hogy az assembler is, és a szerkesztő-betöltő program is külön-külön egy menetben elláthassa feladatát. A modulok betöltése a csatolólapi mezőbe és a programmezőbe folyamatosan, a megtalálás sorrendjében történik. A modul kezdetének cimét az egyes mezőben bázisnak nevezzük.

Vegyük ezután sorba az egyes utasitásokat.

1. Abszolut adatok betöltésének vezérlése. A program utasitásai, adatai rendszerint folyamatosan töltik ki a memóriát, ezért az abszolut adatokat olyan blokkokba célszerü szervezni, ahol először megadjuk az első betöltendő adat cimét, majd felsoroljuk az egymás után betöltendő adatokat. Az első betöltendő adat cime lehet abszolut szám, vagy valamelyik bázishoz képest relativ adat.

2. Bazishoz képest relativ adatok betöltésének vezérlése . A betöltött abszolut részen lényegében ezzel egy első javitási fázist vezérelünk.

A programban a bázishoz képest relativ adat előfordulási helyén - ha az egy később definiálandó (postdefinált) cimkével képezett kifejezés - az adatnak csak a cimkéhez és nem pedig a bázishoz képest relativ értéke ismeretes. Az adat bázishoz képest relativ értéke csak ott válik ismertté, ahol a cimkét definiáljuk. Felmerül tehát a kérdés, hogy a relativ adat előfordulási helyén milyen utasitást adjon az assembler a betöltő program részére? A következő két megoldás kézenfekvő:

a) A cimkifejezés előfordulási helyén abszolut adatként a cimkéhez képest relativ értéket adja ki az assembler betöltésre és táblázatában megjegyzi, hogy melyik cimhez kell majd hozzáadni a szóbanforgó

cimke értékét. Amikor viszont a cimke értékét az assembler meg tudja állapitani, akkor a következő utasitásokat generálja:

- a betöltési pointert állitsuk a feljegyzett cimre,
- a betöltési pointer által meghatározott értékhez adjuk hozzá a cimke bázistól számitott relativ értékét.

b) A cimkifejezés előfordulási helyén azt adjuk meg, hogy hova kell letenni a szóbanforgó cimkifejezésben szereplő bázissal azonos bázisu előző értéket. Ezt az adott cim és a megelőző cim különbségével célszerü megadni. Ilyen módon az azonos bázisu relativ adatok helyeit egy pointer lánccal összefüzzük. A legelső helyet zérussal jelölhetjük meg. Az adatok betöltését ekkor a következőképpen vezérelhetjük:

- megadjuk a bázist, amely mellett képeztük a láncot,
- megadjuk a lánc felgöngyölitéséhez szükséges indulási cimet, a bázishoz képest relativ formában,
- felsoroljuk rendre a betöltendő adatoknak a bázistól számitott értékeit.

Ennél az utóbbi módszernél egy-egy betöltést lényegében egyetlen ujabb adattal vezérelünk, mig a másik módszernél minden egyes relativ adat betöltéséhez három ujabb paramétert kell megadni. Programunkban az utóbbi megoldást választottuk.

3. Névvel ellátott bázisu adatok betöltése, a programmodulok összekapcsolása. Ez a javitás második fázisa. A programmodul kitüntetett utasitásait, adatait a programozó névvel látja el. Ezek globális nevek, amelyekre a programozó más modulban is hivatkozhat, ott kifejezéseket képezhet velük.

A globális névvel képezett kifejezés előfordulási helyén az assembler a kifejezésnek csak a globális névhez képest relativ értékét ismeri. Ennél többet a modul forditása során sem tud meg. Mi a követke-

ző megoldást választottuk ezeknek a kifejezéseknek a kezelésére:

- az assembler a kifejezés előfordulási helyén csak a globális névhez képest relativ értéket tölteti be. Az előfordulási hely címét a globális név mellett táblázatában rögziti.
- a modul végén kiadja a globális nevet, majd alatta felsorolja azokat a cimeket, amelyeknek tartalmához hozzá kell adni a globális névvel jelölt értéket.

A globális nevekről a szerkesztő-betöltő program táblázatot vezet. Egy név a globális nevek táblázatába két esetben kerülhet fel:

- egy modulban értéket kapott,
- egy modulban felhasználták, kifejezést képeztek vele.

Az első menetben a globális nevek táblázatába értékével együtt helyezhető el, mig a második esetben nem biztos, hogy ismert már az értéke. Az utóbbi esetben viszont nem tudjuk végrehajtani azt, hogy a felsorolt cimek tartalmához adjuk hozzá az adott globális név értékét. Ezeket az utasitásokat tehát kénytelenek vagyunk tárolni addig, amig a globális név értékét nem tudjuk meghatározni. A tárolás lényegében a módositandó cimek felsorolását jelenti.

A szerkesztő-betöltő program szervezése. A program a felsorolt utasitásoknak megfelelően három modulból áll, és a memória végén helyezkedik el. A táblázata, amelyet a betöltés során szükség szerint növel, illetve összehuz, a szerkesztő-betöltő program előtt kap helyet.

Egyéb problémák. A TPA-70-es gépen minden 4K nagyságu memória részhez egy-egy csatolólap tartozik. A csatolólapok a memória elején foglalnak helyet. Ezzel kapcsolatban meg kellett oldani a csatolólap betöltését abban az esetben is, amikor egy modul az egyik 4K nagyságu egységből "átfolyik" a következő egységbe. A most idézett probléma egyszerübb megoldására elkészült a loader két menetes változata is.

LYUKSZALAG ORIENTÁLT PROGRAM RENDSZER TPA/i-1001 SZÁMITÓGÉPRE

Szemző Tamás
ORION RÁDIÓ ÉS VILLAMOSSÁGI VÁLLALAT

1. Bevezetés

Vállalatunknál 1972. február hónapjában üzembe állitottak egy TPA/i-1001 számitógépet. Mint ismeretes, e gép sokoldalú és jól felhasználható software-rel ellátott. Ugyanakkor, a felhasználói és programozói gyakorlat néhány olyan hiányosságra hivta fel a figyelmet, amely problémák megoldására érdemes volt energiát és időt forditani, mert a programozói és operátori munka leegyszerüsitését eredményezte.

A kifogásolt hiányosságok:

A. A LOAD-oláshoz feleslegesen sok operátori beavatkozás szükséges

B. SLANG-1 nyelvü, assembler szintü programok irása esetén, a lapváltásokra túl nagy figyelmet kell forditani.

C. A software nem ad elég segitséget új programok hibás futása esetén a hiba behatárolására.

E leirás, rövid összefoglalása azon software fejlesztési munkának, amely - az alapsoftware-re támaszkodva - az A. és B. pontban emlitett hiányosságokat próbálja kiküszöbölni. A jegyzet egy másik leirása /Gábor Andrásné: Egyszerü hibakeresési módok TPA/i-1001 számitógépen/ a C. pontban emlitettekkel foglalkozik. A leirás a TPA gépcsaládnál használt terminológiát ismertnek tételezi fel.

2. LOAD-olás

A számitógép leszállitását módositott hardware loader-rel kértük. A hagyományos RIM loader helyett egy olyan változat került beépitésre, amely RIM és BIN formátumu lyukszalagok beolvasására egyaránt alkalmas, valamint a loader a 7777_8 cimü

szó betöltése után, HLT-on leáll.
A load-olás automatizálása céljából kidolgoztunk egy rendszert /SLS, Segment Loader System/, amelynek loadere a következő formátumu lyukszalag beolvasására alkalmas: egy gépi szó tartalma a lyukszalagon két egymást követő karakter 1-6 bitje, ahol az első karakter a hetedik csatorna lyukasztásával jelzett.

Az SLS formátumban lyukasztott lyukszalag egy fejrésszel kezdődik, amely a loadernek az alább részletezett információkat nyújtja:

1. gépi szó: a betöltendő FIELD száma
2. " " : a betöltési kezdőcim
3. " " : a betöltendő szavak számának kettes komplemense
4. " " : kiegészitő kód, amely az egész beolvasandó lyukszalag kontrolösszegét 7777_8-re egésziti ki
5. " " : hibátlan betöltés esetén a loader erre a cimre adja át a vezérlést
6. " " : és az ezt követő szavak, a betöltendő információ.

Az SLS Loader a 0 FIELD utolsó lapján helyezkedik el, inditási cime 7777_8. Ha egy program, vagy segmens feladatát elvégezve a vezérlést erre a cimre adja át, ujabb program, vagy szegmens tölthető be és inditható.

Kidolgoztuk azt a lyukszalag lyukasztó programot, amely a SWITCH REGISTERen adott információk alapján előállitja a kivánt SLS formátumu lyukszalagot.
Az ismertetett loaderek /hardware, SLS/ felhasználásával minden

rendszeresen használt lyukszalagunk egy BIN formátumu loaderrel kezdődik, amit SLS formátumu program szalag követ.

Operátori utasitások:

1. Legyen IF, DF nulla
2. CLEAR
3. RIM /beolvassa az SLS loadert/
4. CLEAR /megszünik a RIM módus/
5. CONTINUE /inditva az SLS loader/

és a vezérlés átadódik a futtatni kivánt programnak.

Az ismertetett módszer előnyei a hagyományos beolvasással szemben:

1. Nincs szükség a BIN loader külön betöltésére
2. Lényegesen kevesebb kapcsolás szükséges a vezérlő billentyüzeten
3. Nem kell a program inditási cimét megjegyezni
4. Egyszerre több FIELD betölthető.

3. Interpretiv nyelv

Az alább ismertetendő programozási nyelv, vagy módszer kidolgozása egyidejüleg két - egy programozási és egy szervezési - problémánkra adott használható megoldást. Programozási nehézséget a bevezető B. pontjában emlitett lapváltás jelentett. Szervezési kérdés, a különböző programozók által irt szubrutinok egységesitése, hogy a szubrutin könyvtár könnyen áttekinthető és használható legyen.

Áttérve a nyelv ismertetésére emlékezzünk arra, hogy SLANG-1 nyelven irt programban szubrutin hivás, azoknak paraméter átadás ill. azoktól paraméter átvétel általában csak indirekt cimzést alkalmazva lehetséges. Ez a tény, programozás folyamán fokozott figyelmet igényel, a túl nagy helyfoglalást nem is számitva.

Kidolgozva egy interpretert, lehetővé vált, hogy egy szubrutin sorozat aktivizálása a szubrutin nevek /belépési cimek/ és a hozzájuk tartozó paraméter-azonositók /direkt vagy indirekt cimek/ felsorolásává egyszerüsödjenek.

Az interpreter 13_8 gépi szó és 3_8 szó a 0-ás lapon, tehát helyfoglalása minimális. A SLANG-1 nyelvü programban az interpreter inditása vezérlésátadással kezdődik /JMS 17/, amelyet már az előbb emlitett szubrutin név illetve paraméter sorozat követ. Az interpreter vezérli a szubrutinok sorozatának végrehajtását. A paraméterek mozgatását a szubrutinok végzik.

Az interpreter használatából visszatérni SLANG-1 nyelvü programra EXIT = 0000 zárószóval lehet.

Az interpreter felépitése:

```
0017    0000
0020    5421          JMP I INTER
0021    4677  INTER,   4677
```

```
4677    7300  INTER,  CLA CLL
4700    1017          TAD 17
4701    3311          DCA SUBRC
4702    1711          TAD I SUBRC
4703    7450          SNA
4704    5417          JMP I 17
4705    3311          DCA SUBRC
4706    4711          JMS I SUBRC
4707    2017          ISZ 17
4710    5277          JMP INTER
4711    0000  SUBRC,  0000
```

Az interpretert felhasználó program formája:

```
SLANG-1 nyelvü program
   .
   .
JMS 17
SUBR1 CIM
Paraméter 1
Paraméter 2
   .
   .
Paraméter n
SUBR2 CIM
Paraméter 1
   .
   .
0000
   .
```

EGYSZERŰ HIBAKERESÉSI MÓDOK TPA/i-1001 SZÁMITÓGÉPEN

Gábor Andrásné
ORION RÁDIÓ ÉS VILLAMOSSÁGI VÁLLALAT

Bevezetés

A programozói, software fejlesztési munka egyik legszebb és legnehezebb része az újonnan megirt programok hibáinak felderitése, a helyes müködés ellenőrzése, azaz a program u.n. belövése. A számitógépek programkönyvtárában a "utility" programok között szerepelnek általában a programbelövést, hibakeresést segitő programok.

A TPA számitógép software anyagában is megtalálhatók az ilyen jellegü programok /ODT, DDT, MONITOR/, kezelésük azonban nehézkes, kevés a programok kijelzéseinek információtartalma és csak kisebb volumenü programok hibakereséséhez alkalmazhatók.

Az ORION-ban 1972. februárban felállitott TPA/i-1001 számitógépen a kifejlesztett programrendszerrel /l. Szemző Tamás: Lyukszalagorientált programrendszer TPA/i-1001 számitógépre c. előadása/ együtt a rendszerbe jól illeszkedő hibakeresési módszerek, programok kerültek kidolgozásra. Jelen előadás célja, hogy vázlatos betekintést adjon a hibakeresés ezen módszereibe.

Három, egymást jól kiegészitő, de külön-külön is hatásos program kerül ismertetésre, a

MONITOR /nem azonos a TPA eredeti software anyagához tartozó programmal/, mely az új program futása közben ad a futásról információkat, a

NYOMKÖVETŐ INTERPRETER, mely a belövés alatt álló program futása közben jegyzi az interpreteren keresztül hivott és végrehajtott subrutinokat valamint a

POST MORTEM, mely, mint neve is mutatja, a hibásan lefutott programról ad információt.

1. MONITOR

A MONITOR interpretiv nyomkövető program, mely 8K-s TPA kiépités esetén 4K memóriaigényü, nagyobb kiépités esetén 8K memóriaigényü és programmegszakitást nem használó programok nyomkövetésére készült. Slang-1 nyelven irt nagy volumenü programok hibáinak felderitéséhez, futás közbeni ellenőrzéséhez alkalmazható.

1.1 Müködése

Kiépitéstől függően a MONITOR a TPA számitógép I. vagy II. FIELD-jében futó program, mely a 0. FIELD-ben helyet foglaló és kezdőcimével megadott nyomkövetendő program utasitásainak sorozatát saját adatként kezeli, a nyomkövetendő program futását

Pseudo Program Counter	/PSPC/
Pseudo Memory Buffer	/PSMB/
Pseudo Accumulator	/PSAC/ és
Pseudo Link	/PSL/

memóriarekeszek segitségével szimulálja.

A futás során a PSPC tartalma által definiált memóriacim tartalmát /a nyomkövetendő program soronkövetkező utasitását/ a PSMB-be helyezi és a PSMB tartalma által definiált utasitásnak megfelelően változtatja a PSAC, PSL és PSPC rekeszek tartalmát. Amikor a PSMB tartalma feltétlen vagy feltételes ugrató utasitásnak megfelelő gépi kód, a MONITOR teletype-on, vagy gyorslyukasztón kijelzi az ugrás következtében módosult PSPC tartalmat, az ugrást előidéző PSMB tartalmat valamint a PSAC és PSL ugrás előtti értékét.

A MONITOR számára megadható egy alsó és felső memóriahatár /nagy volumenü, már belőtt szubrutinokat használó programoknál célszerü/. A MONITOR a pseudorekeszek tartalmát csak akkor ir-

ja ki, ha a PSPC rekesznek az ugrás végrehajtását megelőző és az ugrás végrehajtása során módosult tartalma közül legalább az egyik a megadott intervallumon belül esik.

Mivel a MONITOR bármely olyan program futását szimulálni tudja, mely a 0. FIELD-ben fut, adatait a 0. vagy I. FIELD-ben helyezi el és nem használ programmegszakitást, a kiterjesztett memória utasitásokat valamint a programmegszakitást megengedő utasitásokat különlegesen kezeli. A HLT utasitás hatására a TPA számitógép HLT állapotot vesz fel és CONTinue-val indítható tovább.
A MONITOR segitségével a nyomkövetendő programról kapott kijelzés információtartalma nagy és a program logikai hibái könnyen kiderithetők.

2. NYOMKÖVETŐ INTERPRETER

Az ORION-ban kifejlesztett programrendszer egyik sarkpontja az INTERPRETER, mely lapfüggetlen subrutin- és paraméterkezelést tesz lehetővé. A NYOMKÖVETŐ INTERPRETER ennek olyan, néhány utasitással bővitett változata, mely a program futását ugyanúgy szervezi, mint az eredeti INTERPRETER és ezenkivül, a TPA/i számitógép I. FIELD-jének első 128 szavát felhasználva a következő módon végez nyomkövetést:

Az első szóba beirja a futó program azon memóriacimét, ahonnan az INTERPRETER-en keresztüli subrutinhivás történik, a következő szóba 7777_8 kerül. A hivott subrutin végrehajtása után a NYOMKÖVETŐ INTERPRETER a 7777_8-t felülirja a végrehajtott subrutin belépési cimével. Ezt a müveletet a futás során a subrutinok hivásának megfelelő sorrendben végzi, ciklikusan használva

a 128 szót. Ilymódon mindig az utolsó 64 végrehajtott subrutin cime áll rendelkezésre. Hiba esetén a hibás subrutin azonnal felderithető, mert a subrutin belépési cime helyett 7777_8 található a cimjegyzékben.

3. POST MORTEM

Az előzőekben leirt két módszer az ott emlitett korlátok miatt nem alkalmazható általánosan. Ilyen esetekben nyújt segitséget a POST MORTEM program, mely lényegileg az SLS Loader olyan módositott változata, mely egy hibásan lefutott programról tud utólag információt nyújtani. Lényege, hogy a loadolás folyamán a számitógép memóriájának a lefutott program által elfoglalt részének a tartalmát hasonlitja össze az eredeti programot tartalmazó lyukszalaggal. Az eltéréseket /és csak azokat/ teletypeon a következő módon irja ki:

cim:	a cim tartalma:	a cim lyukszalagról beolvasható tartalma:
pl:		
6271	2770	3772
7201	0000	0152

A POST MORTEM segitségével elsősorban az adatok előállitásának helyességéről, azok rendeltetésszerü helyre való beirásáról, flag-ek és pointerek helyes beállitásáról lehet információt kapni.

A jelen módszerek, melyek közül a MONITOR és a nyomkövető INTERPRETER dinamikusan jelzi a futás logikai menetét, a POST MORTEM pedig a programfutás eredményéről ad tájékoztatást, a gyakorlatban jónak bizonyultak, jelentős segitséget nyújtottak sok software és néhány hardware jellegü hiba felderitésénél.

EGY SOFTWAREFEJLESZTÉSRE ORIENTÁLT NYOMKÖVETŐ PROGRAM: NIM MONITOR

dr. Kőszegi György
NIM IGÜSZI Számológközpont
Software-fejlesztési Osztály

A feladat megfogalmazása

Hardware feltételek: második generációs számológép /EMG 830-20/. A központi memória /max 64 K-24 bites szó/ szervezése olyan, hogy két mezőre oszlik, az egyik célszerűen a program-, a másik az adatmező. Két független, és mezőnként egy relativ index-regiszterrel van lehetőség indexelésre, valamint indirekt cimzés is lehetséges, bármilyen mélységben. A megszakitás szervezése egyszintü. Illeszthető perifériák: konzolirógép, szalagolvasó, szalaglyukasztó, kártyaolvasó és sornyomtató.

Software feltételek: alkalmazkodni kellett a szabványos vezérlőprogramhoz /BOSS/, azaz biztositani kellett a nyomkövető program /MONITOR/ futását a BOSS mellett, valamint biztositani kellett a nyomkövetett program számára a vezérlőprogram szolgáltatásait. Miután nincs hardware memóriavédelem /és ezt a BOSS sem biztosit/ igy a MONITOR-nak gondoskodnia kellett mind a BOSS, mind a saját védelméről.

A MONITOR: általában szekvenciálisan, a tényleges végrehajtás sorrendjében szimulálja a nyomkövetendő program utasitásait. A működés szempontjából három fő részből áll: a párbeszéd blokkból, amelynek végrehajtása során megkapja a futáshoz szükséges információkat; az utasitásokat szimuláló ciklusból, amely a gép működési ciklusát szimulálja; végül az utasitások és a regiszterek kiirását eredményező kiiró rutinokból. Az ellenőrzés ill. a szolgáltatott információk mértéke szempontjából a következő állapotokban futhat:

- szimulálja az utasitásokat, és irásos információkat ad a program utasitásáról, esetleg regisztereiről is.
- szimulálja az utasitásokat, és nincs kiirás.
- elengedi a programot /megfelelően biztositva a visszatérést: csapdázás/.

2. Speciális követelmények

A rendszerprogramok nyomkövetésével kapcsolatban a következő kérdéseket emeljük ki:

- a MONITOR biztositsa a számológép fizikai szintü kezelését, mind ellenőrzött módon, mind pedig ellenőrzés nélkül. /Pl. a perifériakezelő rutin bejátszáshoz./;
- miután a nyomkövetendő program egy vezérlőprogram is lehet, igy lehetőséget kell biztositani arra is, hogy az összes program a MONITOR felügyelete alatt futhasson /autonómia/;
- figyelembe tudjon venni más rendszerprogram-környezetet, azaz biztositani tudja ennek mind a használatát, mind pedig a memória védelmet;
- különösen nagy méretü, illetve hosszu futási idejü programok esetén biztositania kell, hogy a felhasználó specifikálhassa a nyomkövetés szintjét, azaz azt, hogy a MONITOR milyen állapotban dolgozzon;
- a MONITOR a programmezőből a lehető legkevesebb területet foglalja le;
- legyen lehetőség a program futásának adott cimen történő megállitására;
- legyen lehetőség adott cim tartalmának megváltozása esetén történő kijelzésre;
- legyen lehetőség az adott futási intervallumon belül is intervallumok kitüntetésére /kiirás letiltás, szimuláció letiltás/;

- végül legyen lehetőség a ciklusok és szubrutinok nyomkövetésének változatos, könnyen kezelhető módjaira.

3. A megvalósitás alapelvei

A MONITOR készitése során a következő elveket tartottuk be:

a/ Minimális input információ elve: csak a feltétlenül szükséges információ megadása legyen kötelező a kezeléshez a lehető legkevesebb memoriter információra legyen szükség; a feltehetően gyakrabban használt lehetőségeket külön egyszerűbben is lehessen kezelni.

b/ Cserélhető perifériakezelés elve: a hardware lehetőségeken belül biztositsa a MONITOR a kivánalomnak megfelelő számu és fajtáju periféria használatát, valamint a felhasználó adhassa meg azt, hogy a nyomkövetés irásos információi melyik periférián jelenjenek meg.

c/ Azonnali beavatkozás elve: a felhasználónak bármikor módja legyen a nyomkövetést megállitani, és a továbbiakról határozni /regisztorkiirást kérni, esetleg ott folytatni, ahol abbamaradt stb. - ez a pultmegszakitás fizikai szintü lekezelését jelenti/.

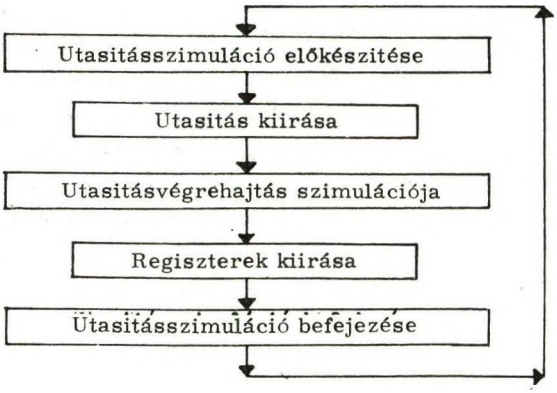
d/ A kiirás rugalmas megszervezésének elve: legyen lehetősége a felhasználónak mind a kiirt információk mennyiségét, mind pedig a formátumokat a lehető legtágabb határok között szabályozni.

e/ Dinamikus intervallumkezelés elve: a megadott intervallumokat /futásra, kiirásra stb./ a MONITOR dinamikusan értelmezi, azaz mindaddig intervallumon kivülinek éres minden címet, amig a kezdőcimet el nem éri, és ezután intervallumon belülinek, amig a végcimet el nem éri, függetlenül attól, hogy ezek milyen nagyságviszonyban vannak a kezdő és végcimmel.

4. Tapasztalatok

A fenti hardware és software feltételek, valamint a felállitott követelmények és elvek figyelembevételével készült el a NIM MONITOR. Tervezése és megvalósitása során több olyan módszert ill. ötletet alkalmaztunk, amelynek felhasználása hasonló feladatok megoldása esetén hasznos lehet.

a/ Kihasználtuk az output perifériák adatátviteli idejét, amit a BOSS perifériás rutinok használata esetén a gép kivárással töltene el. Ennek megvalósitására a MONITOR és vezérlőprogramtól független, az output perifériákat kezelő, önálló fizikai szintü kiirórutinnal rendelkezik, amely mindhárom output periféria esetén biztositja a MONITOR és a periféria egyidejü működését. Amig a periféria az adott utasitást irja ki, addig végrehajtódik az utasitás szimulációja, a regiszterek kiirása alatt pedig az utasitás szimulációjának befejezése és a következő utasitás szimulációjának előkészitése történik meg /lásd 1. ábra/.



1. ábra

A gyakorlati tapasztalatok alapján ez általában azt jelenti, hogy mindegyik periféria folyamatosan, maximális sebességgel nyomtat.

b/ A MONITOR az általa biztositott szolgáltatásokhoz egy-egy bit-állapotu jelzőt rendel hozzá. A jelző zérus állapota jelöli a szolgáltatás kérési lehetőségét, ha valamelyik program az adott szolgáltatást kihasználni, akkor a jelző a program folytatási címére mutat /pointer/.

c/ A gyorsitás érdekében ki lehetett használni az indirekt indexelés lehetőségét oly módon, hogy a MONITOR számára szükséges információkat megfelelően szervezett táblázatokba gyüjtöttük össze.

d/ Az ut sitásszimuláció három különböző módon történik: ugró utasitásoknál csak a cimkidolgozást szimuláljuk, az utasitások további nagy részét a géppel hajttatjuk végre, és igy néhány utasitást kell csak teljesen szimulálni.

e/ A MONITOR a nyomkövetett program ellenőrzését több szinten képes megvalósitani: biztositani tudja a program direkt futását /elengedés/, szimulált futását kiirással vagy anélkül a felhasználó kivánsága szerint.

f/ A kiirás rugalmas megszervezése jól használhatónak bizonyult: a regiszterek kiirását lehet kérni, fix, a MONITOR által meghatározott /standard/ formátumban; valamint az utasitás alaku, karakteres alaku, számblokk alaku /speciális 8 bitenként csoportositott alak/, decimális-, bináris-, ill. oktális szám alaku kiirási formátumok bármilyen kombinációjában.

A PROGRAM-FUTTATÁS, KIPRÓBÁLÁS SEGÉDESZKÖZEI

Farkas Anikó
INFELOR

A számitógép nyujtotta lehetőségek maximális kihasználása csak ugy valósulhat meg, ha biztositott a gépen folyó munkák szervezettsége. Ez azt jelenti, hogy a számitógépnek rendelkeznie kell olyan üzemeltetést irányitó rendszerrel, amely biztositja a gépen folyó munkák folyamatosságát, az egyes feladatok megoldásához szükséges tevékenységek összehangolását, az egyik programról a másikra való áttérés időveszteség nélküli lebonyolitását, a külső tárolók és perifériák és a belső memória közötti kapcsolat irányitását stb. Mindebből nyilvánvalóan adódik a program kiszolgálására forditott idő csökkentésére irányuló törekvés, aminek elsődleges követelménye a manuális operátori beavatkozás minimalizálása.

MINSZK-MONITOR

Ezt a célt kivánta megvalósitani a MINSZK-2 gépre készült MONITOR rendszer, amely a gép autokód nyelvén /MITRA-2/ irt programokra vonatkozik. A rendszer alapját a LISP-nyelv MINSZK-2 gépen kidolgozott LAnc FOrditója /LAFO/ képezi. A program futására vonatkozó döntéseket a kezelési utasitásba épiti be a felhasználó, tehát nem kell a gép mellett lennie programjának futásakor.

A szimbólikus minta-mondatok elvileg kétfélék. Egyik részük a munka megszervezéséhez /működtető rendszer és gép kapcsolat/ szükséges, a másik pedig az autokód programmal kapcsolatos tevékenységek /működtető rendszer és MITRA-program kapcsolat/ leirására adnak lehetőséget.

A szervező utasitások egyrészt adminisztrativ jellegüek /pl. felvétel a programkönyvtárba vagy törlés onnan, munkaterület vagy tárolóterület igénylés, program javitás/ másrészt utasitó jellegüek a gépkezelő felé valamilyen manuális tevékenység végrehajtására.

A felügyeleti rendszer és MITRA-program kapcsolatát olyan MONITOR-utasitás valósitja meg, mely megengedi, hogy a futó MITRA-program kivezéreljen a MONITOR-programba, ott végrehajtson előre megtervezett tevékenységeket és ha a felhasználó igényli vissza is tudjon térni. Ennek kivitelezése egy közbülső csapda-tábla segitségével történik, amelyben a megszakitási pönton kivül olyan információk kerülnek megjegyzésre, amelyeket a MITRA-program folytathatósága megkövetel.

A rendszer segitséget kiván nyujtani a programozónak a programhibák okának és helyének meghatározásával is. Olyan hibák kezelésével foglalkozik, amelyek egyébként lehetetlenné teszik a program futását. A programozó maga is tervbe veheti az ilyen hibák kezelését, pl. a tevékenység tulcsordulás esetén.
Ha azonban a felhasználó nem élt ezzel a lehetőségével /nem "számitott" az esetleges hibára/, a rendszer akkor is közli vele az előfordult hiba okát és helyét. 16 hiba-tipus figyelésére van felkészitve a rendszer, amelyek között ott van a tulcsordulás figyelése, a ki nem tett referenciaszámra vezérlés, stb.

Természetesen a rendszer hatékonyságát nagymértékben meghatározzák a hardware adta lehetőségek mint a megszakitás, memóriavédelem, perifériális felszereltség, belső óra, stb. 1967-68-ban, amikor a rendszerünk készült csak nagyon minimális igényeink lehettek a géppel

szemben. Igy gépidő csökkentést nem is vártunk a rendszer használatától, csupán folyamatos üzemmódot, és tapasztalatot kivántunk szerezni későbbi komolyabb operációs rendszer készitéséhez.

EMG 830 FELÜGYELŐ PROGRAMOK

Mig ez a felügyeleti rendszer operációs-rendszer jellegü volt az EMG 830 gépekre készült felügyeleti programok a program-kipróbálás segédeszközei. Olyan programok ezek, melyek nyomon követik a program futását és információt adnak ki róla. A felhasználó számára ezen programok hatékonysága abban jut kifejezésre, hogy olyan szinten adja az információkat, amilyen szintü a követendő program, továbbá tud a program futásáról részletes információt adni, illetve képes arra, hogy adott esetben a programozó által kért leszükitett információ-mennyiséget adja csak ki.

EMG 830 NYOMOZÓ

Ezen programoknak két tipusát készitettük el. A NYOMOZÓ esetén a program követése az utasitások szimulálásával történik, melynek mélysége az utasitástipustól függően változik. Szimulálásra azért van szükség, hogy nyomon követhessük és rögzithessük az utasitások hatására bekövetkező regiszter- és tárrekesz-tartalom változásokat. A felhasználó igényeinek megfelelően alakitottuk ki müködési állapotait:

- szimulálja a programot, de információt nem ad ki, ha a program azon utasitásait követi, melyről a felhasználó nem kért információt;
- szimulálja a programot és információt ad ki a felhasználó által megkivánt mélységben és formában;

- szabadon enged futni minden esetben, ha az operációs rendszer területére történik vezérlés, egyébként, ha a felhasználó kérte, de mindkét esetben lehetőséget ad a nyomkövetés folytatására.

A felhasználónak lehetősége van előirni, hogy a teljes programjáról vagy annak csak részintervallumairól kér információt, illetve az intervallumon való minden átfutáskor kéri-e az információt vagy csupán az első meghatározott számu átfutáskor igen és később nem, esetleg megforditva.

Az információkiadás mélységét döntően az határozza meg, milyen regiszter-tartalom változásokra kiváncsi a felhasználó, de természetesen kérhet pl. csak cimkövetést, indirekt cimlánc követést is.
Bizonyos kulcs-szó intervallumokhoz való hozzárendelésével pedig opcionálissá tehető a nyomkövetés.

EMG 830 MONITOR

A nyomozó mellett szükségesnek látszott olyan program elkészitése is, amelynek feladata azt biztositani, hogy a felhasználó programjáról, annak futása során, információt nyerhessen a program tetszés szerinti pontjában
- regiszterek pillanatnyi állapotáról,
- tárintervallumok pillanatnyi tartalmáról,

valahányszor a program a jelzett ponton halad át. Az EMG 830 gépekre készült fenti rendeltetésü programot MONITOR-nak neveztük. Ennek felépitése lehetővé teszi, hogy hivható: konzolirógépről, assembly-programból és magasabb szintü nyelven irt programból. A megvalósitás

mindhárom esetben a megjelölt információs pontokhoz /ahol az információt kérik/ rendelt csapdatábla intervallumok közbeiktatásával történik.

BEFEJEZÉS

A két program-kipróbálást segitő eszköz feladata látszólag hasonló, de több olyan jelentős eltérés van a két program felhasználása és belső strukturája között is, mely szükségessé teszi mindkettő meglétét.

Eltérés van:

- a kapott információ tartalmában,
- az információkiadás helyében,
- a program futási idejének növekedésében.

A VIDOS könyvtárrendszere

Mócsi Zoltán
INFELOR

A VT-1010B géphez kapcsolt FEX-3-as mágneslemezre kidolgoztunk egy könyvtárrendszert. A rendszer tervezése közben figyelemmel kellett lennünk arra, hogy

1. a gép operativ tára kicsi
2. a lemez kapacitása kicsi /800 KB/, a lemezcsomagok nem cserélhetők
3. a hardware operativitása erősen korlátozott, ami a VIDOS rendszerének számtalan sajátosságában megnyilvánul.

Eltérünk tehát a szokásos file szervezési technikáktól. A lemez teljes területét öt "könyvtár" között osztottuk szét.
Először a könyvtárak funkciójáról fogunk beszélni

SVL könyvtár /SuperVisor Library/ feladata

a/ tárolni a supervisor-t
b/ tárolni a supervisor u.n. "belső hivású" szolgáltatásait, amelyek nincsenek állandóan a fő tárban, hivásuk esetén egy cserét

külön behozza ezeket a nekik fenntartott helyre.

c/ tárolni az operátori parancsok, külső szolgáltatások rendszerét, amellyel a supervisor az embergép kapcsolatot létrehozza.

EML könyvtár /Extraevde-Macro Library/.
Itt található

a/ az u.n. extrakódok rendszere, amely a gép tényleges utasitásrendszerét képezi. Ezek értelmező-szinten hivható szubrutinok, amelyek a hardware lehetőségeit hivatottak kibőviteni. Az extrakódok szekciókba vannak csoportositva. A loader szekcionként épitheti bele a célprogramba az extrakódokat szükséglet szerint.

b/ a makroassembler "könyvei" azaz egy tipusba tartozó standard makroutasitás csoportjai.

SPL könyvtár. /System Program Library/.
Ez a könyvtár tartalmazza a géptermi segédprogramokat - utility-ket. Javitó, szerkesztő, másoló stb. programok, speciális szubrutinrendszerek tartoznak ide.

URL könyvtár /UsérR Library/ örzi a felhasználók állandó file-jait, mig a WKL könyvtár /WorK Library/ területe adja a munkafile-okat.

Az első három könyvtár területe rögzitett /csak külön az e célra szolgáló rendszerprogrammal változtatható/, mig URL és WKL határa dinamikus URL mindig tele van, ha ide, valamit fel akarunk vésni, az ehhez szükséges területet WKL-ből vesszük el. Mivel a WKL-beli tartományok csak egy program futása alatt élnek, nem okaz hibát, ha a futás után a program által megörzendődnek jelölt tartományokat URL-be ilymódon átteszszük; illetve két felhasználói program futása között URL-t bárhogyan megváltoztatjuk.

Minden könyvtárba tartozik egy katalógus, amelyben a könyvtár minden tartományáról van egy bejegyzés. A bejegyzés tartalmazza a tartomány

1. nevét /6 karakter/
2. tipusát /adat- program, ha program milyen formában van stb./
3. lemez-cimét és hosszuságát
4. különféle, a tipustól függő segédinformációkat.

A katalógusok első szektora az SVL könyvtárban található, innen indulva a katalógusok egy-egy egymáshoz láncolt szektor-sorozatot alkotnak az illető könyvtár területén.

Az egyes könyvtárak katalógusai egymásba is láncolva vannak, amely bizonyos esetekben lerövidítheti egy tartomány megkeresését. E láncolás módja:

A könyvtárak kezelése főleg operátori parancsokkal történik. Ezek az operátori parancsok lehetőséget nyujtanak

a/ könyvtárakba tartományok felvételére valamilyen perifériáról
b/ tartományok logikai törlésére
c/ logikailag törölt tartományok fizikai törlésére /az adott könyvtár tömöritésére/
d/ adott tartomány kicserélésére
e/ adott tartomány másik könyvtárba történő áthelyezésére
f/ könyvtárak katalógusának kiiratására
g/ könyvtárak állapotának megvizsgálására

stb. hogy csak a futtatottakat emlitsük.

Rendszerünk láthatóan kevesebbet nyujt, mint a nagy automatikus file-kezelési rendszerek, Többször és mélyebben kell az operátornak beavatkoznia, szorosabb kapcsolatot kell tartania a rendszerrel mint általában. De - szerintünk - a VT 1010B kisgéphez nagyobb rendszer épitése, ha nem is lehetetlen, de felesleges lett volna. A fenti rendszer lehetőségeit elég dinamikusan nyujtja, bár koncepciója nem egészén homogén.

A VIDOS RENDSZER INPUT/OUTPUT SZERVEZÉSE

Nagy Kázmérné
INFELOR

Bevezetés

A VIDOS I/O rendszer felépitése megkönnyiti az I/O tevékenységekre assembly nyelven való hivatkozást.

Az I/O rendszer funkcióját tekintve két részből áll:

- fizikai szintü rész, amely a tényleges adatmozgatást szervezi, a tár és a perifériás készülékek között;
- a logikai rész, amely szekvenciálisan szervezett adathalmazok /file-ok/ feldolgozását végző tevékenységeket /makrokat/ adja.

Fizikai szint

A perifériás készülékek mozgatásához szükséges információkat az un. CCB táblában helyezi el /a programjában/. Az átvitellel kapcsolatos valamennyi tevékenységet értelmező szinten ugy jelzi, hogy a cimét adja meg.a táblának.
A tábla felépitése lényegében készüléktől független, de az egyes paraméterek értelmezése már a készüléktől függő.
A tábla 16 byte hosszuságu, és az alábbi lényegesebb információkat tartalmazza:

- az egyes tevékenységek kezdőcimei
- állapotbyte, amely az átviteli tevékenység pillanatnyi helyzetét mutatja
- periféria logikai száma
- az I/O tevékenységhez szükséges vezérlő információk
- pufferhossz, puffer kezdőcim
- számláló, amely a karakteres átvitelnél müködik
- lemezcim, amely a diszk mozgatáshoz szükséges
- elágazási cim, amelyre az adott tevékenység rendellenes végrehajtásakor kerül a vezérlés.

Az átvitel kezdeményezése az un. PERI rutinnal történik. Értelmező üzemmódban a CCB tábla 3. byte-jának cimét kell megadni az aktiváláshoz.
Az átvitel alatt a főprogram visszakapja a vezérlést, és a WAIT rutinnal /a tábla első byte-jának cime/ figyelhetjük az átvitel befejezését.

Logikai szint

A logikai szintü rutinok adathalmazok /file-ok/ kezelésére, szekvenciális feldolgozására alkalmasak. Az adathalmaz leirását az LFD táblával végezzük, amelyben egyuttal a logikai szintü rutinok paramétereit is megtaláljuk, továbbá a munkaterületeket. A tábla első részében lényegében azokat az információkat találjuk meg, amelyek a CCB táblában vannak, a fizikai szinten. Ezeket az adatokat a tábla második felében lévő adatokból lehet meghatározni, esetenként.

A tábla második felében található információk továbbá:

- cimke tipus, amely megmondja, hogy van-e cimke, s ha igen, akkor az standard formáju-e;
- a felhasznált charakterkód tipusa;
- a rekordszerkezet, amely lehet
  - egy blokkban több fix hosszu rekord,
  - egy blokk, egy rekord,
  - határozatlan vagy fix hosszu,
  - egy blokkban több változó hosszu rekord;
- file vége rutin, amely megállapitja az adathalmaz végét és a megadott cimre adja a vezérlést;
- maximális blokk- és rekordhossz;
- a rekordcim cime, amit a GET, illetve PUT utasitások használnak;
- filenév;
- hosszuság, amely lemez esetén az adathalmaz számára lefoglalt területet adja.

A programozó által aktiválható rutinok, az OPEN és CLOSE, amely a file nyitását és a file lezárását végzi. A GET és PUT, amely az input, illetve output tevékenységet szervezi olymódon, hogy rekordokat mozgat, amelyekhez akár a pufferben, akár a programozó által kijelölt területen lehet hozzáférni. A RELSE, illetve TRUNC a fenti I/O tevékenységeknek a kiegészitője abban az esetben, ha nem kivánjuk a teljes átvitelt.

Befejezés

A fent vázolt rendszer pillanatnyilag a következő perifériás készülékekre müködik;

- irógép,
- lyukszalag,
- lyukkártya,
- diszk,
- sornyomtató, és
- mágnesszalagra, de csak fizikai szinten.

A teljes kiépitéssel még ebben az évben elkészülünk.

A MINORG-2 PROGRAMOZÁSI NYELV ÉS MINSZK-2 3 COMPILERE

Makay Árpád és Hunya Péter
József Attila Tudományegyetem Kibernetikai Laboratóriuma

1. A MINORG-2 programozási nyelv megtervezésénél messzemenően figyelembevettük, hogy a MINSZK-23 számitógép "tiszta" byte-szervezésü gép. Ez alatt azt értjük, hogy nincsenek olyan utasitások, amelyek előre meghatározott hosszuságu byte-csoportok tartalmán végeznek müveleteket. Végezhető müvelet egy byte-on, az utasitásban megadott számu byte-on (K-byton) és az utasitásban megadott byte-tól kezdődő, határjelet tartalmazó byte-ig terjedő byte-csoporton (mezőn). Ebből következően aritmetikája is szegényes: csupán egész számokon végzi az alapmüveleteket és az eredmény csak előre preparált mezőbe helyezhető.

A programozási nyelv elemi tipusainak rögzitésekor tehát "szabad kezet" kaptunk, mivel a müveleteket mindenképpen szubrutinokkal és nem egyetlen hardware-utasitással kell végeztetnünk. Megtartottuk a byte-szervezés előnyeit oly módon, hogy nem a szokásos integer és real számtipusok az alapmennyiségek, hanem a deklarációban megadható egész- és tizedesjegyet tartalmazó fixpontos számok és a kétjegyü kitevővel ellátott, szintén tetszés szerinti jegyü mantisszás lebegőpontos számok. Az aritmetikai kifejezések értékének kiszámitásakor a célprogram verem-memóriát használ: a változók értékei verembe-töltéskor egy standard formára konvertálódnak, a müveletvégzés pedig a veremben található, standard formáju számokon történik. Az értékadást szintén a standard formáról a változó formájára való konvertálás előzi meg.
További elemi tipus a változó hosszuságu karaktersorozat, mig az összetett tipusok az egy és kétdimenziós mátrixok és az (egy vagy több-

szintü) rekordok, amelyek mezőiként az elemi tipusok engedhetők meg. A változókra vonatkozó memóriafelosztás statikus: a program szegmensekre tagolódik, minden szegmens rendelkezik saját változómezővel. A saját változómezőben az egyes változók helye fordításkor rögzitődik. A szegmensek közötti kapcsolatot a közös változómező biztositja, amelynek felosztását szintén a fordító végzi el.
Aritmetikai kifejezések és relációk képzéséhez konstenseket, változókat, elemi függvényeket és (egy vagy több paraméteres) függvényeljárás-hivásokathasználhatunk fel. A müveletek precedencia-szabályai és a zárójelezés az ALGOL-60 feltétlen aritmetikai kifejezéseivel azonosak. Aritmetikai kifejezéseket értékadó utasitásokban és eljárások aktuális paramétereiként használhatunk, mig a relációknak a feltételes vezérlésátadó és a ciklusutasitásokban van szerepük.

2. A MINORG-2 programozási nyelv egyik fő felhasználási területe az adatfeldolgozás. Ezért tartalmazza azokat a lehetőségeket, amelyek biztositják az adatfeldolgozó eljárások egyszerü megfogalmazását.
Mig a rekord formáját egyértelmüen meghatározzák a rekord mezőit alkotó elemi tipusok belső ábrázolási formái, a külső adathordozókon megjelenő rekordokra forma-specifikáció vonatkozik. A specifikáció felsorolja a mezők tipusait (fix- vagy lebegőpontos szám, karaktersorozat) és a mezők formáit az adathordozón (lyukkártyán, lyukszalagon, nyomtatón, irógépen). Az input/output utasitások mindig hivatkoznak egy forma-specifikációra és felsorolják azokat a változókat, amelyek rendre megfelelnek a forma-mezőknek. Input esetén a külső formáról a neki megfelelő változó belső formájára történik konverzió, output esetén pedig forditva. Néhány speciális utasitás gondoskodik a táblázó müveletek változatos formátumainak kialakithatóságáról.

Háttérmemóriaként a MINSZK-23 számitógépen mágnesszalagok használhatók. Ezért a programnyelvben csak szekvenciális file-feldolgozás alkalmazható. A file elemei a rekordok, amelyek mágnesszalagon is a belső ábrázolási formának megfelelően tárolhatók. A file specifikációjában meg kell adnunk a file tipusát (input vagy output) a korábban deklarált rekord nevét, amelyből a file felépül, a mechanizmus azonositóját és a file memóriabeli puffer-területének méretét. File-okra a megnyitás, lezárás, folytatás (korábban lezárt file-hoz további rekordok hozzáirása) és a váltás müveletét alkalmazhatjuk. Mivel minden rekord számára deklarációjakor memóriaterületet biztosit a fordító, file-váltáskor a puffer- és rekordterület között csere történik, igy a rekord feldolgozása nem a puffer-, hanem a rekordterületen történik. (Az egységesség is ezt a megoldást indokolja: ugyanis rekord deklarálható anélkül is, hogy file eleme lenne.)

3. A MINORG-2 programok szegmensein belül az utasitások, specifikációk és deklarációk sorrendje közömbös. Az azonositókat első felhasználásuk előtt deklarálni kell, kivéve a cimkéket és eljárásneveket. A szegmensbe lépés csak az elején történhet, kilépés bárhol, ahova szegmensvég direktivát irtunk.

   A program szegmensekből (amelyek közül az elsővel indul a program végrehajtása), utánuk eljárásokból (eljárásutasitással illetve függvénykifejezéssel felhivható szubrutinokból) és külső eljárásnév-specifikációkból áll. Külső eljárás lehet önállóan forditott MINORG-2 nyelvü szubrutin, vagy szegmens, vagy olyan assembly-nyelvü program, amely a megadott szabvány szerint kapcsolódik a főprogramhoz (paraméterátvétel és függvényeljárás esetében az eredmény közlése tekintetében).

Szubrutinok formális paraméterként fix- vagy lebegőpontos változót (amelyet érték szerint kezel a fordító), karaktersorozatváltozót, tömbnevet és file-nevet (amelyeket cim szerint kezel a fordító) használhatunk. A szubrutinok egymást hivhatják, de rekurziv hivás nem megengedett. Aktuális paraméterként kifejezés, tömbnév, file-név, formátum és rekordnév használható. (A formátumot és rekordnevet, mint aktuális paramétert csak az assembly-nyelvü szubrutinok képesek kezelni.)

4. A MINSZK-23 számitógép assembly-nyelvén felirt fordító három menetben készit a forrásnyelvi programból az operációs rendszer szerkesztő programja számára alkalmas szegmenset. A szerkesztő program gyüjti össze a célprogramhoz szükséges szubrutinokat, külső eljárásokat, azonkivül kijelöli a forrásnyelvi szegmensek saját és közös változómezőinek memóriabeli helyét. Az operációs rendszerben lényeges változtatások lennének ahhoz szükségesek, hogy az overlay-technikával a célprogram memóriaszükségletét csökkentsük, igy futáskor a teljes program és változómezeje a memóriában van.
A forrásnyelvi program beolvasása, javitása, esetleges nyomtatása utáni első menet a tulajdonképpeni szintaktikával egybeépitett fordítás. Mivel a nyelv operátor-jellegü, az egyes operátorok (utasitások, deklarációk, specifikációk) fordítását egymástól független részek végzik. Az aritmetikai kifejezések, relációk, tömbdeklarációk fordításához közös verem-memóriát használunk. A többi operátor fordításához általában elegendő volt az egymásutáni jelpárok (a feldolgozás alatt álló jel és a programban következő jel) figyelembevételének technikáját alkalmazni.

A menet alatt gyüjtött és később felhasználható információkat a változótáblázat tartalmazza. Az azonositók mellett feljegyzésre kerül a tipus,

azon belüli a méret, a mezők és a rekordok közti kapcsolat, szegmens-információk, formális paraméter-jelzések, stb. Az első menet közben a fordító azonositók alapján keres a táblázatban, mig a célprogramhoz az azonositókat a változótáblázatbeli cimek képviselik. Külön konstans- és külső cimke táblázatok készülnek az előforduló számokból, idézetekből, forma-, tömb-, rekord- és file vektorokból illetve a felhasznált szubrutinok és külső eljárások neveiből.

A második menet az elkészült program végigfutásával megállapitja a programban előfordult illetve az első menetben beiktatott cimkék célprogrambeli cimét, ezután kitölti a változótáblázatot az azonositók célprogrambeli, konstanstáblabeli, saját és közös változómezőbeli relativ cimeivel.

A program harmadik átfutása közben a változótáblázatra vonatkozó cimeket a táblázat alapján az emlitett relativ cimekre cseréli a fordító. Egyuttal gondoskodik arról, hogy kialakuljon a szerkesztő program által kivánt szegmensforma a megfelelő információs táblázatokkal együtt. Fordítás közben nyomtatón szintaktikus hibajegyzéket kapunk, azonkivül néhány olyan információt a célprogramról, amely a programpróbákat megkönnyiti.

5. A MINORG-2 nyelv a MINSZK-23 számitógép egyetlen problémára irányitott programozási nyelve a makro-utasitások nélküli assembly-nyelv mellett. Ezért igyekeztünk alkalmassá tenni adatfeldolgozási célokra éppugy, mint tudományos-müszaki számitások elvégzésére a file- és input /output-rendszer illetve az eljárások, függvények/ hajlékony kezelésével.

FORTRAN COMPILER KISSZÁMITÓGÉPRE

dr. Bach Iván, Laufer Judit

MTA Automatizálási Kutató Intézet

Az alábbi rövid ismertetés során egy BASIC FORTRAN compiler leirásának néhány tapasztalatát és megoldását szeretnénk közreadni.

A gép amelyre a compiler készült, a CII 10010 /Videoton 1010B/. Ez mint ismeretes 8 bites byte szervezésü, lapozott, szegényes gépi utasitáskészlettel rendelkező gép. Ilyen körülmények között természetesen nincs más ut, mint compilálás során egy közbenső nyelvre történő forditás, amelyet a végrehajtó program értelmez, interpretál. Ennek a nyelvnek a kidolgozása is a compiler iró feladata volt.

A compiler olyan konfigurációra készült, amely nem tételez fel háttértárat, ebből logikusan következik a követelmény, hogy a forditás egy menetben történjék. Igy a már egyszer compilált textus elvész és minden szükséges információt a forditás során táblázatokban kell rögzitenünk.

A közbenső nyelv tekintetében szabad kezünk volt, mind az utasitáskészlet, mind az utasitások funkciói tekintetében. Az az absztrakt gép, amely a végrehajtó programcsomag birtokában a közbenső nyelven irt programokat értelmezi, már természetesen rendelkezik egész és valós aritmetikával, szubrutinhivó, cikluskezdő és záró stb. utasitásokkal.
Az absztrakt gép utasitásai természetesen rutinokat aktiválnak, és igy a hardware gép szintjén ezek gyakran 100-nál több valódi gépi utasitást jelentenek. Minthogy a gép eredeti hardware lehetőségei minimálisak - legalábbis aritmetika ori-

entált nyelvek tekintetében - és igy a hardware csak áttételen keresztül volt felhasználható, lazitottunk a forrásnyelv néhány hardware orientált kötöttségén. Igy például a FORTRAN ismert korlátozása az index kifejezésekre nagyon előnyös lehet egy indexregiszterrel biró gép esetében. Nálunk ez teljesen érdektelen volt, és bármilyen aritmetikai kifejezést engedélyezünk.

Hasonlóan kissé más filozófiát kényszeritettek ránk a gép adottságai a hibajelzés technikájának kialakitásában. A detektálhatóság tekintetében két hibafajtát különböztethetünk meg. Az egyik a forditás, a másik csak a futás során észlelhető hiba. Az előbbi tekintetben hátrányos helyzetben voltunk, nem is a gép adottságai, hanem az egy menetes forditás következményei miatt. Igy például az u.n. kiterjesztett DO ciklus engedélyezése helyett azt a korlátozást érvényesitettük, hogy DO ciklusba való ugrást sohasem engedünk meg.

A futás közben detektálható hibák tekintetében "előnyös" helyzetben vagyunk. Ez ugyanis nemcsak software-technikai, hanem gazdaságossági kérdés is. Nagy utasitáskészletü gépek esetén, ahol a forditás végső soron gyakorlatilag gépi nyelvre történik, egy hibafelismerő ellenőrzés beiktatása a célfeladatokkal összemérhető időigényü. A mi esetünkben, ahol egy aritmetikai müvelet 100-200 gépi utasitás, egy esetleges ellenőrzés beiktatása nem szignifikáns. Ennek figyelembevételével lényegesen nagyvonalubban bántunk a futás közben kiadott hibajelzésekkel.

Ennek eredménye néhány igen lényeges hiba jelzése. Igy indextúlcsordulás, értéktelenség, ellentmondás szubrutin hivásban. E két utóbbiról néhány szót.

A második azt jelenti, hogy nem tulajdonithatunk értéket olyan mennyiségeknek, amely még nem kapott értéket. Technika-

ilag ez ugy van megoldva, hogy a futás elején értelmetlen - formálisan minusz nulla jelentésü - karaktersorozatot irunk a változók byte-jaiba. Ha a felhasználáskor ezt a karaktersorozatot találjuk a memóriahelyen, hibajelzést adunk.

Szubrutin hiváskor, ha értéket kapott formális paraméternek megfelelő aktuális paraméterek azonos memóriahelyre utalnak ez inkompattibilitást eredményez, ami hibajelzést vált ki. Mellesleg ezeket a szolgáltatásokat én nagy gépek forditóprogramjában is opcionálisan kivánatosnak látnám.

Példaképpen először az aritmetikai kifejezéseket fordító rutin néhány gondolatát ismertetjük.

Mindenekelőtt megemlitjük, hogy a rutint minden olyan esetben aktiválnunk kell, amikor aritmetikai kifejezést várhatunk, igy az értékadó utasitáson kivül tömbelemek meghatározásánál, szubrutin és FUNCTION hivásoknál, az IF utasitásban és utasitásfüggvényekben. Minthogy ezek egymásba ágyazódhatnak, biztositanunk kellett a mélységkorlátozás nélküli rekurziv hivás lehetőségét. Ez lehetőséget ad olyan kifejezések irására, mint

```
/ SH-deklarált FUNCTION/
        SH (SH (SH (X)))
és / I deklarált tömb/
        I(I (I (5)))
```

Az aritmetikai kifejezések kidolgozásánál vázlatosan három verem van alkalmazva.

Kettő forditás során, egy a futás alkalmával. A forditásnál az operandus és kód verem szerepel.

Az operandus-veremben tárolunk a szereplő mennyiségekről

minden információt mindaddig, amig teljes feldolgozásuk meg nem történt. A kód-verem a müveleti jeleket tárolja. A RUN--verem a részeredmények tárolására szolgál.

A rutin három főrészt tartalmaz, azonositó elemző, müveleti jel elemző és programgenerátor, bár ezek funkcionálisan nincsenek eleve szétválasztva.

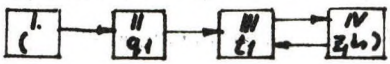
A forditás mechanizmus olyan, hogy lehetőleg "sorfolytonos" legyen. Ez annyit jelent, hogy mihelyt valamely kifejezést értelmezni tudunk, annak megfelelő kódot generálunk még akkor is, ha a felhasználásra nem kerülhet sor. Ilyenkor ezt a RUN-verembe helyezzük és felhasználásáig ottmarad.

Második példánk a FORMAT utasitás.
A FORMAT utasitás általános alakja /az USA Standard FORTRA alapján/ az alábbi :

$$N \quad \text{FORMAT}\,(q_1 t_1 z_1 \ldots . t_n z_n q_2) \qquad /1/$$

ahol : q slashek sorozata vagy space
t_i mezőleiró vagy mezőleirók egy csoportja
z_i határoló karakterek
n nulla is lehet

A FORMAT lista elemzésekor programunk egy olyan automataként fogható fel, melynek bemenőszava az elemzendő sor.
A továbbiakban a FORMAT utasitásnak csak a listájáról fogunk beszélni. /1/-ből látható, hogy ez a lista három jelcsoportból, valamint kezdő és végzárójelekből épül fel. Az egyes jeltipusok elemzése egy-egy részautomata feladata. A négy részautomata egymáshoz való viszonya az alábbi :



A fenti automata Mealy féle automata, mert kimenőjelét /esetünkben a tárgyprogram vagy hibaüzenet/ a bemenőjel /azaz a forrásprogram/ és az az állapot, melyben az automata van, egyaránt megszabja.

Felirtuk az automaták átmeneteinek és kimenőjeleinek táblázatát, majd ezt addig alakitottuk, mig egy minimális automatát nem kaptunk. Ezt az automatát azután ugy lehetett megvalósitani, hogy egy-egy állapotnak egy-egy rutint feleltettünk meg.

A számstring esetén kénytelenek voltunk váltót alkalmazni, melynek különböző állása alapján különböztettük meg, hogy multiplicitást vagy mezőszélességet vár a program.

TPA FORTRAN RENDSZEREK

Viszt Éva

MTA Központi Fizikai Kutató Intézet

A TPA 1001 számológép második generációs, kisteljesitményü, univerzális gép. Az alapgép tárkapacitása 4K szó (12 bit/szó), de 32K szóra bővithető. Alap kiépitésben egy ASR33 teletype irógép és 1 db gyors lyukszalagolvasó csatlakozik hozzá. Illeszthető hozzá továbbá gyors lyukszalaglyukasztó, sornyomtató, mágnesszalag, diszk, plotter stb.

1969/70-ben a KFKI Számitástechnikai Osztályán egy öttagu munkacsoport elkészitett a TPA 1001 alapkiépitésü gépre egy 4K FORTRAN fordító, program kipróbáló és futtató rendszert. A rendszer elemei a következők:

- FORTRAN szintaktikus analizátor
- FORTRAN fordító
- áthelyezhető bináris loader
- nem áthelyezhető szolgálati rutinok
- áthelyezhető bináris formátumu szubrutin könyvtár
- szerviz programok.

A rendszer néhány megkötéstől eltekintve a BASIC FORTRAN nyelv implementációja a TPA 1001 kisszámológépre. Gyors lyukszalag lyukasztása megengedett.

A kis tárkapacitásu operativ memória miatt a szintaktikai vizsgálat és a forditás két menetben, különböző programokkal történik. A fordító bemenete egy szintaktikusan helyes forrásnyelvü szegmens vagy ilyen szegmensek sorozata. A fordító lyukszalagos overlay technikával dolgozik. Ez a technika a fordító kezelését nehézkessé teszi, és a forditási időt jelentősen megnöveli. A fordító kimenete a leforditott áthelyezhető bináris (a/b) szalag.

Az áthelyezhető bináris loader egy két menetes betöltő program. Első menetben memória térképet készit, második menetben betölti az a/b szegmensekből alkotott programot. A szegmensek forrásnyelve vagy FORTRAN vagy SLANG2. (A SLANG2 a TPA 1001-re implementált assembler nyelv, melyet a Számitástechnikai Osztályon dolgoztak ki. Legfontosabb tulajdonsága a szegmentálhatóság.)

A nem áthelyezhető szolgálati rutinok tartalmazzák a 4 szavas lebegőpontos, a két szavas egész aritmetikát és az input/output rutinokat. A fordító a forrásnyelvü szegmenseket, egy-két gépi utasitástól eltekintve, a fix kapcsolócimü szolgálati rutinok hivásának sorozatára forditja le. A szolgálati rutinok elfoglalnak a memóriában 3432_{10} rekeszt, a program számára 1664_{10} rekesz áll rendelkezésre.

A TPA 8K-FORTRAN kialakitásánál Mező Istvánnal arra törekedtünk, hogy a 4K-FORTRAN rendszer kezelési nehézségeit kiküszöböljük, a program rendelkezésére álló helyet növeljük, biztositsuk a 4K-s rendszerrel való kompatibilitást olyan értelemben, hogy a 4K-s fordítóval forditott programok futtathatók legyenek az uj rendszerben is, és legalább a rendszerprogramok input/output perifériájaként bekapcsoljuk a rendszerbe az NC-245 tipusu 32K alapkapacitásu diszket, amely maximálisan 128K-ig bővithető.

A fenti céloknak megfelelően két rendszer készült el, a TPA 8K-FORTRAN a 4K-s rendszer továbbfejlesztéseként papirszalagra orientált, majd ennek bővitésekont a diszkre orientált TPA 8K-D FORTRAN. Ez utóbbi, természetesen, papirszalagos üzemmódban is használható.

A TPA 8K-FORTRAN forditónál az overlay technikát megszüntettük, az overlay programok és a táblázatok a felső modulba kerültek. (Szokásos szóhasználat: "alsó" vagy "nullás" modul; "felső" vagy "egyes" modul.) A megnövelt táblázatok lehetőséget adnak bonyolultabb, több azonositót tartalmazó szegmensek leforditására is. Ahhoz, hogy a két rendszer outputja azonos legyen a 8K-s fordítóval forditott program is csak egyetlen modulban helyezkedhet el. Ez jelen esetben a felső modul majd-

nem teljes egészében. A nem áthelyezhető szolgálati rutinok az alsó modulban maradtak. Aktivizálásuk viszont mindkét modulból lehetséges, mivel nemcsak a program hivhatja azokat, hanem egymást is hivják. Az aritmetikai és input/output rutinokat alkalmassá kellett tenni a felhasználás mindkét formájára. Biztositani kellett a felső modulban egy-egy átkapcsoló rutint mindenegyes fix kapcsolócimü alsó modulbeli rutinhoz. (A modulváltás a programban két speciális utasitással történik.) Az átkapcsoló rutinok a felső modul első három lapját foglalják el, az első, un. zérus-lapot csak részben, a fennmaradó hely felhasználható FORTRAN-SLANG2 vegyes nyelvü programokban zerus common-ként (zerus common csak a SLANG2-ben van).

Az alsó modul jobb kihasználása érdekében a FORTRAN standard függvényeket is itt helyeztük el fix cimre. A 4K-s rendszerrel való kompatibilitás érdekében elkészitettük ezek átkapcsoló rutinjait és egy egy-lapos (200_8 szó) SLANG2 szegmenssé füztük össze azokat. Ez a szegmens minden olyan esetben beépül a programba, ha legalább egy FORTRAN standard függvényt használunk. Igy ez a rendszer maximum 29 lap (3712_{10} szó) hosszuságu program betöltésére ad lehetőséget.

A fordító rendelkezik egy un. installációs paraméter készlettel: gyors papirszalag olvasó, lyukasztó, paritás ellenőrzés beolvasáskor és teljes program fordítás. (Teljes egy program, ha a szegmensek sorozata FINISH utasitással van lezárva.) Ha a felhasználó másként kiván rendelkezni, akkor a fordító elinditása után az un. "kezdeti beszélgetés"-ben adhatja meg a perifériákra és egyéb szolgáltatások felhasználására vonatkozó kivánságait, az un. "fordítási paramétere"-ket.

Az áthelyezhető bináris loader az eredetitől koncepcióban nem különbözik. A memória térkép kiadása és a betöltéssel párhuzamosan bináris szalag készitése opcionális. Ezek a szolgáltatások a "kezdeti beszélgetés"-ben igényelhetők, illetve letilthatók.

A 8K-D FORTRAN rendszer programjai a DISK MONITOR SYSTEM (DMS) felügyelete alatt a diszkre felkerülhetnek, onnan lehivhatók, és aktivizálhatók. A DMS egy általános file kezelő rendszer. System (S), user (U) és ASCII (A) tipusu file-k nyilvántartását, tárolását és előhozását végzi el. Az S és U tipusu file-októl azt követeli meg, hogy ne legyen ezeknek a programoknak közös része a memória-rezidens MONITOR HEAD-del.

A 8K-D FORTRAN rendszer programjai S tipusu file-ok. A forrásnyelvü programok és a szubrutinkönyvtár A tipusu, az a/b programok pedig speciális A tipusu file-ok.

A fordítót tehát kibővitettük egy speciális file kezelő rendszerrel, amelyik a DMS szabályai szerint adott nevü file-t megnyit inputra vagy outputra. Az inputra megnyitott file-ról 7 bites karaktereket adogat a fordítónak, a fordító által generált 7 bites karaktereket pedig 6 bites pakolt formátumban az outputra megnyitott file-ba lerakja. Vezeti a szükséges nyilvántartásokat. Output terminátor hatására a file-t lezárja.

Az analizátorhoz és a betöltő programhoz tartozó file kezelő rendszer értelemszerüen csak file inputra vonatkozik.

Az analizátor, a fordító és a betöltő programok IN-, OUT- és OPT kérdéseket tesznek fel a felhasználóknak, éppugy mint a többi DMS alatt futó rendszerprogramok. Az IN-re adott válasz specifikálja az input, az OUT-ra adott válasz az output perifériát. Perifériaként megadható gyors vagy lassu papirszalagos egység, illetve a diszken egy-egy file. Az OPT-kérdés valamely opcionális szolgáltatásra vonatkozik, programonként specifikus.

Ez a rendszer a DMS azon programjaival együtt, amelyek feltétlenül szükségesek a 8K-D FORTRAN rendszer müködéséhez, elfoglalja a 32K diszk kapacitás jelentős részét. (A forrásprogram hossza erősen befolyásolja az üresen maradó blokkok számát.) Más rendszerekkel együtt a diszken tárolni csak legalább 64K tárkapacitásu diszk esetén lehet-

séges. A 8K-D FORTRAN rendszer maximálisan 128K kapacitású diszket tud kezelni.

A 8K-D FORTRAN rendszer nagy mértékben lerövidíti a programok fordítási idejét. Gyors, rugalmas szöveg javitási lehetőséggel rendelkezik, a programok kipróbálásának, belövésének ideje is jelentősen csökkenthető használatával.

Irodalom

[1] Lőcs Gyula - Horváth Iván - Nagy Mihály - Stuka Károly - Viszt Éva: A TPA 4K-FORTRAN programozási nyelv.
KFKI, 1970.

[2] Mező István - Viszt Éva: A TPA 8K-FORTRAN rendszer
KFKI, 1972.

[3] Disk Monitor System
KFKI, 1971.

COBOL 1010/B

Sztanev Ivánné

INFELOR

1. A COBOL 1010/B előzményei és célja

A VT 1010/B gépre készült COBOL fordító és értelmező programok rendszere (továbbiakban CBR) az ugyanerre a gépre készült MINICOBOL fordító és értelmező programok továbbfejlesztése.
A továbbfejlesztés a következőkben áll:

- A MINICOBOL oktatási célokra készült, ezért a forrásszöveg is leszükitettebb volt.
- File kezelésnél bővült a file-ok strukturája, több-féle adathordozón jelenhetnek meg.
- Bár a fordítás eredményeként előállitott tárgyprogram szerkezete nem változott lényegesen, mégis a fordító és értelmező programok felépitése teljesen megváltozott, mivel egy fejlettebb szubrutinkezelő és fizikai I/O rendszert használnak fel.

2. A CBR -hez szükséges minimális konfiguráció

- 16K byte tár
- minidiszk
- konzolirógép
- lyukszalag vagy lyukkártya olvasó
- sornyomtató

A CBR fel van készülve a felállitható nagyobb konfigurációkra. (A CBR programjai az 1010/B assembler nyelvén vannak megirva. A deklarációk

megváltoztatásával és ujrafordítással nyerhető a nagyobb konfigurációhoz illeszkedő CBR).

3. A COBOL 1010/B forrásnyelv

A COBOL 1010/B forrásnyelv a standard COBOL nyelv részhalmaza. Megtartottuk a COBOL programok standard strukturáját. A forrásprogram "fejezetekre" (division) tagolódik.

Az IDENTIFICATION DIVISION csak a program nevét tartalmazza.

Az ENVIRONMENT DIVISION csak a SELECT -tel kezdődő mondatot tartalmazza.

A DATA DIVISION -ban a szokásos módon meg lehet adni a file-ok, rekordok és adatok felépitését, hierarchiáját. Az adatokat karakter formában, EBCDIC kódban ábrázoljuk az adatmezőben. Az aritmetikai müveletek is ezeken a karakter formáju adatokon dolgoznak.

A PROCEDURE DIVISION -ban a COBOL nyelv szokásos utasitásait engedjük meg, kisebb leszükitésekkel.

4. A CBR -rel feldolgozható file-ok szerkezete

A file szerkezetek megtervezésénél, ahol lehetett figyelembe vettük a standard file strukturákat.

4.1. A karakter perifériáknál a rekordok

"blokkolatlanok", ami azt jelenti, hogy egy I/O müvelettel egy rekordot lehet mozgatni. A rekordok méretét a fizikai adathordozók tulajdonságai határozzák meg. (Kártya esetén egy kártya, lyukszalag, irógép, sornyomtató esetén egy sor egy rekord).

4.2. A mágnesszalagon a file-ok az IBM standard file-okkal megegyező szerkezetüek, kis korlátozásokkal.

- Standard cimkékkel rendelkeznek
- Egykötetes és többkötetes file-okat engedünk meg.

4.3. A minidiszk eltérő hardware tulajdonságai miatt nem tette lehetővé az IBM standard file-szerkezet alkalmazását.

A minidiszken kétféle file-szerkezetet valósitottunk meg:
- szekvenciális szervezésü,
- indexelt szekvenciális szervezésü file.

Mindkét esetben a file-okra vonatkozó szükséges információk egy katalógusban vannak összegyüjtve.

5. A forditóprogram.

A forditóprogram egy menetben forditja le a forrásprogramot egy értelmezendő tárgyprogrammá. A tárgyprogram két részből áll:
- attributum-tábla
- tárgykódu program.

Az attributum tábla a DATA DIVISION -ban definiált file-nevekhez, rekord-nevekhez, adat-nevekhez rendelt tulajdonságokat tartalmazza, továbbá a literálokat.

A tárgykoduprogram a COBOL utasitásokból előállitott értelmezendő utasitások összessége. Minden COBOL utasitásnak megfelel egy vagy több tárgykodu utasitás.

A szintaktikai és szemantikai elemzésnél felhasználtuk azt a tényt, hogy a COBOL forrásnyelvi szó után egy vagy legfeljebb néhány lehetőség van a forrásszöveg folytatására. A szavak válaszaiként olyan szubrutinokat irtunk, amelyek kiválasztják a megengedett lehetőségek közül a forrásszövegben lévő szót és elágaznak, majd végrehajtják a szükséges szemantikai vizsgálatokat.

Hiba észlelése esetén a forditóprogram folytatja a forditást, ha lehetséges. A hibák listáját a forditás végén adja outputra a tártérképpel együtt.

A tárba forditunk, ezért a forditóprogramot két "szegmensre" kellett bontani. Az első szegmens az első három division forditása után a diszkről hivja be a PROCEDURE DIVISION forditóját.

A forditás közben történik meg az adatterület és az I/O müveletekhez szükséges pufferterületek kijelölése is.

6. Az értelmező-program

A fordítás eredményeként kapott program futtatását egy értelmező-program végzi. A tárgykodu program egy-egy utasitása meghatároz valamilyen tevékenységet, amelyet az értelmező-program egy-egy szubrutinja hajt végre.

Az I/O utasitások végrehajtásával kapcsolatban meg kell emliteni, hogy kihasználják az 1010/B gép nyujtotta megszakitási lehetőségeket, a program futásával párhuzamosan történik meg a fizikai I/O művelet.

Az értelmező-program egésze nem fért el a tárban. Két szegmensre kellett osztani. Az egyik szegmens az I/O utasitások, a másik szegmens a többi utasitás végrehajtásához szükséges szubrutinokat tartalmazza.

7. A CBR "vezérlő" programja

A CBR programjai a diszken helyezkednek el. A vezérlő-program a konzol-irógépről kapott parancsok alapján behozza és elinditja a CBR különböző programjait.

Ezek a programok:
- A fordító
- Az értelmező program
- File allokáló a diszken
- Mágnesszalag cimkéző
- Diszk katalógus kiirató

- Diszk katalógus törlő
- Tárgyprogram kiíró (dump)
- Tárgyprogram betöltő

8. A CBR előnyei és korlátai

A CBR lehetővé teszi, hogy az 1010/B gépen a COBOL nyelven megirt adatfeldolgozási feladatokat forditani és futtatni lehessen.

A 1010/B -hez illeszthető perifériák bármelyikén megjelenhetnek az adatok. A file-szerkezetek alkalmazkodnak a standard file-szerkezetekhez, ha ez lehetséges.

A TPA 1001-I ÜGYVITELI ADATFELDOLGOZÓ RENDSZERE

Buday László - Görbe Tamás

Központi Fizikai Kutató Intézet

Napjainkban a kisszámitógépek egyre jobban elterjednek az adatfeldolgozás területén. Kisebb intézmények, üzemek céljaira sok esetben még alkalmasabbak is, mint a közepes teljesitményü számitógépek.

Ennek legfőbb okai:

- Alacsony beruházási költségek.
- Kisszámitógépen a feladatok futási ideje nem lényegesen hosszabb, mint a közepes teljesitményü gépeken, a gépidő viszont lényegesen olcsóbb.
- On-line beavatkozási, javitási lehetőségek épithetők ki.

Ilyen meggondolások alapján dolgozta ki csoportunk a TPA kisszámitógép ügyviteli adatfeldolgozó rendszerét, amelynek vázlata a következő ábrán látható:

I. Filekezelő rendszer
   1. Adatelőkészités
      a/ CHANGE
      b/ COPY
      c/ SORT
   2. Jelentéskészités
      a/ SELECT
   3. Filenyilvántartás
      a/ LIST
      b/ DELETE
      c/ RENAME

} On-line hozzáférés

II. Egyszerü ügyviteli nyelv
   1. MINIBOL compiler
   2. MINIBOL operating system

A fenti programrendszer müködését a DISK MONITOR SYSTEM koordinálja.

Hardware igények:

TPA 1001-I Központi egység 8K szó /12 bit/ memóriával /32K-ig bővíthető/
Teletype irógép
Gyors szalagolvasó- és lyukasztó
Mágneslemezes tároló /32-128K szó , ill. 800 Kbyte/

Bővítési lehetőségek:

Sornyomtató
Kártyaolvasó- és lyukasztó
Mágnesszalagos egységek

Adatszervezés

Az adatfeldolgozó rendszerrel kezelt legnagyobb információs egység a file. A file egymáshoz rendelt rekordok halmaza. A rekord adatokból áll. Azokat az adatokat, amelyek a feldolgozás szempontjából lényeges szerepet játszanak, kulcsoknak nevezzük. File-jaink kulcs-szekvenciálisan rendezettek.

A rekordok tervezésénél három lényeges szempontot kell figyelembe venni:

a/ Tárolóigény minimalizálása.
b/ A rendszer módositása egyszerü legyen.
c/ A futási idő minimalizálása.

Ezek között optimimra kell törekedni.
Egy file állhat azonos tipusu, fix hosszuságu, vagy különböző tipusu, változó hosszuságu rekordokból. Az azonos tipusu rekordokat tartalmazó file több tárolóhelyet igényel, a különböző tipusu rekordokat tartalmazó file feldolgozása viszont nehézkesebb.

A filekezelő rendszer csak azonos tipusu rekordokat tartalmazó file-okat tud feldolgozni, a MINIBOL nyelv segitségével azonban a különböző tipusuak feldolgozása is egyszerüen megoldható.

I. Filekezelő rendszer

Az adatkezelő müveletek előtt rekordleirást kell adnunk a felhasználni kivánt file-okról. Itt kell megadnunk az adatok nevét, tipusát és hosszát, a COBOL picture leirásával azonos formában.

Ha a program bármilyen szintaktikus hibát /a rekordleirásnak nem megfelelő karakterek/, ill. szemantikus hibát talál /kontrollösszegek különbözősége/, hibaüzenet után ezt a rekordot kiirja a Teletype-on, s ugyanakkor kihagyja a további feldolgozásból. Ezzel eleget teszünk a következő követelményeknek:

- A hibát már fellépéskor kiküszöböljük.
- Néhány hiba kedvéért nem tartjuk vissza a teljes adatfeldolgozást.
- A hiba kijavitásának időtartamára nem tartjuk fel a számitógépet.
- A gép irásos feljegyzést készit a hibáról.

A javitott rekordot pótlólagosan kell feldolgozni.

1. Adatelőkészités

a/ CHANGE

A file adatainak javitását végzi. Egy törzs- és egy javitófile-ot kell megneveznünk. A javitófile tartalmazza a rekord azonositásra és a rekord javitásra vonatkozó információkat.

b/ COPY

Lényegében általános input-output program. Segitségével adatfile-okat az egyik perifériáról a másikra áttehetünk, egybemásolhatunk, az adatok formátumát megváltoztathatjuk, egyes adatokat elhagyhatunk, vagy uj adatoknak helyet foglalhatunk.

c/ SORT

File-ok megadott kulcsok szerinti rendezésére szolgál. A rendezés tipusa növekvő vagy csökenő lehet.

2. Jelentéskészités

a/ SELECT

Az adott input file-ból bizonyos kulcsok alapján kiválasztott rekordokat kiir az output file-ra.

3. Filenyilvántartás

a/ LIST

A háttértárolón lévő adatfile-ok tartalomjegyzékszerü kilistázására szolgál.

b/ DELETE

A háttértárolón elhelyezett adatfile-t törli a nyilvántartásból.

c/ RENAME

Adatfile-ok átnevezésére szolgál. Ha ragaszkodunk ahhoz, hogy egy file a különböző feldolgozások során ugyanazt a nevet viselje, ezt a funkciót használjuk.

II. A MINIBOL nyelv

A MINIBOL nyelv /MINI COBOL/ a COMPACT COBOL TPA-ra való alkalmazása. A MINIBOL nyelv elég sok tekintetben eltér az általában szabványként elfogadott specifikációktól.

Az eltérés okai a következők:

a/ A COMPACT COBOL - mivel egy lényegesen bővebb nyelv kivonata - annak teljes szerkezetét megtartotta, s igy bizonyos szerkezeti egységek jelentősége erősen lecsökkent. /pl. Azonositási Főrész, Környezetleiró Főrész/

b/ A MINIBOL Compiler és Operációs rendszer szervesen kapcsolódik az on-line File-kezelő rendszerhez. Igy a MINIBOL-nak bizonyos funkciókat nem kell ellátnia /pl. rendeztetés/, viszont kompatibilitási problémák merülnek fel.

c/ Maximális egyszerüségre törekedtünk.

A MINIBOL és a COMPACT COBOL alapvető eltérései

A COBOL nyelvet, ill. a COMPACT COBOL-t ismertnek tételezzük fel, a hely korlátozottsága miatt csak az eltérések felsorolására szoritkozhatunk.

Alapvető szerkezeti egységek tekintetében:

a/ Az Azonositási és a Környezetleiró Főrész hiányzik.

b/ Az Adatleiró Főrész csak a Working Storage Section-ból áll. A File Section speciális feladatait az OPEN, READ és WRITE utasitások veszik át.

Az Adatleiró Főrészen belül:

a/ A szintszámok használata nem megengedett. Igy csak egyedi adatokat és egymás mellé rendelt elemi mezőkből álló rekordokat definiálhatunk.

b/ A REDEFINES utasitás csak rekordokra adható ki. Ezzel kiküszöbölhetjük a szintszámok hiányából adódó nehézségeket.

c/ Csak DISPLAY tipusu adatok vannak megengedve.

Periféria specifikálása, file sezrkezet:

a/ Mágnesszalagon egy blokk csak egy rekordot tartalmazhat.

b/ Csak soros szervezésü file-ok használhatók. A mágneslemezes tárat a mágnesszalaggal azonos módon kezeljük. Ez ilyen kis diszknél megengedhető.

c/ Csak standard cimkét lehet kiadatni, de ezt bármilyen input vagy output perifériára vonatkozóan. A standard cimke csak ellenőrző összegeket tartalmaz. A file cimkézettségére vonatkozó állitást az OPEN utasitásban közöljük.

A perifériális adatforgalom esetében:

a/ MINIBOL minden beérkező adatot azonnal - kérés nélkül - szintaktikusan ellenőriz, vajon megfelel-e a hozzá tartozó picture-nek.

b/ A szintaktikus ellenőrzés, s a különböző tipusu rekordok bevezetésének követelményei miatt a rekord első karakterének a rekord tipusára kell utalnia. /Emiatt különbözik a READ utasitás a COBOL-beli megfelelőjétől./

c/ LABELLED input file-ok esetében a MINIBOL az ellenőrzést szemantikusan a kontrollösszegek segitségével is elvégzi. LABELLED output file esetében pedig kiirja a kontrollösszegeket.

5. Az Eljárási Főrészben:

a/ A COMPUTE utasitás végez el minden aritmetikát igénylő müveletet.

b/ Az IF utasitásban a feltétel teljesülése esetén csak GO TO utasitást irhatunk elő.

Gépi megvalósitás

A MINIBOL nyelvet metanyelvi egyenletekben leirtuk, ennek alapján szintaktikus táblázatokat készitettünk. A SYNTAX DRIVER nevü igen rövid program a táblázatok alapján a beolvasott forrásnyelvi szöveget értelmezi és kódsorozatot állit elő.

A compilernek a szintaktikus egyenletek alapján való megirása nagymértékben lecsökkentette a programozási időt, hiszen a hibakeresést elegendő volt csak a szintaktikus egyenletek szintjén végezni.

Az előállitott kódsorozatot a MINIBOL Operációs rendszere értelmezi. Mivel a gép eléggé gyors /1,5 us ciklusidő/, a kódok értelmezése miatt fellépő időveszteség még nem lassitja a perifériák müködését.

Egy MINIBOL program maximális hossza kb. 400 utasitás lehet.

A MINSZK-22 ALGOL-60 FORDITOJÁNAK (TAM) FEJLESZTÉSEI A JATE KIBERNETIKAI LABORATORIUMÁBAN

Makay Árpád - Matievics István - Máté Eörs - Fülöp József - Diamant Tibor
JATE Kibernetikai Laboratórium

Bevezetés

A számitógéppel szállitott TAM néhány hiányossága, a géphez illesztett uj perifériák, a vásárolt uj adatelőkészitők és az operációs rendszer bevezetése indokolttá tették a fordító néhány szegmensének kijavitását, illetve kicserélését. A változtatások részben magára a fordító programra, részben a MINSZK-ALGOL-ban használatos standard-ekre (szubrutinokra) vonatkoznak.

1. A compilerrel kapcsolatos változások

1.1. A TAM első szegmensének feladatai a javító és a forrásnyelvi program beolvasása, a javitott program kialakitása, esetleg lyukszalagra és szélesnyomtatóra történő kiirása, átkódolás 9 bites kódra a fordító további szegmensei számára. A tevékenységeket az eredeti szegmens müködési sorrendjének megfelelően soroltuk fel. Célszerünek látszott a perforálás és a nyomtatást az átkódolás után végezni, mivel igy lehetőség nyilott, hogy a javitott programokat már a 9 bites kódban olvassuk be, illetve az összetett alapjelek ebben a fázisban könnyebben felismerhetők, igy ezeknek az aláhuzott *formában történő* nyomtatása egyszerübben programozható. Az első szegmens átirása a következőket tette még lehetővé: a forrásnyelvi program 9 bites kódban is beolvasható, a javitást ASR-33 konzolirógéppel is megadhatjuk, forrásnyelvi könyvtárat alakithatunk ki. Az első szegmens átirása a forditási idő csökkenését eredményezte.

1.2. A fordító eredetileg 4K-s ferritmemóriára készült, ez indokolta a standardok interpretáló rendszerban való működését. Ezáltal hosszabb programok irása vált lehetővé, de a célprogramok futási ideje jelentős mértékben megnyult.

A fordító átalakitása során az interpretálás lehetőségét megtartva lehetővé tettük a standard-ek compilálását a célprogramhoz. Az interpretáló technika mellett a belső szubrutin lehetősége érthetően nem volt adott, a compiláló technika gazdaságos alkalmazásához feltétlenül szükség volt.

2. Az eredetileg elég szegényes szubrutinkönyvtárat ujabb standardekkel egészitettük ki. Ezek közül a legfontosabbak: a mágnesszalagozó és a file kezelő standard-ek, lineáris egyenletrendszer, differenciál egyenletrendszer, magasabb foku egyenlet megoldó, numerikus integráló, statisztikai szubrutinok stb.

3. A fordító program és a standard-ek operációs rendszerhez illesztésével elértük a kezelés egyszerüsitését, a hibalehetőségek csökkentését, a dokumentált futtatás lehetőségét.

Az átalakitással kapcsolatos tapasztalataink kedvezőek.

AZ ALGOL 68 MEGVALÓSITÁSÁNAK EGYES KÉRDÉSEI

Náray Miklós
NIM IGÜSZI Számológközpont
Software-fejlesztési Osztály

A számológépek fejlődésével párhuzamosan a programozási nyelvek is fejlődnek. Az egyedi feladatok megfogalmazását megkönnyitő nyelvek /szimulációs nyelvek, adatkezelő nyelvek, forditóprogram-leiró nyelvek stb./ és a számológépek párbeszédes üzemmódban való használatát lehetővé tévő nyelvek /APL, BASIC stb./ kialakulása mellett a két fő alkalmazási terület, a kereskedelmi és a müszaki-tudományos számitások általános célu programozási nyelvei is /COBOL, FORTRAN, ALGOL 60/ fejlődtek. E fejlődés eredményeképpen jött lére - mint ismeretes - a PL-1 nyelv is, a COBOL és FORTRAN egyesitéseként és bővitéseként, átvéve néhány ALGOL 60 lehetőséget is.

Természetesen ezek a változások az ALGOL 60 nyelvet sem kerülték el. Definiálták és bevezették az ALGOL 60-ba az un. "case" fogalmát, amely egyaránt alkalmazható utasitások és kifejezésekkel kapcsolatban, valamint lehetővé tették a változók értékének bitenkénti elérését és értelmezését is. Pótlólag megadták a szabványos beviteli- és kiviteli eljárásoknak az ALGOL 60 jelentésből hiányzó definicióit. Több javaslat született arra is, hogy string tipusu változók ALGOL 60 programokban kezelhetők legyenek.

Az ALGOL 68 nyelv nem az ALGOL 60 fenti módon való bővitgetésével keletkezett, bár létrehozásának kiindulópontja kétségtelenül az ALGOL 60. Az ALGOL 68 nyelv minden olyan lehetőséget megenged, amely az ALGOL 60-ban is meg van engedve, de az ALGOL 68 lényegesebben több eszközt ad a programozó kezébe.

Az ALGOL 68 az ALGOL 60 bizonyos fogalmait általánositja, de emellett ujakat is vezet be. A nyelv konstrukciója olyan, hogy a definiált fogalmak használata minden olyan lehetséges helyzetben meg van engedve, ahol értelmes jelentés tulajdonitható nekik. Ezért a nyelv a viszonylag kevés fogalom ellenére rendkivül hatékony. Ez a nyelv ortogonális tervezésének köszönhető.

Az ALGOL 68 ugy definiálja az érték fogalmát, hogy értéknek nevezi az un. neveket is, amelyek további értékre utalnak. Az utaláslánc vége nyilván olyan érték, amely nem név. A nem-név értékek tulajdonságait az un. mód határozza meg, ami az ALGOL 60 tipus fogalmának az általánositása. Az ALGOL 68-ban végtelen sok mód létezik, a programozónak is megvan rá a lehetősége, hogy uj módokat hozzon létre a programjában un. móddeklaráció segitségével.

Az uj módokat a már léteziőkből lehet létrehozni. Ilymódon lehet definiálni a többszörös értékek módját /az ALGOL 60 tömbfogalmának általánositása/, a szerkesztett értékek módját és az egyesitett módokat. A többszörös értékek deklarálhatók rugalmas határokkal is, ekkor az elemszám a program végrehajtása közben dinamikusan változhat. A nyelv által definiált módok, amelyekből az uj módok definiálása kiindul, lehetővé teszik az egész, valós, logikai, karakter, formátum, eljárás, komplex, bitek, szósejtek, füzér, szemafor, telep módu értékek deklarálását. Attól függően, hogy valamely érték a programban hol fordul elő, az érték módja meghatározott szabályok szerint automatikusan a környezet által kivánt móddá alakulhat. /Pl. valósból komplex./ Ez az un. kényszeritás.

Az ALGOL 68-ban nemcsak uj módok, hanem uj müveletek deklarálására is van lehetőség. A deklarált uj operátor jelének,/müveleti jelnek/ nem kell feltétlenül az összes többi már létező operátor jelétől különböznie, az operátor azonositásakor ugyanis szerepe van az

operandusok módjának is. Egy formulában előforduló operátor mindig azt az operátort azonositja, amelynek a deklarációjában szereplő operandusok módja megegyezik a formula operandusainak módjával, vagy abból kényszeritheto.

Az ALGOL 68 nyelv egyik leghatékonyabb eszköze az azonosságdeklaráció. Magába foglalja az ALGOL 60 tipusdeklarációját, tömbdeklarációját, kapcsolódeklarációját és eljárásdeklarációját, de ezeken kivül lehetővé teszi bármilyen módu konstansok deklarálását is és az eljárások paraméterátvételi mechanizmusának alapját képezi.

Az ALGOL 68 jelentés az un. kötött nyelvet definiálja, de megad un. bővitéseket is, amelyek lehetővé teszik, hogy bizonyos programrészeket más - egyszerübb - alakban is felirhassunk. Ezeknek a bővitéseknek a segitségével vannak definiálva /az ALGOL 60 for-utasitásánál tömörebb és hatékonyabb/ un. ismétlőutasitások és az esetmondatok /ALGOL 60-ban a "case"/.

Egy ALGOL 68 forditóprogram készitésekor felmerülő kérdések egy részének megoldásában alkalmazhatók azok a módszerek, amelyek általában egy forditóprogram elkészitésekor használatosak. Vannak azonban olyan általános lehetőségek az ALGOL 68-ban, amelyek különleges, uj módszerek keresését és felhasználását igénylik. Ilyenek például:

- rugalmas tömbök /speciálisan a füzérek/,
- generátorok,
- módazonositás,
- operátorazonositás,
- kényszeritések.

A nyelv definiciójának közzététele után több helyen elkezdték a megvalósitást. E munkálatok során a fenti problémákra születtek megoldások,

bár a megvalósitók egy része csak résznyelv megvalósitását tüzte ki célul, és a résznyelvet ugy alakitotta ki, hogy az olyan fogalmakat, amelyek megvalósitása nagyon nehéznek látszott, kizárta a nyelvből.

Jelenleg egyetlen müködő megvalósitásról tudunk, amely egy ICL 1907F számológépre készült Angliában, a Royal Radar Establishment-nél, de ez csak egy szük résznyelvet valósit meg.

Több helyen dolgoznak azonban a teljes nyelv - vagy kismértékben korlátozott résznyelv - megvalósitásán, és várható, hogy a közeljövőben több számológépen is elérhető lesz az ALGOL 68 nyelv forditóprogramja.

I R O D A L O M

A. van Wijngaarden /Ed./, B.J. Mailloux, J.E.L. Peck and C.H.A. Koster /1969/, Report on the Algorithmic Language Algol 68, Numerische Mathematik 14, 79-218.

Az ALGOL 68 szemantikájának szerkezeti felépitése /P. Branquart, J. Lewi, M. Sintzoff, P.L. Wodon/, a NIM Ipargazdasági- és Üzemszervezési Intézet Számolóközpontjának kiadványa /forditás/, 1971.

Bedő Árpád: Algol 68, a matematikusok programozási nyelve, Számológép, 1971, 3. szám, 89-91. old.

A NAGYÜZEMI SOFTWARE-GYÁRTÁS PROBLÉMÁINAK NÉHÁNY PÉLDÁJA A FUJITSU LTD -vel VALÓ MUNKA-KAPCSOLATUNK ALAPJÁN

Bakos Tamás - Dettrich Árpád
INFELOR

A FACOM-R kisszámoló gép a Fujitsu LTD "minikomputere", amely a 230-as rendszertől független. Amikor tavaly hazánkban megjelent, /Számitástechnikai Koordinációs Intézet-Müszaki Egyetem/ akkor a meglehetősen nagy konfiguráció mellé egyedi /egymástól független/ programok érkeztek. Ezzel egyidőben megrendelést kaptunk a Fujitsu LDT-től egy moduláris programozási rendszer kialakitásának első lépéseire.
Ennek keretében elkészült egy modul strukturáju assembler, egy makro preproceszor, egy linkage-loader, makro könyvtár generátor, szubrutin könyvtár generátor.

A részek leirása

Az assembly-nyelv szintaxisát a megrendelő ugy irta elő, hogy annak a korábbi assembly-nyelv része legyen. A direktivákat kellett a modul strukturának megfelelően változtatni, illetve ujakat felvenni. Az ilymódon definiált nyelven megirt programnak egy-egy sora - a kommentektől eltekintve - 16 jelet, illetve jelcsoportot alkalmaz. Ezért egy olyan elemző automatát készitettünk, amelynek tevékenységét egy mátrix-szal szemléltethetjük. A mátrix oszlopait rendeljük hozzá az automata egyes állapotaihoz, mig a sorokat a jelekhez. A mátrix elemei tevékenységek, minden elem legfeljebb négy tevékenység egyikét jelentheti. Összesen kb. 80 tevékenységre volt szükség.
Ezek után az állapotjelzőt a szintaktikus elemzés megfelelő poziciója állitotta be, és az input pufferből beolvasott jel segitségével iniciáltuk a fenti mátrixból adódó tevékenységet.

Az assembler két menetben dolgozik. Az első menetben felismeri és szótárban elhelyezi a címkéket, amelyek valamilyen deklaráló direktiva után szerepelnek, vagy az utasitások elé kitéve. Ezen kivül az egyes utasitásokról megállapitja azok relativ helyét a modul kezdetéhez számitva.
A második menet az igy kialakitott cimkinformációk alapján elvégzi a forditást, kialakitja a tárgymodult. A tárgymodul a tárgyfejből és tárgytörzsből áll. A fejben a modul nevét, a hosszát és a hozzá deklarált kommonmező hosszát találjuk, továbbá azon cimkék nevét és modulrelativ cimét, amelyekre más modulokból lehet hivatkozni /globál cimkék/.
Felsoroljuk még azokat a cimkéket is, amelyek más modulokban vannak definiálva, és az adott modulban hivatkoznak rájuk /external cimkék/.
A törzsben a gépi utasitásokból és konstansokból 16 bites bináris konstansok lesznek. A DC direktivában előforduló cimkifejezések: modulhoz relativ, kommon relativ, és external cimeket tartalmazhatnak. Ezeknek és a forditóprogram utasitásszámlálóját módositó ORG direktiváknak egy-egy tárgynyelvi, un. loader direktiva felel meg.
A tárgymodul lyukszalagon, lyukkártyán vagy diszken jelenik meg, továbbá a megfelelő listázás a sornyomtatón, a felhasznált cimkék listájával a végén.
A makro preprocesszor az elemzés fent leirt eszközét használja fel, és két üzemmódban dolgozik.
Az első a makrodefinició tevékenysége, amikor a makro nevét elhelyezi a makro szótárban és kikésziti a makro választ, amelynek bevezető információi mellett egy szótárban megtaláljuk a formális paraméterek neveit, és tulajdonságait /pozicionált, illetve kulcsszavas/ és kulcsszavas paraméter esetében az alapfeltételezést is, ha ez nem üres. A továbbiakban a makrotörzs karaktereit

rakjuk le, nem törődve azok értelmével. A makro válaszokat a diszken tároljuk.
A második fázis, a helyettesités fázisa, amelyben a felismert makronév után egy verembe helyezzük a makroválaszt, s utána bemásoljuk az aktuális paramétereket. Ezután soronként generáljuk a makrotörzset és ahol formális paramétert találunk, ott az aktuális értéket behelyettesitjük. A veremre azért van szükség, mert a törzs sorai között is lehetnek ujabb makrohivások.

A linkage-loader két részből áll. A szerkesztő és a betöltő rész. A szerkesztő a tárgymodulok fejeit beolvasva összeállitja a globális nevek szótárát, elkésziti közben a tár felosztást is.
Ezután a betöltő a tárgymodulok törzsét olvasva a loader direktiváknak megfelelően vagy változatlanul elhelyezi a beolvasott információt, vagy módositja azt a programbázis, illetve a kommonmező bázisának megfelelően, vagy ha externál, kikeresi a szótárból a neki megfelelő cimet. Ebben a tevékenységben a szerkesztőt felül irhatjuk, a betöltőt nem, legfeljebb a kommommezővel.

A makro-generátor tulajdonképpen ugyanazt a tevékenységet végzi, mint a makrodefinicios rész, csak hozzá kell tenni olyan tevékenységeket, amelyek a könyvtár betoldását, törlését, listázását, stb. lehetővé teszik.

A szubrutin-könyvtár-generátor tárgymodul formájában lévő szubrutinokat kétfelé választ. A fejeket összegyüjtve elhelyezi a linkage-loaderben, mig a törzseket kimásolja lyukszalagra, vagy lyukkártyára. Ez a szétválasztás csak a szerkesztés meggyorsitása kedvéért történik.

A munka megtervezése és megszervezése

Első lépésben megalakitottuk azt a team-et, amely két rendszertervezőből állt, 3 senior - 4 junior programozóból. A rendszertervezők definiálták az egyes részek közötti interface-ket, majd elkészitették az egyes részek pontos tervét, blokkdiagram mélységig.
A programozók ezen idő alatt megismerkedtek a géppel, a vele szállitott software-vel. A következő lépésben kettős ágon haladtak. Az egyik ág a MINSZK-22-re készitett egy szimulátort, egy már korábban kiépitett kezelőrendszerbe ágyazva. A másik ág a Facom-R-en összegyüjtötte a FUJITSU által szállitott egyedi programokat, kiegészitette azokat a program kipróbáláshoz szükséges további programokkal, és ezeket egy egységes monitor rendszerbe foglalta. Ez a rendszer felhasználja a diszket és az egyes elemek operátori paranccsal konzolirógépről aktiválhatók.

A programok elkészitése és próbája

Miután a rendszertervet a megrendelőnek bemutattuk, megjegyzéseivel kiegészitve átadtuk a programtervezőknek, és programozóknak. A programtervezők a blokkdiagram áttanulmányozása után elkészitették a részletes blokkdiagramot, ott, ahol az szükséges volt, s megvitatták a programozókkal együtt a programokat. Ezzel párhuzamsoan az egyes programok pontos specifikációja alapján egy két főből álló csoport elkészitette az egyes programok tesztprogramját /mintegy 100 programot/.

Befejezés

A fenti módon megtervezett és összeállitott programokat a hozzá készitett tesztprogramokon kivül a régi assembly-nyelven megirt és leforditott programokkal kipróbáltuk

és végül az uj assemblert modulokra bontva saját magát is leforditottuk. Ezek után küldtük el a megrendelőnek, ahol egyhónapos ellenőrzési procedurának vetik alá a munkánkat.

KISGÉPEK ASSEMBLERÉNEK VIZSGÁLATA

Sánta Lórántné
INFELOR

Bevezetés

Intézményünknél az elmult években mind belső kutatási munka, mind pedig megrendelés keretében (EMG, VIDEOTON, FUJITSU) több különböző szintü és felfogásu assembler készült kis- és középgépekre. Ezekről - és a készitésük közben szerzett tapasztalatokról - külön előadások keretében fognak készitőik beszélni; a következőkben az assemblerek /azaz a géporientált nyelvek fordítóprogramjainak/ kialakulásáról, az assembly nyelvek strukturájáról, az assemblerekről általában lesz szó, utalva az általánositási lehetőségekre is.

Az assemblerek kialakulása

Már a számitógépek megjelenésének első időszakában is voltak olyan törekvések, hogy a gépi objektumokat /regiszterek, tárrekeszek, perifériális készülékek/ szimbólumokkal jelöljék meg a program irásakor. Az első lépésnél még a programozó tervezte meg a tárfelosztást: pl. sorszámmal ellátott blokkokra osztotta a programját, és az oktálisan, decimálisan, vagy hexadecimálisan irt utasitásokban ezen blokkszámokra, illetve blokkon belüli relativ cimekre hivatkozott. A következő lépés volt a blokkszámoknak szimbólummal való helyettesitése, pl. az abc betüivel.

Amikor az input eszközök lehetővé tették, hogy a hétköznapi értelemben használt teljes karakterkészletet fel lehessen használni a számitógéppel való információközlésre, minőségi változás állott be az assembly nyelvek szintaxisa területén, mivel a gép valamennyi objektumára egy-, később több-karakteres szimbólummal való hivatkozásra nyilt lehetőség - beleértve pl. az utasitáskód, vagy cimzési mód jelölését is.

Az assembly nyelvek szerkezete

Miután a gépi objektumok szimbólikus jelölésének transzformációját egy fordító programra (assembler) bizhatták, természetesen adódott, hogy a tárfelosztás egyéb funkcióit is kivegyék az assembly nyelven programozók kezéből.

A legegyszerübb assembly nyelveknél a program egyetlen összefüggő egész, amelyben csak az egyszerre leforditandó program objektumaira és a hardware - regiszterekre való hivatkozások szerepelnek. Ezeket a belső hivatkozásokat ugy szervezik, hogy az utasitásokat, adatmezőket, vagy azokon belül egyes elemeket szimbólikus cimmel, cimkével jelölik meg. Az utasitások cimkézése önmagáért beszél, mig az adatmezők cimkézésének több formája alakult ki:

- Az egyik változatban a programozó szimbólumokkal azonosithatja az adatelemeket és adatmezőket, és az assembler rendel ezekhez tárterületet (változók deklarációja). (ELLIOTT 4100 NEAT, MINSZK-22 MISI nyelve).

- Másik lehetőség, mely az ujabb assmeblerekre jellemző, hogy a gépi utasitásokhoz hasonló direktivák segitségével a programozó az általa tervezett helyen jelöl ki adatelemeket vagy adatmezőke- esetleg kezdőérték megadásával egyidejüleg. (Az előbbi két lehetőség keverve is előfordulhat.)

Egyes assembly nyelvekre a blokkstruktura a jellemző, megengedve a blokkok egymásba skatulyázását is. Ennek a megoldásnak az a jólismert előnye, hogy egy belső blokkban ujra definiálhatjuk a külső blokkban is használt neveket. Ez a módszer volt az első lépés a moduláris programirás megoldására. (Ilyen pl. az ELLIOTT 4100 NEAT nyelv és az EMG 830-as gépre készült SIMPLE).

Miután hardware-területen előtérbe került a modularitás kérdése, ez az assembly nyelvek fejlődésére is hatott, hiszen az az egyetlen nyelv, ami a hardware-t követni engedi. A modul-strukturáju programirás alapgondolata a következő:

A programot különálló modulokból épitjük fel. Minden modul önmagában zárt és csak bizonyos speciális eszközök felhasználásával nyilik lehetőség a modulok közti információcserére. Az információcseréének két tipusáról beszélhetünk: vezérlésátadás, és adatok átadása.

A vezérlésátadásra az externál-globál (vagy entry) cimke-kapcsolatot használják. Az adatok átadásának többféle módja van: egyrészt a hardware lehetőségeire támaszkodó paraméterátadással, másrészt

kommonmező kijelölésével, harmadrészt pedig nyelven kivüli elemek segitségével (file-ok).

Az assembly nyelvü program olykor mereven tükrözi a végrehajtható program strukturáját, az egyedi hardware-lehetőségekkel (bázisregiszterek, cimzési módok stb.) rendelkező gépek esetén (MITRA 15 gép MITRAS nyelve).

Az assembly nyelvü programok strukturája után a programok belső felépitéséről néhány szót:

egy assembly-nyelvü program "utasitás"-okat tartalmazó, egymástól függetlenül fordítható "sorok"-ból áll (egyes assemblerek megengedik, hogy egy logikai sor több fizikai sorban nyerjen elhelyezést).

- Egy (logikai) sor feldolgozása során általában egy gépikódu utasitás keletkezik,
- az assembler szerkesztési tevékenységéhez azonban szükség van olyan utasitásszerü információkra (direktivákra), melyek a leforditott programban közvetlenül nyomot nem hagynak, csupán az assembler állapotát változtatják meg.
- Előfordulnak még továbbá adatokat definiáló utasitások is.

Az gépi utasitásoknak megfelelő sorok különböző rendeltetésü "mezők"-ből állnak; ezek a következő sorrendben követik egymást (kötött- vagy szabad formában): cimke-, utasitáskód-, operandus- és megjegyzés -mező.

- A cimkemező lehet üres (olykor kötelezően az), ill. tartalmazhat egy (szabad forma esetén több), az utasitássort azonositó cimkét.

- Az utasitáskód poziciójában találjuk azokat a jeleket, amelyek megkülönböztetik a tényleges gépi utasitásokat az assemblernek szóló direktiváktól, vagy azoktól a direktiváktól, amelyek adat-konstansokat definiálnak, ill. munkaterületeket deklarálnak. Itt találhatjuk meg, az un. makroutasitások neveit, valamint esetlegesen az assembler számára feltételes és ismétlő tevékenységeket előiró pszudó kódokat is (melyek pl. rendszergenerálásnál előnyösen alkalmazhatók).
- Az operandus részben cimkékből, változónevekből kialakitott cim-kifejezések, vagy konstansok szerepelnek - a hardware által megengedett cimzési mód előirásával (indirekt, indexelt stb.) együtt (bár olykor ezek mint az utasitáskód módositási szerepeltethetők). Gyakori a konstansokra un. literál formában való hivatkozás.

  A deklaráló utasitásoknál ez a felépitès nem mindig következetes, olykor az operandus részben egy listát találunk (esetleg a mező tipusának előirásával és ismétlési tényező alkalmazásával).

  Egyes nyelvekben a programozónak módja van az ismétlődő tevékenységek leirásához (rendszer-) könyvtárban, valamint az assembler és/vagy saját maga által definiált, nyitott vagy zárt szubrutinokat használni, melyek a fordités során - vagy előtt (pl. makro preprocessor), a szerkesztés, betöltés vagy futás során lesznek

a programhoz (statikusan vagy dinamikusan) hozzácsatolva.

Az assemblerek szerkezetéről és müködéséről

Az assemblerek általában egy-, két-, esetleg többmenetesek; az első menet a cimkéről gyüjt információt (szimbólumtálbák), a második menet a tárgyprogramot állitja elő, az esetleges harmadik menet egyéb információkat szolgáltat a programozó számára.

Az assembler outputja egy félig-forditott un. tárgymodul; a kivént modulokat egy szerkesztőprogram (Linkage Editor) végrehajtható programmá dolgozza össze, melyet egy betöltő program (Relocating Loader), vagy esetleg önmaga azonnal (Linking Loader) tölt be a tárba. (A load and go assemblerek elvégzik a Linking Loader funkcióját is.)

Összefoglalva: az assembler fő tevékenysége egy forrássor jeleinek felismerése, a jelekből a szintaktikus egységek összeállitása és végül az egyes utasitásoknak megfelelő tárgykód összeállitása. Eközben a program strukturájáról, változóiról, cimkéiről olyan információkat gyüjt össze, melyeket mintegy járulékos információt a tárgyprogramhoz csatol.

A cimkékkel kapcsolatban a postdefinit hivatkozások feloldásánál különféle ötletes megoldások ismeretesek.

Befejezés

Az assemblert tervező számára ésszerü általánositás ha az assembler géptől független részeivel operál, a hardware-függő részeket külön rutinként csatolva ezekhez, majd e különálló részeket az assembly nyelv sajátosságainak figyelembevételével kapcsolja össze.
Már az assembler készitőjének munkájára is kihat az előbbi megoldás, ha a feladat: egy gépen több gép assemblerét kell kidolgoznia, megfelelő szimulált környezetek számára.
Az egy gépre készitendő különböző forditóprogramok tervezése és irása esetén is gazdaságos ezen elv (közös tárgykód, stb.) megvalósitása.

Az előbbi törekvéseket tükrözik a különböző assemblereket generáló programok (META/MASTER) és lényegében a makro-processorok is.

VIDAS (A VT-1010/B SZÁMITÓGÉP VIDOS OPERÁCIÓS RENDSZERÉNEK AZ ASSEMBLERE)

Mandler György
INFELOR

1. Bevezetés

Az eddigiekben több előadás elhangzott, melynek témája a VIDOS (VIdeoton Data Orientest System) volt. Ezért ebben az előadásban, melyben a VIDOS rendszer assemblerérőI számolunk be, feltételezzük a rendszer felületes ismeretét.

2. Az assembler lényeges vonásai

2.1. Az assembler modul-fordító, azaz az egyes program modulok egymástól függetlenül fordithatók le. A leforditott tárgy-modulokat egy szerkesztő-betöltő program állitja össze és helyezi el a tárban.

2.2. Az assembler segitséget ad az értelmező szinten való programozásra, de a gépikód szintü programok irását is lehetővé teszi.

2.3. Az assembler megkönnyiti a rendszer szolgáltatásainak alkalmazását.

2.4. Az assembler egyaránt alkalmas rendszer-programok valamint felhasználói programok leforditására.

3. Az assembler kiépítésének a fokozatai

A kiépitésnek három szintjét képzeltük el, mely szintek mindegyike részeként tartalmazza az előzőt, Az első szint elkészült, a második szint belövése folyik, a harmadik szint programtervezési stádiumban van.

3.1. Az alapszint (VIDAS1)

- Lehetőséget ad gépikód szintü programok irására. A felhasználó szimbólikusan hivatkozhat a hardware utasitásaira. (Gépikód mnemonikok, valamint módositók a shift, skip és regiszter transzfer utasitásoknál). A programok önálló modulokból állithatók össze.

- A modulokon belül további kétféle szegmentálási lehetőség van, a mezőkre bontás, valamint a szubrutinokra osztás. A mezőkre bontás azt jelenti, hogy a programozó modulja egyes részeit a következő mezők bármelyikébe elhelyezheti megfelelő direktivák segitségével:
  - program mező
  - adatmező
  - common mezők
  - externál mezők (másik szegmens program mezeje)
- Biztositja az egyes szegmensek egymásra hivatkozási lehetőségét, segiti a szegmensek közötti információátadást.

3.2. Az extrakód szint

Kényelmessé teszi az értelmező-szinten való programozást. A programozó direktiva segitségével hozzákapcsolhat programjához könyvtári extrakód szekciókat, melyek pl. I/O tevékenységeket látnak el, vagy különböző szintü utasitásrendszereket valósitanak meg.

Másrészt a programozó maga is definiálhat értelmező szintü utasitásokat.

3.3. A makró szint

A programozó hivhat könyvtári makrókat, valamint maga is definiálhat makrókat.

Ehhez a szinthez tartoznak a feltételes utasitások is.

4.Az assembler készitésében alkalmazott módszer.

4.1.Az UTRA értelmezési mód.

Nincs módunk az UTRA ismertetésére /1/. Megjegyezzük, hogy az assembler elkészitéséhez megvalósitottunk egy speciális UTRA értelmezési módot, mely a VIDOS szubrutinkezelő rendszerére /SKR/ épül. Az /1/-ben ismertetett UTRA két lényeges ponton tér el.

- Lehetőség van jelsorozatokban cimparaméterek irására, melyek átvételéről az SKR gondoskodik
- Minden jelsorozatnak szubrutinfejjel kell kezdődnie.

4.2. Az assemblert két lépcsőben készitettük el. Az első lépcső volt az assembler egy egyszerüsitett változatának (SAS) elkészitése CDC-3300 COMPASS-ban. Ezután az assemblert felirhattuk saját magában.

Ennek a megoldásnak a választását többek között azzal indokolhatjuk, hogy az Astrol nem alkalmas UTRA értelmezési módu programok irására. A fent emlitett módszerbeli alkalmazásában kedvező tapasztalatokra tettünk szert az EMG 830 SIMPLE assemblerének elkészitésekor. Ekkor a MINSK MISI- fordító programja volt a munka kiinduló pontja.

5. Az assembler belső szerkezetének és müködésének vázlatos ismertetése.

5.1. Az assembler forditóprogram leglényegesebb vonása az, hogy két menetben dolgozza fel a forrásmodult, és késziti el az output modult.

Az első menet során a forrás modul soronként beolvasásra kerül a kijelölt inputról. A forrás-sor szintaktikus elemzése után elkészül egy közbenső formátum, mely táblázatokra hivatkozásokat tartalmaz. A második menet ezt a közbenső formát dolgozza fel és késziti el a tárgylistát és a bináris tárgymodult. E forditási technika előnye egyrészt, hogy a tárgylista a tárgymodul hü képét adja meg (ellentétben az Astrollal), másrészt a közbenső formátum pontos specifikálásával a

a fordítóprogram természetes szegmentálását értük el.

5.2. Az assembler minden számára ismert és ismeretlen nevet különböző szótárakban tart nyilván. A szótárazási technika szekvenciális. A szótárak diszken helyezkednek el. Egy szótáron belül a rekordhossz rögzitett.

5.3. Az első menet során a programozó által definiált minden névhez sorszám rendelődik, és egy táblázatban gyülik össze a nevekhez tartozó információ. A második menet a sorszám alapján találja meg a táblában a névhez tartozó információt.

5.4. Az assembler egy rezidens részből és három overlay szegmensből áll. A három overlay szegmens az assembler vezérlő programjának, az első és a második menetnek felel meg.

6. Tapasztalatok

6.1. Először az assembler CDC-3300-ra irt változata készült el. Ez a változat egy ASTROL- uj assembler konvertert is tartalmaz.
A VT 1010/B gépre készült változat programterve és a program elkészitése a program belövése során kiderült, hogy egyes rutinokat (pl. szótárazó) át kell irni értelmező szintről gépi szintre, mivel eredeti formájukban a fordítást különösen hosszabb programok esetén igen lelassitják.
Ebben a fázisban terveztünk kettős pufferezési

technikát, szintén a gyorsitás érdekében.

Jelentős többletmunkát okozott, hogy a rendszer logikai input és file-kezelést szervező rutinjai elkészülte előtt kellett az assemblernek elkészülnie, ezért a rendszer ezen szolgáltatásait nem használhatta.

EGY MINSZK OPERÁCIÓS RENDSZERBE TARTOZÓ MACRO ASSEMBLER KIDOLGOZÁSÁNAK PROBLÉMÁI

Buzder Lantos József
INFELOR

Bevezetés

Még egyetemi hallgató koromban részt vettem az INFELOR -ban felállitott speciális kiépitésü MINSZK-22 -re kisérletképpen tervezett operációs rendszer munkálataiban.
Az egyik problémakör a forditóprogramok kidolgozása különböző szinten, s olyan modulstrukturában, hogy egy program tetszés szerinti nyelven irt modulokból felépithető legyen.
Mivel a gépen akkor csak egy egyszerü assembler müködött, azt a feladatot kaptam, hogy diplomamunkaként készitsek el egy teljes assembler -t, amely modulstrukturában fordit és makro preprocessorral is rendelkezik.

Az assembler tervezete

Elsősorban megterveztük a programirás formátumait a gép két-cimü indexelhető utasitás-tipusainak megfelelően.
Továbbá kidolgoztunk egy mnemonik rendszert az utasitáskódokra. A modul szerkezetére vonatkozóan definiáltunk belső és külső neveket, az utóbbiakat a modulok egymásra hivatkozásának elősegitésére.

A modulok közti információcsere céljából definiáltuk a kommonváltozókat.

A programozó deklarálhat indexeket, lokális- és kommonváltozókat, és egyes változókat ujra elnevezhet.
Megengedett továbbá a makro-definiálás is.

A fordító outputját ugy terveztük meg, hogy az a gépen egzisztáló egyszerü assembly nyelvre fordit. A tárgymodul fej és törzs részből áll. A fejrész a tárfelosztáshoz szükséges információkat tartalmazza.
A szerkesztő-betöltő program ezt az egyszerü assemblert használja fel olymódon, hogy a szerkesztőnek szóló jeleket felismeri és átadja az assemblernek megfelelő jelek formájában, a tárgyszövegben lévő egyéb jeleket pedig egyből átadja az assemblernek.

A fordítóprogram tervezése

A vállalatnál korábban kidolgoztak egy általános fordító rendszert /UTRA/, amelynek módszerét kiindulásnak tekintettük az assembler megvalósitásában /és a szerkesztő-betöltő programban is/. Ezen fordítórendszer elsősorban a jelek kezelésével foglalkozik, és ha arra szükség van, egy speciális szürőn ereszti át őket szintaktikus egységek felismerésére. A /Dömölki Bálint által kidolgozott/ szintaktikus elemző tevékenység azonban jobb hatásfoku, magasabb szintü nyelveknél, ezért itt kombinálva használtuk a MINSZK -re kidolgozott operációs rendszer egy információgyüjtő programjával.

Ez az információgyüjtő az egyes szintaktikus elemek felismerését végezte, és a Dömölki Bálint-féle szürőre csak a sorokban szereplő szintaktikus egységek helyes összeállitásának ellenőrzését biztuk.

A forditóprogram egy menetben dolgozik. Elsősorban a cimkéket, változó-neveket ismeri fel, amelyeket azonban leképez a tárgy assembly nyelv megfelelő elemeire. Ilymódon kikerüli a postdefinit hivatkozásokat, illetve rábizza az utána következő forditóra.

Ezek után a mnemonik feldolgozásban az egyes szótárakban megkeresi a kérdéses nevet, és leágazik a neki megfelelő tevékenységre. Utasitáskód esetén megjegyzi az utasitás lehetséges formáját, és a további rész feldolgozását ennek megfelelően végzi. Direktiválnál ráállit az általuk definiált tevékenységre. /pl. deklaráció felismerése és végrehajtása stb./

Amennyiben makro-deklarációt talál, ugy a formális paramétereket külön szótárba helyezi, és lerakja a törzsben lévő jeleket, amely jelsorozatra pointert állit, s ezt tekinti a makronév válaszának. A makrohivásnál az UTRA jelgenerátor része ezt a pointert kapja meg, és kezdi generálni a definiciókban megadott jelsorozatot.
A megoldás annyiban egyszerüsödött, hogy a makro válaszában lévő jelsorozat csak az egyszeri assembly nyelv számára érthető jelekből állhat, és a formális

paramétereket specifikáló jelből. Ilymódon egy makróban ujabb makró hivás nem lehetséges.

A tárgymodul elkészitése viszonylag egyszerüen történik, hiszen az utasitás válaszának megfelelő pointer egy karakterosorozatra mutat, amelynek outputra adása közben olyan pointer tipusu jeleket találunk, amelyek az operandusoknak megfelelő jelsorozatokra mutatnak. Ebben a tevékenységben is igen jól tudjuk használni az UTRA eszközeit.

Befejezés

Mivel a fenti munkát kisérletnek szántuk, semmiképpen nem tekinthetjük lezártnak, a későbbiekben nem foglalkoztunk a teljessétételével, de tapasztalatainkat a folyó munkáinkban felhasználtuk /lásd. FACOM-R, assembler).

A STAGE2 MAKROPROCESSOR IMPLEMENTÁLÁSA ÉS ALKALMAZÁSAI

Krammer Gergely

MTA Automatizálási Kutató Intézet

Zimányi Józsefné

MTA Központi Fizikai Kutató Intézet

1. Bevezetés

A makroprocesszorok első alkalmazása az assembly nyelven való programozás megkönnyitése volt. McIlroy (1960) foglalta össze először az általánosabb alkalmazási lehetőségeket. Az ő gondolatait továbbfejlesztve az utóbbi években számos szerző dolgozott ki programozási nyelvhez nem kötött makroprocesszorokat (Strachey /1965/, Waite /1967/, Brown /1967/). Ezeknek tipikus alkalmazási területei: szöveggenerálás, szövegszerkesztés, programozási nyelvek kiterjesztése, programozási nyelvek fordítása.

Az egyik legérdekesebb kezdeményezés a W.M.Waite és munkatársai által kidolgozott un. Hordozható Programozási Rendszer (Mobile Programming System), melynek magva a STAGE2 makroprocesszor ([1], [2] és [3]). A STAGE2 rugalmassága és hatékonysága miatt algebrai nyelvek fordítására is alkalmas, másrészt könnyen implementálható különböző számológépeken, igy a probléma-orientált nyelven irt software implementálásának hatékony eszköze.

2. A STAGE2 makroprocesszor

A legegyszerübb makroprocesszorokban a makro definició szerkezete

MACRO MNEV(P_1,P_2,...,P_n)
makro törzs
MEND

A "MACRO" ill. "MEND" stringek jelölik a definició kezdetét és végét, "MNEV" a makro neve, $P_1,\ldots,P_n$ a formális paraméterek. A makro törzs határozza meg a makro helyére behelyettesitendő szöveget, melyben hivatkozások vannak a formális paraméterekre. A feldolgozandó szövegben a makro hivása MNEV($A_1,\ldots,A_n$) alaku, ahol $A_1,\ldots,A_n$ az aktuális paraméterek. A processzor a név alapján ismeri fel a hivást, és elvégzi a makro kifejtését, azaz a hivás helyére bemásolja a makro törzset, amelybe az előirt módon behelyettesitette az aktuális paramétereket.

A STAGE2 makro definicióiban a makronév szerepét a leforditandó nyelv sablonjai játsszák. A sablon egy string, melyben a paraméterek helyén paraméter-lyukak vannak. (Ezeket a definicióban egy speciális karakter, a paraméter-flag jelöli.) Példa sablonra:

' = ' * '

ahol a ' karakter a paraméter-flag.

A STAGE2 a feldolgozandó szöveg sorait rendre összehasonlitja a sablonokkal. Ha egy sablon rögzitett részei pontosan megfeleltethetők a sor karaktereinek, azt mondjuk, hogy a sablon illeszkedik a sorra. A sornak azok a rész-stringjei lesznek az aktuális paraméterek, melyek a sablonban lévő paraméter-lyukaknak felelnek meg. Pl.:

P = Q * (R + S)

megfelel a fenti sablonnak, P, Q és (R+S) a három paraméter. A makro felismerés egyértelmüségét egyszerü algoritmus biztositja. A makro kifejtés során a paraméterek egyszerü behelyettesitésén kivül számos további lehetőség áll rendelkezésre: feltételes elágaztatások, paraméterek ciklusban való feldolgozása stb.

A különböző makrok egymást, vagy rekurziv módon önmagukat is hivhatják. Egymásnak információt a paramétereken vagy globális string-változókon keresztül adhatnak át.

3. A STAGE2 processzor implementálása

3.1 A FLUB nyelv

A STAGE2 megirásánál lényeges szempont volt, hogy könnyen implementálható legyen, ezért Waite és munkatársai az általuk tervezett FLUB (First Language Under Bootstrap) nyelven irták meg. Ez egy "absztrakt számológép" assembler nyelve, amely

- olyan egyszerü, hogy egy, a STAGE2-nél jóval primitivebb bootstrap-processzorral lefordítható egy konkrét számológép assembler nyelvére,
- alkalmas a STAGE2-ben használt adat-tipusok (egész számok, stringek, bináris fa-strukturák) kezelésére.

A "FLUB gép" szerkezete:

- 36 regiszter,
- I/O buffer,
- memória
- legfeljebb 9 I/O csatorna.

Utasitásrendszere 28 utasitásból áll.

3.2 Bootstrapping. A SIMCMP processzor.

A FLUB nyelv egy, a STAGE2-nél primitivebb makro-processzorral lefordítható egy konkrét számológép assembler nyelvére. Ilyen egyszerü processzor a SIMCMP, a Mobile Programming System alapja. A SIMCMP-t FORTRAN nyelven irták meg, igy FORTRAN forditóprogrammal rendelkező gépre azonnal alkalmazható, de egy assembler nyelvü változat is gyorsan beprogramozható. A SIMCMP-vel lefordithatjuk a STAGE2-t az adott gép nyelvére. Az igy kapott (helyfoglalás és futási idő szempontjából nem jó hatásfoku) STAGE2 felhasználható önmaga optimalizált változatának előállitására. A "bootstrapping" lépései tehát:

a) SIMCMP implementálása (FORTRAN vagy assembly nyelv)

b) makrodefiniciók SIMCMP számára (FLUB→assembly nyelv)
c) STAGE2 első változat (fordítás SIMCMP-vel)
d) optimalizált makrodefiniciók STAGE2 számára (FLUB → assembly nyelv)
e) STAGE2 optimalizált változat (fordítás STAGE2 első változatával)

4. Alkalmazási lehetőségek. Az absztrakt számológép.

A STAGE2 igen rugalmas és hatékony általános célu makroprocesszor. Tipikus alkalmazási területe a probléma-orientált nyelven irt software implementálása. A probléma-orientált nyelvet ugy foghatjuk fel, mint egy absztrakt számológépet, amelynek utasitásrendszere és adatstrukturája az adott feladat természetéhez igazodik. A makroprocesszor az az eszköz, amely az absztrakt számológépnek egy konkrét gépre való leképezését megvalósitja. Ilyen absztrakt számológép a FLUB is.

5. Implementáció, eddigi alkalmazások

A STAGE2 implementáció a 3. pontban leirt "bootstrapping" módszerrel az MTA CDC 3300 és a KFKI ICL 1905 számológépére készült el. Eddig a következő feladatok megoldására használtuk fel:

- ICL 1900 FORTRAN gépi reprezentációban irott programok forditása ANSI FORTRAN-ra,
- a FORTRAN nyelv kiterjesztése polinomok szimbolikus kezelésére szolgáló utasitásokkal (a SYMBOLANG rendszer szubrutinjaira alapozva) [4].

Irodalom

[1] R.J.Orgass - W.M.Waite: A Base for a Mobile Programming System. Communications of ACM 12. (1969) 507-510.
[2] W.M.Waite: The Mobile Programming System: STAGE2 Communications of ACM 13. (1970) 415-421
[3] W.M.Waite: Building a mobile programming system The Computer Journal 13. (1970) 28-31
[4] Lovas Istvánné: Egy szimmetrikus listakezelő rendszeren alapuló algebrai manipulátor (Szegedi előadás, 1972.)

EGY PROCESSZOR MAKRO KITERJESZTÉSRE, ÉRTELMEZÉSRE ÉS FORDITÁSRA

Farkas Ernő
MTA Automatizálási Kutató Intézet

Mivel az itt leirt makroprocesszor igen általános célu, igy először müködését vázoljuk és csak a müködés ismeretében világitjuk meg néhány felhasználási lehetőségét.

1. A processzor müködése

A processzor inputjáról rekordokat, esetünkben sorokat kap. A sorokat két csoportba osztjuk, direktivákra és szöveg sorokra. A direktivák speciális jelző karakterrel kezdődnek.

- A direktivák hatására a processzor müködési módja megváltozik.
- A szövegsorokat a processzor feldolgozza és a feldolgozás eredményeként az outputján egy uj szöveg kerül kiadásra.

2. Néhány alapvető direktiva

2.1. Makro definició:

& MACRO NO n NEXT k

hatására a makró könyvtárba n sorszámu makróként kerülnek be a soronkövetkező sorok mindaddig, amig az

& END OF MACRO

sort el nem érjük.

2.2. Müveletek a processzor belső változóival:

Az abc minden betüjéhez a processzorban egy-egy munkarekesz tartozik, ezek segitségével egész aritmetikai müveleteket végezhetünk. Pl.

$$\&A = \&B \left\{\begin{matrix}+\\-\\*\\/\\:\end{matrix}\right\} \&C \quad \text{vagy} \quad \&E = 12$$

ahol / az osztás hányadosát, : a maradékát jelenti.

2.3. A soronkövetkező sorok feltételes, vagy feltétlen kihagyása:

& SKIP n

& SKIP n IF & A $\left\{\begin{matrix}\text{POSITIV}\\\text{NEGATIV}\\\text{ZERO}\end{matrix}\right.$

2.4. Input, vagy output berendezés kijelölése:

& INPUT : n

& OUTPUT: n

n = 1 konzolirógép
2 szalagolvasó, ill.lyukasztó
3/4 mágnesszalagok.

3. A szövegsorok feldolgozása

A processzor az inputról érkező sort megvizsgálja, hogy azonos-e egy kijelölt makró hivásával. Ennek a makrónak a kijelölése a & BEGIN AT n direktivával történik.

Ez alól csak azok a sorok kivételek, amelyek előtt a

& MATCH WITH k

direktiva áll, ezeket a k számu makróval veti egybe. Ha ugy találja, hogy a sor nem a megfelelő makró hivása, akkor az egybevetést a NEXT-tel megjelölt sorszámunál próbálja meg. És ezt folytatja mindaddig, amig 0, vagy 1-es sorszámra hivatkozunk. Ha 0-ra hivatkozunk, hibajelzést ad, ha 1-esre, a sort változatlanul kiadja az outputra.

4. A makróhivások felismerése

A definiált makró két részből áll, az első sora egy minta a hivás formájára. A következő sorok az u.n. kódtörzs, amellyel a hivást helyettesitjük.
A minta három különböző elemből állhat:

- kulcsszavakból /a sor elejét és végét jelző jelek mindig kulcsszónak számitanak/

- kötött paraméterekből /amelyeket !-kel jelölünk/.

Ezek mindig egy kulcsszó előtt, vagy után állnak és annyi karaktert jelölnek ahány ! áll ott és a hozzájuk tartozó string (és) jelet nem tartalmazhat.

- szabad paraméterekből, amelyek a kulcsszavak, illetve a hozzájuk tartozó kötött paraméterek között állnak /jelük ?/ és egy tetszőleges hosszu, de helyesen zárójelezett karaktersorozatot jelölnek.
Pl. egy FORTRAN IF utasitás mintája:

!!!!! IF (?) ?,?,?

5. A makró kiterjesztés

Ha a sort sikerült egy makrómintához illeszteni, azaz a kulcsszavakat megtalálni benne és a kulcsszavak közti jelsorozatot megfelelő módon felosztani szabad és kötött paraméterekre, akkor a sort a megfelelő kódtörzzsel helyettesitjük. A kódtörzsben hivatkozások lehetnek az egyes paraméterekre, illetve azoknak valamilyen konvertált formájára.

Miután a kódtörzsből és a paraméterekből megalkottuk azt a szöveget, amellyel a sort helyettesiteni kell, a szöveget ujból átvizsgáljuk. Ha a következő sor nem direktiva, akkor outputra kerül, ha direktiva, végrehajtjuk.

6. A processzor által nyujtott lehetőségek

A közönséges makrómanipulációkra már a fenti lehetőségek is elegendők, de arra a célra, hogy egyszerübb nyelvek forditását, interpretálását megkönnyitsük még a következő lehetőségeket akarjuk bevezetni.

- Makrókészlet nevek, azonositók, számok felismerésére.
- Szótározási lehetőség: bármely karaktersorozathoz 4 byte-ot rendelhetünk, amelyben valós, vagy egész számot tárolhatunk és ezzel müveleteket végezhetünk.
- Gépikód generátor: képes arra, hogy beadott számokból megfelelő bináris szalagot készitsen.

7. A processzor felhasználási lehetőségei

A processzor a 6. pontban felsoroltak nélkül is használható a következő célokra:

- Nyelvek kibővitése makrókkal.
- Szövegszerkesztési feladatok /forrásnyelvü programok javitása, paraméterezése/.

A teljes processzor ezen kivül használható:

- Egyszerübb nyelvek forditására.
- Bizonyos nyelvek interpretálására.
- Posztprocesszornak.

8. Technikai adatok

A processzor a CII loolo gépre készül, disket és 16K memóriát használ, mágnesszalagot opcionálisan.

Elkészülésének várható ideje; alapváltozat 1972 októbere, teljesváltozat 1972 decembere.

EGY SZIMMETRIKUS LISTAKEZELŐ RENDSZEREN ALAPULÓ ALGEBRAI MANIPULÁTOR

Lovas Istvánné

MTA Központi Fizikai Kutató Intézet

1. Bevezetés

A számológép nem numerikus alkalmazásában jelentős szerepet játszott a listakezelő nyelvek kialakulása. A mesterséges intelligencia, az információ tárolás és visszanyerés, az algebrai manipuláció és számos hasonló terület fejlődésével az általános numerikus programnyelvek nem bizonyultak megfelelőnek a felmerülő problémák megoldásához.

Az elsősorban numerikus alkalmazások céljára készült programnyelvekben (ALGOL, FORTRAN) az összetartozó adatok a memóriában folytonosan helyezkednek el, és a feldolgozás általában a tárolás sorrendjében történik.

A nem-numerikus alkalmazások, igy az algebrai kifejezések szimbolikus kezelése is általában olyan tárolási módot igényelnek, amelyben az adatok elérése a tárolás fizikai sorrendjétől függetlenül, belső összefüggésük alapján történik. Szükséges továbbá, hogy az adatstruktúrák dinamikusan változtathatók legyenek a feldolgozás folyamán. Ilyen dinamikus tárológazdálkodást biztositanak a különböző listakezelő rendszerek (LISP, SLIP, IPL-V). Ezért az algebrai kifejezések szimbolikus kezelésére szolgáló rendszerek legtöbbje valamilyen listakezelő rendszerre épül.

A SYMBOLANG szubrutinrendszer polinomokkal való szimbolikus müveletek elvégzésre készült. A SYMBOLANG a SLIP listakezelő rendszerre épül, amely a FORTRAN nyelvbe van beágyazva. Igy a SYMBOLANG felhasználójának a FORTRAN nyelv numerikus lehetőségei is rendelkezésére állnak.

2. SLIP (Symmetric List Processor)

A SLIP szimmetrikus listák kezelésére szolgáló FORTRAN szubrutin rendszer, Weizenbaum [1] dolgozta ki 1963-ban. Adatstrukturája szimmetrikus listastruktura, egy un. "listafej" elemmel.

A listák elemekből állnak, a listák elemei vagy tovább nem bontható adatok: atomok vagy listák, a főlista allistái. Egy atom egy valós vagy egész szám, vagy gépi reprezentációtól függő számu karakter.

A listaelemek tárolására a számológép memóriájában cellákat használunk. Minden cella tartalmaz információt a rendszer számára; a lista következő, illetve megelőző elemének gépi cimét. Ezeket a mutatókat (pointereket) követve a lista egymás utáni elemei könnyen kezelhetők, a lista bővithető, lebontható stb.

Mivel a lista-elemek mutatókkal vannak egymáshoz kapcsolva, nem kell, hogy a lista cellái a memóriában egymás után következzenek. A legtöbb listakezelő rendszerhez hasonlóan a SLIP a memóriát dinamikusan osztja ki, a rendelkezésre álló memóriaterületből elkészíti a szabad cellák listáját, amelyről minden lista igényének megfelelően kaphat uj cellákat. Ha valamely cellára egy listának a továbbiakban nincs szüksége, a rendszer a felszabadult cellát visszahelyezheti a szabad cellák listájára. Azt, hogy egy listára szükség van-e vagy sem, az un. "referencia count" határozza meg. A referencia count egy állandó, folytonos, majdnem automatikus "garbage collection"-t biztosit.

A SLIP nem önálló nyelv, hanem a FORTRAN nyelvbe beágyazott szubrutinrendszer, igy a FORTRAN nyelv minden lehetősége a felhasználó rendelkezésére áll. A különböző listakezelő funkciókat FORTRAN szubrutinok végzik el. Igy a rendszer szubrutinjaival listastrukturákat épithetünk fel, adatokat helyezhetünk el a listán, a listán tárolt adatokhoz hozzáférhetünk. Egy adatot megkereshetünk egy listán, ki-

cserélhetünk vagy kitörölhetünk egy listáról, listákkal vagy liststrukturákkal különböző müveleteket végezhetünk.

3. SYMBOLANG - egy egyszerü algebrai manipulátor

A SYMBOLANG egyszerü, de munkaigényes és sok hibaforrást magába rejtő, az algebra és analizis tárgykörébe eső feladatok megoldására alkalmas olyan esetben, amikor a megoldás lépései előre megadhatók.

Ilyen feladatok például polinomok szorzása és összevonása, hatványsor részletösszegének behelyettesitése, polinomok differenciálása és integrálása stb. A rendszert Lapidus és Goldstein [2] dolgozta ki 1965-ben IBM 7094-es számológépre.

A SYMBOLANG a SLIP rendszeren alapuló FORTRAN nyelven irt szubrutin gyüjtemény. A rendszer bővitése, esetleges ujabb operációk bevezetése a szubrutincsomag kibővitését jelenti.

3.1 Az algebrai kifejezések jelölése

A rendszer tervezésénél mind a külső, mind a belső ábrázolásmódot figyelembe kell venni. A SYMBOLANG olyan jelölésmódot használ, amelynél a külső megjelenési forma és a belső ábrázolás megegyezik.

Egy algebrai kifejezés listastrukturaként van tárolva, melyen a kifejezés minden tagja allistaként helyezkedik el.

Az allista a következő sorrendben tartalmazza az adatokat:

- a tag együtthatója (lebegőpontos szám)
- első változó neve (legfeljebb 8 karakter)
- első változó exponense (lebegőpontos szám)
- második változó neve
- második változó exponense

•
•
•

- n-dik változó neve
- n-dik változó exponense.

Ez a jelölésmód zárójelet nem enged meg.

3.2 Manipulációs rutinok

A SYMBOLANG algebrai manipulátor rutinjai két nagy csoportra oszthatók:

1. Algebrai müveleteket végző rutinok:

- két tag szorzása
- két kifejezés szorzása
- egy tag osztása konstanssal vagy változóval
- egy kifejezés osztása konstanssal vagy változóval
- behelyettesités
- differenciálás
- integrálás
- egyszerüsités
- csonkitás (magasabbfoku tagok elhagyása)
- lineáris egyismeretlenes egyenlet megoldása.

2. Segédrutinok egyszerübb feladatok elvégzésére: pl. egy tag lemásolása, egy változó exponensének megkeresése stb. Ide tartoznak a következő rutinok:

- egy kifejezés szétválasztása adott változót tartalmazó, és nem tartalmazó részre
- egy adott exponensü változó együtthatójának megkeresése
- egy tagban egy adott változó exponensének meghatározása
- egy kifejezés vagy egy tag lemásolása
- egy kifejezés beolvasása
- egy kifejezés kinyomtatása.

4. A SYMBOLANG implementálása, bővithetőség

A SYMBOLANG algebrai manipulátor implementálása az irodalom [2], [3] alapján történt. Mivel a SLIP listakezelő rendszer az MTA két számológépén rendelkezésre állott [4] a SYMBOLANG implementációja nem okozott nehézséget. A rendszer FORTRAN nyelven irt szubrutinokból áll, amelyek más tipusu gépen is felhasználhatók.

A munka célja egy müszaki és tudományos feladatok megoldására jól használható rendszer megvalósitása volt.

A SYMBOLANG rendszer szerkezeti felépitése igen nagy flexibilitást biztosit. Az implementáció során a rendszerhez hozzá illesztettünk néhány ujabb rutint is (integráló, beolvasó rutin stb) és tervbe vettük a rendszer kiterjesztését racionális törtfüggvények kezelésére.

Irodalom

[1] Weizenbaum J: Symmetric list processor.
Communication of ACM 6. (1963) 524

[2] Lapidus A., Goldstein M.: Some experiments in algebraic manipulation by computer.
Communication of ACM 8. (1965) 501

[3] CLAM; A compatible list processor and algebraic manipulator Kézikönyv (Courant Institute of Mathematical Sciences)

[4] Krammer G., Lovas Istvánné: SLIP - szimmetrikus listakezelő rendszer az MTA CDC 3300-as és az ICT 1905-ös számológépén. KFKI, 1972.

HÁZILAG KÉSZITETT, MÁGNESLEMEZEN TÁROLT MÜKÖDTETŐ RENDSZER

Kostyán Ákos

KGM ISZSZI

Az elektronikus adatfeldolgozó rendszer, amely mellett dolgozom, 1966. október 1o. óta müködő, második generációs, közepes teljesitményü gép. A külső egységek a központi egységgel közvetlen kapcsolatuak, be- és kivitel csak a központi tárolón keresztül lehetséges. A központi tároló ferritmagos, 12 ezer cimü jeltároló, 1 jelben az ábrázolás 6 információ-, 1 szójel- és 1 ellenőrző-biten, BCD kódban történik.

A gép az utasitásokat időben egymás után hajtja végre; a központi egység mindig várakozik, ha a külső egységek végeznek müveletet. Ezért egyidejüleg csak egyetlen program futtatható.

A produktiv programjainkat kizárólag Autocoder nyelven irjuk, minden programba makró-könyvtárból épül be az összes szükséges adattár-kezelő eljárás. A végrehajtható programok mintegy 1oo--2oo kártyás csomagokban nyerhetők a fordítóprogramtól.

Hamar kiderült, hogy a programok 3 mágnes-lemez hajtó egységnél és 8-1o ezer központi tároló cimnél általában nem igényelnek többet, miközben a feladatok végrehajtása egyre több kezelési nehézséggel járt. A kártyacsomagok kezelése ugyanis türhető addig, amig egy-egy programra mondjuk havonta egyszer kerül sor, de tarthatatlan, ha a feladat naponta ismétlődik.

Két évi tapasztalatgyüjtés után, 1968-ban elhatároztam, hogy a nagyobb hatékonyság érdekében müködtető rendszert készitek, amely ugyan a korszerüeknél sokkal kisebb képességü, de a mi céljainknak jól megfelel. A rendszer 1969 elején került bevezetésre, majd 197o-ben nyerte el végleges formáját.

HÁZILAG KÉSZITETT, MÁGNESLEMEZEN TÁROLT MÜKÖDTETŐ RENDSZER

Kostyán Ákos

KGM ISZSZI

Az elektronikus adatfeldolgozó rendszer, amely mellett dolgozom, 1966. október 1o. óta működő, második generációs, közepes teljesitményü gép. A külső egységek a központi egységgel közvetlen kapcsolatuak, be- és kivitel csak a központi tárolón keresztül lehetséges. A központi tároló ferritmagos, 12 ezer cimü jeltároló, 1 jelben az ábrázolás 6 információ-, 1 szójelés 1 ellenőrző-biten, BCD kódban történik.

A gép az utasitásokat időben egymás után hajtja végre; a központi egység mindig várakozik, ha a külső egységek végeznek müveletet. Ezért egyidejüleg csak egyetlen program futtatható.

A produktiv programjainkat kizárólag Autocoder nyelven irjuk, minden programba makró-könyvtárból épül be az összes szükséges adattár-kezelő eljárás. A végrehajtható programok mintegy 1oo--2oo kártyás csomagokban nyerhetők a forditóprogramtól.

Hamar kiderült, hogy a programok 3 mágnes-lemez hajtó egységnél és 8-1o ezer központi tároló cimnél általában nem igényelnek többet, miközben a feladatok végrehajtása egyre több kezelési nehézséggel járt. A kártycsomagok kezelése ugyanis türhető addig, amig egy-egy programra mondjuk havonta egyszer kerül sor, de tarthatatlan, ha a feladat naponta ismétlődik.

Két évi tapasztalatgyüjtés után, 1968-ban elhatároztam, hogy a nagyobb hatékonyság érdekében működtető rendszert készitek, amely ugyan a korszerüeknél sokkal kisebb képességü, de a mi céljainknak jól megfelel. A rendszer 1969 elején került bevezetésre, majd 197o-ben nyerte el végleges formáját.

A működtető rendszer kialakításánál azt tartottam fontosnak, hogy a kezelendő programok számára minél nagyobb központi tároló területet adhassak, a programok kezelése minél gyorsabb legyen, és a programozóknak az addig megszokott módszereiket ne kelljen megváltoztatniuk. Igy elértem, hogy bármilyen korábbi programot is kezelni tud a rendszer, bár a mágnes-lemez kihasználás ilyen esetben kevésbé gazdaságos.

Mágnes-lemezeink 1oo cilindert tartalmaznak, minden cilinderben 1o sáv és minden sávban 2o szektor van. Minden szektornak 6 jelü cime van, amely őt megelőzi. A cimben is és a szektorban is BCD jelábrázolás van, de a szektorban kétféle módon: szójelekkel 9o, ezek nélkül 1oo jel fér 1 szektorba. Egy lemezen belül a cimek oXoooo és oY9999 között folytonosan növekedve helyezkednek el; X értéke o,2,4,6 vagy 8 lehet, Y=X+1.

Rendszertechnikai célokra 1o2o központi tároló cimet és 8ooo szektort terveztem, igy a 4 lemez hajtó egységből 3 mindig a felhasználóé, sőt még a rendszer-lemez első 12 ezer szektorát is használhatja. Tárvédelmi okból a lemez elején oooooo-tól o11999-ig, majd o92ooo-től o99979-ig vannak a szektorcimek. A legutolsó sávot nem használom. A program-könyvtár számára tehát 4o cilinder van kijelölve; ezekben 38 felhasználói program tárolható, mert 2 rendszer-programot is velük együtt kezelek. A kezelés alapelve az 1 program -- 1 tároló-kép -- 1 cilinder elv, igy igen sok feladattól mentesülhetett a rendszer.

A felhasználók számára szükséges központi tároló terület a 87. cimtől kezdődik, és a 1o976. cimig terjedhet. Az e területen lévő 1o89o jel 121 szektorban fér el, tehát minden program - cilinderen marad egy 79 szektoros "lyuk" /711o jel/ is. Ezekből az első 5-öt és az utolsót megköveteli a rendszer, a többi rendelkezésre áll fejlesztési célokra. Ha a felhasználó az esetleg 2-3 ezer jelnyi programjaira sajnál teljes területeket lekötni, saját programozással több programot összefoghat.

A működtető rendszer jelenleg 9 programból áll, ezekből 7 az említett "lyukak"-ban tartózkodik. A központi tárolóban minden feladathoz szükséges a CTL nevü vezérlő-program jelenléte, amely az utolsó program-cilinder fenntartott részéből tölthető be a hozzá tartozó, gépi nyelven tervezett program-szektor beolvasásával és végrehajtásával. Erre való a DKB nevü kiegészitő program, amelynek feladata, hogy beolvassa és végrehajtassa egy lemez valamelyik szektorának utasitásokból álló tartalmát. Gépi nyelven terveztem, a magtároló első 148 cimét igényli és 1 lyukkártyában fér el. A 49. oszlop tartalma határozza meg a lemez hajtó egységet.

A CTL feladata kapcsolat teremtése a felhasználó, a gépkezelő és a könyvtár, a könyvtár és a gép, valamint a rendszer programjai között. Betöltése után un. dátum-kártyát olvas és konzol irógépen kiirja, majd vezérlő-kártyákat olvas a 11. oszlopig és az első 5 jelet kiirja. Ha a jelsorozat PAUSEii, megállitja a gépet és kijelzi ii értékét; ujrainditáskor a vezérlőkártya olvasásra tér vissza. Ide történik normális visszatérés minden rendszer- és felhasználói programból is. Ha a 3. jel /, a CTL az 5 jelet a programnév-katalógusban keresi, egyébként az 1-2. jelet előre meghatározott 2 jelhez hasonlitja, hogy a téves hivásokat megakadályozza. Ha a jelek egyenlők, az xyRSB nevü futás-számoló programot keresi. A katalógusban megtalált programot egyetlen utasitással olvassa be a 87-1o976. cimekre. Az ez alatti területet a rendszer előtti időből megszokott módon alakitja ki; itt lesz minden esetben a dátum is, mert lehet, hogy a programba épült IOCS számára szükséges. A CTL dátum-kártya helyett csak UPDAT kártyát fogad el, és hatására az UPD nevü módositó-programot hivja be.

Az UPD feladata a felhasználói programok gépi nyelvü változtatásainak átvezetése vagy törlésük a könyvtárból: PATCH, DELET és »END« kártyákkal kell irányitani. A CLOSE kártya végleges megállást okoz, a REMIT a CTL dátum-olvasáshoz térit vissza.

INSER kártya hatására az UPD a CTL helyére az LDR nevü elhelyező programot betölti és átadja neki a vezérlést; INSER-SORT "rövidre" állitja a CTL-t és bekéri tőle az SRC nevü, rendezés felvevő programot.

Az LDR a felhasználói programot kártyacsomagból a magtárolóba tölti; megkeresi benne az összes olyan jelet, amely a külső egységek irási/olvasási müveleteit zárja le, és ezeket megváltoztatja, cimüket pedig a program nevével és belépési pontjával együtt a katalógusba irja. A 87-1o976. cimek tartalmát egyetlen utasitással a soron következő program-cilinder alsó 121 szektorába beirja. A programok között a magtárolót törli, kivéve a rendszer-területet. Az utolsó kártya után módositja a katalógus-nyilvántartó szektort is/o99939/. A katalógus a o9994o-o99979 szektorokban van.

Az SRC 1 kártyát 1 szektorba ir be/o92121-o92198/. A kártyák a munkamenet azonositó számát, legfeljebb 1o rendezési mező relativ cimeit, a hosszusági- és blokkolási paramétereket tartalmazzák. Az utolsó kártya után a vezérlő-szektorokat nyilvántartó szektort/o92199/ is módositja. Ha a vezérlő-szektorok ujraszervezése szükséges, jelezni kell a konzolon. Az SRC a CTL dátum-olvasáshoz visszatérithet.

Az RSB egy kezdetben ooooo tartalmu változójának értékét 1-el megnöveli, majd visszairja a saját lemez-területére. Ezután sorszámként kiirja a konzolon, majd egy régi alprogramommal 7-szeresre nagyitva a sornyomtatón is /a nagyitás csillagokból alakitja ki a papiron az irásjeleket/. Erre azért van szükség, hogy a gépkezelők semmit se dobhassanak el. Az xyRSB a CTL-nek ugy adja vissza a vezérlést, mintha ő be sem jött volna.

Az SRT nevü program valamennyi rendező-menet közös főprogramja. Behivása xySRT-jjj- kártyával történik: a konzolon kiirja a -jjj- jeleket, ezután a vezérlő-szektorok között keresi a

jjj tartalmut. A megtalált szektor paramétereit feldolgozza és eltárolja. A kulcsokból és cimekből cimjelzőket képez, ezeket berendezi és a rendezett füzéreket a rendszer-lemez oolooo-es cimü szektorától felirja: igy mód van rá, hogy az első looo szektorban tartósan őrzött adattár lehessen. A program inputja csak lemez lehet; a kezelési eljárások a rendszer nélküli módtól csak a gépkezelői hibák kiiktatására létrehozott részekben különböznek. A program automatikus kapcsolatu az SR3 és SR4 programokkal. Az SR3 a rendező-menetek füzér-összeválogató és cimtábla képző fázisa; az SR4 a cimtábla szerint irja át rendezett outputtá az inputot.

Elhárithatatlan hiba esetén a gépkezelő a CTL 11666. cimére adhatja a vezérlést; ugyanez bármelyik felhasználói programból is lehetséges. Hatására a CTL azonnal betölti a rendszer-területre a DMP nevü tároló-nyomtató programot, ami által ő maga megsemmisül a magtárolóban. A DMP a O. és 112oo. cimek között nyomtatja ki a központi tároló tartalmát, loo jelenként, cimazonositó és szójel-bit jelző sorokkal együtt. A gépi nyelvünk ugyanis átalakitás nélkül, azonnal érthető: a tároló tartalmából rövid idő alatt minden hibára /főleg program-hibára!/következtetni tudunk. A nyomtatás befejeztével a program a CTL-t betöltő program-szektort olvassa be és átadja rá a vezérlést.

A központi tároló felosztása a rendszer programjai között állandó, vagyis minden program mindig ugyanarra a helyre kerül. A töltésre szolgáló gépi szerkezet használatakor az 1-8o. cimekre kerül 1 kártya tartalma; a 82-86. cimek a dátumot tartalmazzák, a 87-99. cimeken 3 index-regiszter van. A loo-232. cimek mindig kártyákból magtárolóba töltő programot tartalmaznak, amelyet sok program letöröl, és az igy keletkező területről nyomtatja a megszerkesztett sorokat. A lo977-11999. cimek között tartózkodik a CTL, 4 kijelölt bemenettel a programozás számára. A LDR programot az UPD a CTL helyére hivja be, és az 1-232. cimeket a fentiek szerint tölti meg tartalommal. A CTL

kijelölt bemeneteivel meg lehet kerülni a vezérlő-kártya olvasást, be lehet hivni a DMP programot, stb. Az UPD, SRC, RSB, SRT, SR3, SR4 mind a felhasználói területre kerül, csak külön böző hosszuságban; az SRT-t és az RSB-t még ugy is kezeli a rendszer, mint bármelyik felhasználói programot. Egyszóval a felosztás állandó a fázisokban /felvétel,módositás,futtatás/, csupán az egyes fázisok között különböző.

A müködtető rendszer számára szükséges lemezt egy kb. 7oo kártyás generáló program-sor és futtatási könyv segitségével néhány perc alatt lehet elkésziteni. A teljes időigényt az elhelyező /LDR/ program minden müködése is 1-2 perccel növeli meg.

Irodalom:

System Operation Reference Manual IBM 144o Data Processing System Form A24-3116, 1965. p.A1-E7,F16-33,H11,28-4o,I1.

Autocoder /on Disk/ Language Specifications IBM 14o1,144o, and 146o Form C24-3258, 1966. p.5-56.

Autocoder /on Disk/ Program Specifications and Operating Procedures IBM 14o1,144o, and 146o Form C24-3259, 1966. p.5-64.

Input/Output Control System Specifications IBM 144o Form C24-3o11, 1965. p.5-67.

Input/Output Control System Operating Procedures IBM 144o Form C24-3299, 1965. p.5-8.

Bengyel Gyula - Kostyán Ákos:
2o ooo óra belső müveleti idő az IBM 144o-es elektronikus adatfeldolgozó rendszeren
IBM Tájékoztató, 1971. május: 15-18, 31-35, 38.o. továbbá Gépgyártástechnológia, 1972. január: 37-38, 4o.o.

PROGRESO/1 PROGRAMOZÓNYELV SZÖVEGKEZELŐ ÉS DOKUMENTÁCIÓS FELADATOKRA

Münnich Antal

Infelor

A Progreso/1 fő jellemzői

A programozónyelvek száma már többezerre rúg. Indokolt tehát a kérdés: érdemes-e újat készíteni? A számítógép-felhasználás térképén azonban vannak olyan fehér foltok, amelyeknek a feltárása új programozónyelvet kíván.

Az elmúlt húsz év során a számítógépek ár/teljesitmény viszonya mintegy ezredére csökkent. Az új eljárások (mágnesbuborék-technika, lézeres holográfia stb.) alapján általában azt várják, hogy ez a meredek zuhanás a következő néhány évtizedben is folytatódik. Ha igy lesz, akkor egyre több munkahelyre kerül számítógéppel összekötött terminális. Mark Halpern, az IBM egyik kiváló rendszerprogramozója szerint akkor már "éppen úgy nem lehet hivatásos programozót adni mindenkinek, akinek számítógép kell, mint ahogy nem lehet soffőrt adni mindenkinek, akinek autó kell." A Progreso/1 nyelv első jellemzője tehát az, hogy nem hivatásos programozóknak, hanem a közvetlen felhasználóknak készült. Feltesszük, hogy ezek rendszerint interaktiv terminálison át folytatnak párbeszédet a számítógéppel.

A Progreso/1 második jellemzője, hogy nem numerikus, hanem szöveges feladatok megoldására való. A számítógépek árzuhanásával ezeknek legalább két nagy területe nyilik meg: az ügyvitel és a szakirodalmi dokumentáció.

Az ügyvitel numerikus feladatait ma is sikeresen oldják meg számítógépen; a szöveges feladatok közül azonban csak a triviálisakat, pl. tételek ábécé szerinti sorrendezését. Pedig a tisztviselő munkája abból áll, hogy bemenő szövegek függvényében kimenő szövegeket hoz létre. Ennek a munkának egyrésze előre megadható ügyrend szerint megy, másrésze azonban felelős emberi döntést kiván. Az első részt (legalábbis elvben) számítógépre lehet programozni; a második miatt interaktiv üzemmódban. Nyilvánvaló, hogy az ügyintézés gyorsabb és jobb lesz, ha jórészét gép végzi.

Ami a szakirodalmi dokumentációt illeti, ezt ma a szakemberek csak kevéssé használják. Amerikai adatok szerint a kutatásra forditott pénz és munkaidő kb. 60%-át már ismert és a szakirodalomban leirt tények, eljárások újrafelfedezésére forditják. (Ha ott 60%-át, akkor Magyarországon alkalmasint több, mint 90%-át!) Ennek egyik oka a nyelvi nehézség, a másik pedig az ismeretkeresés (ang.: information retrieval) mai módszereinek a nehézkessége. Az utóbbi nehézség azonban legyőzhető az adatbank-technika kihasználásával és továbbfejlesztésével. Nyilvánvaló, hogy az ismeretkeresés programozónyelvét célszerü összevonni a szövegfeldolgozásával.

Emlitettük, hogy a szöveges feladatok jórésze ma még csak elvben programozható. Ennek az az oka, hogy a tervszerütlenül kialakult nemzeti nyelvek gépi feldolgozásra kevéssé alkalmasak. Bar Hillel [1] szerint az angol nyelvü szövegek gépi elemzése elvileg sem lehetséges, hiszen még az ilyen egyszerü kifejezés sem egyértelmü, mint pl.: "slow electrons and neutrons". Nyelvileg ugyanis nem állapitható meg, hogy a "slow" (a.m. lassú) melléknév csak az "electrons" főnévnek, vagy az "electrons and neutrons" főnévi kifeje-

zésnek a jelzője; pedig a reaktorfizikus számára ez korántsem mindegy. Az angollal és más tervszerütlen nyelvekkel végzett munka sikertelenségével szembeállithatjuk az eszperantó elemzésével foglalkozó kutatók jó eredményeit. Botos (Magyarország), Cejtin, Kolker (Szovjetúnió), Maas (NSzK), Naemura (USA), Sellin (Dánia), Sjögren (Svédország), van Themoat (Hollandia) egybehangzóan arra az eredményre jutottak, hogy az eszperantót világos, logikus felépitése alkalmassá teszi gépi feldolgozásra. Ha ez a felismerés közkinccsé válik, akkor egyre több szakmai folyóirat fogja ezen a nyelven közölni cikkeinek eleinte csak tárgyszavait [2], később tartalmi kivonatát is, végül pedig teljes szövegét. Azok a folyóiratok ugyanis, amelyek ezt elmulasztják megtenni, hátrányba kerülnek, amikor a számitógépek árcsökkenésével a könyvtárak szerves részévé válik a szövegfeldolgozó és ismeretkereső berendezés.

Ezt a fejlődést anticipálva adódott a Progreso/1 harmadik jellemző tulajdonsága: az, hogy az eszperantónak részhalmaza. Tehát minden Progreso/1 mondat szintaktikusan helyes eszperantó mondat is. Ezzel két előnyt érünk el. Egyrészt azt, hogy a szövegfeldolgozással foglalkozó személynek nem két független nyelvet kell megtanulnia: az eszperantót, mint adatnyelvet, és valami egészen mást, mint programozónyelvet; igy tehát lényeges munkamegtakaritást érünk el. Másrészt igen érdekes, hogy a Progreso/1 nyelvü programot eszperantó szövegként értelmezve lehet majd elemezni és feldolgozni. Ez talán lehetővé teszi, hogy a hibakereső program a szintaktikus hibákon kivül a szemantikusakat, vagy ezek közül legalább néhányat (pl. végtelen ciklus) is megtaláljon. Érdekes lehetőségeket kinál a nyelv önfejlesztése terén is. A "Progreso" név egyrészt "Programada Esperanto" röviditése,

másrészt haladást jelent eszperantóul, célozva az önfejlesztés lehetőségére.

Ember--számitógép párbeszéd Progreso/1 nyelven

A programozó a terminálison leir egy mondatot. Ezt a számitógép értelmezi; ha szintaktikusan hibásnak találja, akkor ezt azonnal közli. Ha a mondat helyes, akkor megvizsgálja, hogy parancs-e. Ha az, akkor azonnal végrehajtja, ha más, akkor tárolja.

A parancsnak két fajtája van. Az alapparancs szintakszisát és szemantikáját maga a nyelv határozza meg, a hivóparancsét ellenben a programozó. A hivóparancs ugyanis a programozó által előzőleg deklarált eljárást hiv. Az eljárásdeklaráció háromféle mondatot tartalmazhat: közlést, kommentárt, és utasitást. Első mondata csak közlés lehet. Ez határozza meg a hivás szintakszisát. A kommentár, éppúgy, mint más programozónyelvekben, a programozó számára irt tájékoztatás. A gép tárolja és a program kiirásakor kiirja, egyébként azonban nem törődik vele. Az eljárásdeklaráció törzse utasitásokból áll. Az utasitás szemantikailag abban különbözik a parancstól, hogy nem beolvasása után azonnal hajtódik végre, hanem az őt tartalmazó eljárás hivása után. Szintaktikailag a különbség csak annyi, hogy a parancs utolsó karaktere felkiáltójel, az utasitásé pedig pont. Így a deklarálás közben is adhatunk parancsokat a gépnek, ezek nem válnak az eljárás részévé, mint az utasitások, hanem azonnal végrehajtódnak.

Mind az utasitás, mind a parancs valamilyen géptevékenységet ir elő. Az utolsó karakter meghatározza a tevékenység végrehajtásának az idejét, a többi karakter maga az elői-

rás. Az előirás meghatározza a tevékenység jellegét, valamint esetleg tárgyát, módját, eszközét stb. Az eszperantóban a parancsoló módú ige végződése "u", a tárgyesetben álló főnévé "n", a mód-, eszköz-, stb. -határozókat pedig gyakran prepoziciós szerkezettel fejezzük ki. Ezt a felépitést értelemszerüen követi a Progreso/1 is.

Pl. egy alapparancs:
LEGU LA TEKSTON PER KARTLEGILO!
Eszperantóul: Olvasd a szöveget kártyaolvasóval! Progreso/1-ben ezt a mondatot a következőképpen értelmezzük. Az "U" végü szó határozza meg a végrehajtandó tevékenységet; ez a jelen esetben olvasás. "LA" az eszperantóban határozott névelő, a Progreso/1-ben azonban csak töltelékszó; el is lehet hagyni. Az "N" végü szó annak a változónak a neve, amelybe a kártya tartalmát olvastatjuk. Példánkban "TEKSTO". Ha a terminális több olvasómüvet tartalmaz, meg kell mondanunk, hogy melyikkel olvastatunk. Ennek a neve a "PER" eszközhatározó prepoziciót követő szó, példánkban "KARTLEGILO". A változóneveket külön deklarálni nem kell. Azonban a programban elsőizben értékadó előirás tárgyaként kell előfordulniuk. Az olvasás értékadó müvelet, tehát az idézett mondatban való előfordulása egyúttal deklarálja is a "TEKSTO" nevü változót (ha még nem volt deklarálva). Szintaktikus hiba volna ellenben, ha a változónév elsőizben pl. irásparancs tárgyaként fordulna elő.

Az eszperantóban, éppúgy mint a magyarban, a tárgyat, a határozót stb. nem szoktuk megismételni, ha azonos az előző mondatéval. Ezt az előnyt a Progreso/1 is megtartja. Pl. a következő olvasási parancsnak elég ennyi:
LEGU!

Pl. egy hivóparancs:
TRANSFORMU LA FORMULON EL PARENTEZA AL POLA NOTACIO!
Eszperantóul: Alakitsd át a formulát zárójelesről lengyel jelölésre! Progreso/1-ben ezt a mondatot csak akkor tudjuk értelmezni, ha előzőleg deklaráltuk; pl. igy:
(PROCEDO POR TRANSFORMI (A)N EL (B) AL (C) NOTACIO: ...)
A deklaráció zárójelek közt áll. A "PROCEDO POR" kezdet a jelöli, hogy eljárásdeklarációról, és ennek melyik fajtájáról van szó. Az "I" végü szó az eljárás neve; hiváskor "I" végződéssel jelenik meg. A zárójelekben a paraméterek találhatók, amelyeket híváskor az argumentumok helyettesitenek. Példánkban "A" változónév, "B" és "C" pedig ugrásfeltétel (az utóbbiak határozzák meg, hogy az A nevü karakter sorozatot milyen jelölésről milyen jelölésre alakitsa át a eljárás. Az utasitásokat itt pontokkal helyettesitettük.

Fájlkezelés

Nemcsak eljárásokat, fájlokat is deklarálhatunk. A fájl rekordokból áll, ezeknek mind hossza, mind száma lehet állandó vagy változó. A fájlba lehet irni, vagy megadott kritériumoknak eleget tevő rekordokat kiolvasni. A keresőparancs tartalmazza a fájl nevét, a kritériumokat, és ezek logikai kapcsolatát. Konjunkcióval, diszjunkcióval, és negációval minden kapcsolatuk képezhető. Pl. keressük azoknak az autótulajdonosoknak a nevét, akiknek a kocsija Trabant vagy Wartburg, 3 évnél nem régibb, szine nem fehér, rendszáma C-vel vagy I-vel, de nem CA-val kezdődik. A megtalált neveket lehet pl. ábécésorrendbe szedni és kinyomtatni. A fájlt utólag is lehet új rekordokkal bőviteni, sőt a rekord szerkezete is bővithető. A fájldeklaráció, valamint a fájlkezelő utasitások ill. parancsok is korrekt eszperantó mondatok.

Irodalom

[1] Bar Hillel, Yehoshua: Language and Information
Addison-Wesley Publishing Co., Reading, 1964

[2] Vásárhelyi Pál: Az eszperantó, mint a nemzetközi együttmüködésen alapuló számitógépes dokumentáció eszköze
Könyvtári Figyelő, 12 (1966) 4. 195-203.

Halpern, Mark: Foundations of the Case for Natural--Language Programming
IEEE Spectrum, 4 (1967) 140-149

[4] Minsky, Marvin (ed.) Semantic Information Processing
The MIT Press, Cambridge, 1968

[5] Meadow, Charles T.: Man--Machine Communication
John Wiley, London, 1970

[6] Kreis, Peter: ADABAS - Einführung in das System
Az Institut für Automatische Informationsverarbeitung prospektusa

Grafikus display assembly nyelve

Endrődy Tamás, Bernus Péter
MTA AKI

1. Bevezetés

A GD'71 Grafikus Display assembly nyelve kétdimenziós/2D/ vonalas ábrák leirására szolgál. A 2D vonalas ábrák képelemekből, ismétlődő- és összetett képrészekből állnak. A képelemek pontok, egyenesszakaszok, körivdarabok, karakterek lehetnek. A müszaki rajz követelményeknek megfelelően 4-féle vonaltipus és 4-féle vonalvastagság /intenzitás/ lehetséges. 3D és perspektivikus vonalas ábrákat először axonometrikusan kell előállitani.

A DAL /Display Assembly Language/ eleget tesz a 2D vonalas ábrák geometriai kódolási követelményeinek. Könnyen tanulható nyelv és jó hatásfoku forditóprogrammal rendelkezik, amely kihasználja a GD'71 Grafikus Display berendezés és a VT 1010B vezérlő számitógép programozási lehetőségeit.

A felhasználó számára igen kedvező, hogy nem kell figyelembe venni a GD'71 gépi kódu utasitásrendszerét és a Display File /DF/ felépitéséhez a gépi kódu képiprogram elhelyezésére vonatkozó előirásokat, korlátozásokat. A DAL forditóprogramja a DF szerkesztés kötöttségeinek figyelembevételével olyan formátumban helyezi el a képi programot, amely a későbbiekben a képiprogram megváltoztatása szempontjából előnyös. A DAL által készitett képiprogramok pointereket /cimző mechanizmusokat/ és flageket /jelölőket/ is tartalmaznak, és minden olyan képelem, illetve képrész számára, amelyet a felhasználó azonositott /vagy azonositani akar a későbbiekben/ külön szótári feljegyzést készit. A DAL által készitett azonositókhoz

és táblákhoz a GD'71 konzol grafikus programrendszerének más egységei is hozzáférhetnek.

2. A DAL nyelv felépitése, szerkezete

A DAL nyelv direktivákból és utasitásokból áll. A direktivákat és utasitásokat külön sorban programozzuk. Az utasitásokat alapszó, a direktivákat * és alapszó kódolja. A DAL direktivái az assembler számára adnak információt és ugy módositják annak müködését, hogy ezáltal bővitik annak szolgáltatásait. Az utasitásai képelemeket, képelem-sorozatokat kódolnak, vagy deklarált képi szubrutinok aktivizálását végzik. Az alapszavak egybetüs röviditését is elfogadja az assembler. A DAL nyelvü képi-programokról kapott nyomtatási kép mindkét esetben a teljes alapszót tartalmazza.

A direktivák formátuma és rövid értelmezése a következő:

*P aaaa: /PICTURE/ képiprogram kezdete, hatására a DAL assembler megkezdi vagy folytatja a megnevezett képi-program forditását.

*F : /FINI/ képiprogram vége

*S aaaa: /SUBROUTINE/ képi szubrutin-deklaráció kezdete, hatására a DAL assembler a megjelenitett képi programrészről átvált a vázlatoló lapra, kezdőpontja a képi szubrutin referencia-pontja /lásd *XY direktivát/

*L nl : /LOOP/ ciklikusan ismétlődő képi programrész kezdete, az ismétlések száma: nl

*E : /END/ képi szubrutin-deklaráció és ciklikusan ismétlődő képi programrész vége

E : /END/ a képi szubrutin deklaráció és ciklikusan ismétlődő képi programrész vége, amelyet az assembler kioltott egyenessel zárttá tesz.

*W n1 n2 n3 n4: /WINDOW/ módositja a forditóprogram /és a felhasználó/ számára rendelkezésre álló hasznos képernyőfelület helyét és méretét

*Q n1 n2 : /QUTIENT/ a nagyitás mértéke

*XY n1 n2 : /XY COORDINATES/ a vázlatoló lapon rögziti képi szubrutin deklarációnál a referenciapont koordinátáit

*N n1 : /NUMBER/ hatására az assembler szótári feljegyzést készit és a központi számitógépes referenciaszámot rendel a soronkövetkező képelemhez, képi programrészhez

*Z n1 : /ZERO/ hatására az assembler időlegesen kioltható /n1=1/, vagy kioltottá /n1=2/ teszi a soronkövetkező képelemeket, képi programrészeket. Ezt a hatást üres utasitások elhelyezésével éri el. Az assembler csak a következő *Z hatására tér vissza eredeti müködési módjához /n1=0/.

*O n1 : /OPTION/ segitségével a felhasználó módosithatja a geometriai elemek /pont, egyenes, köriv/ tárgynyelvi formátumát /n1=1/. Normál müködési módnál /n1=0/ az ssembler szabja meg a formátumot /rövid, ill. hosszu adatszavak/.

*C c...c : /COMMENT/ megjegyzéseket kódol

*A : /ABSOLUTE/ abszolut koordinátarendszerü geometriai kódolást vezérel

*R : /RELATIVE/ relativ koordinátarendszerü geometriai kódolást vezérel

* D : /DELETE/ az utoljára irt utasitássort törli.

Az utasitások formátuma és rövid értelmezése a következő:

Megjegyzések: Az utasitások alapszava után két slash karakter között a módositókat tüntetjük fel. Az 1 a vonalvastagságot/intenzitást/ kódolja. Az 1 = 0, 1, 2 és 3 érték lehet, az intenzitás értékének megfelelően. A k a vonaltipust kódolja. A k = , D, P, R lehet, a/folytonos szaggatott, eredmény/ vonaltipusnak megfelelően.

P/1/n1 n2 n3 n4 } : /POINT/ azonos intenzitásu pontsorozatot kódol

T/1/'c...c' : /TEXT/ azonos intenzitásu karakterekből álló szöveget /képernyő-feliratot/ kódol

L/k1/n1 n2 n3 n4 } : /LINE/ azonos vonaltipusu és intenzitásu egyenes-sorozatot kódol

C/k1/n1 n2 n3 n4 n5 n6 n7 n8 : /CIRCLE/ azonos vonaltipusu, intenzitásu és pozitiv körüljárásu köriv-sorozatot kódol

N/k1/n1 n2 n3 n4 n5 n6 n7 n8 : NEGCIRCLE/ azonos vonaltipusu, intenzitásu és negativ körüljárásu köriv-sorozatot kódol

S aaaa : /SUBROUTINE/ előzőekben deklarált szubrutin hivását kódolja.

3. DAL assemblerek

A DAL nyelv assemblereit a következőképpen csoportosithatjuk.

a/ Számitógép szerint:

VT 1010B szatellita Astrol nyelvén és a CDC 3300 központi számitógép FORTRAN IV nyelvén megirt változatok

b/ Müködési mód szerint:

on-line és
off-line változatok

c/ szolgáltatási rendszer szerint:

alap-assembler és az arra épülő
makro assembler változatok.

Hibajelzési rendszerük, felépitésük egységes. Kezelési módjuk biztositja a képi programok geometriai-hibáinak közvetlen javitását és a képi mocell hatékony kialakitását.

KISÉRLETEK ADATSTRUKTURÁKKAL

Hermann Gyula, Krammer Gergely

MTA Automatizálási Kutató Intézet

1. Bevezetés

Nagyobb modellekkel dolgozó számitógépes programok, elsősorban tervező és interaktiv rendszerek, szükségszerüen valamilyen adatstrukturát használnak.

Az általános adatstruktura felfogható, mint valamely, un. primitiv adatok halmaza és ezen a halmazon definiált relációk összessége. Az adatokkal együtt az előbb emlitett relációkat is tároljuk egy gráf formájában, az adatok igy a gráf élei mentén könnyen elérhetők. Ebből következően az adatstruktura számitógépes megvalósitása, a szekvenciális cimzésü memóriában egy, az adott adatstrukturának megfelelő adatkezelési mód szimulációja.

2. Általános meggondolások

Bár az általános adatstrukturarendszer lehetőséget nyújt az adatok tetszőleges elrendezésére, egy-egy alkalmazáson belül ténylegesen két, esetleg három szerkezeti forma fordul csak elő. Az általunk ismertetett rendszer is két fő szerkezeti elemre épül fel: adattároló és kapcsoló cellára.

Az adattároló cella tulajdonképpen egy vektor, komponensei a modell egy elemét jellemző adatok, struktura leiró mutatók. A mutató lehet valamilyen cim, vagy egy algoritmus neve, amely a megkivánt elem elérését biztositja. Az elemek láncát megvalósitó cella egy tovább fejlesztett vektor, meghatározza az elemek sorrendjét, ezenkivül dinami-

kus memória gazdálkodást is lehetővé tesz: a lánc elemeiként egyszerü láncokat is megenged.

Az adatstruktura rendszerek alkalmazásának az adatok asszociációja és struktura szerinti elérése mellett a másik fő előnye az előbb emlitett dinamikus memóriagazdálkodás. Ha felhasználói program sürün változtatja a modellt, akkor az azt leiró adatok és azok elrendezése is gyorsan változik. A munka során bekövetkező struktura változást a pointerek módositásával irjuk le. Az adatok változása új cellák előállitását teszi szükségessé, miközben régiek válhatnak feleslegessé.

A rendelkezésre álló szabad memória maga is láncként szervezhető. Egyforma méretü cellák esetén a memória előre felosztható és a szabad cellák láncra füzhetők. A feladat sokkal bonyolultabb, ha a rendszer több cellamérettel dolgozik.

A tárológazdálkodás természetesen a rendszer belső ügye kell, hogy legyen: a felhasználó felelőségét minimalizálni kell.

Nagyméretü modell alkalmazása háttér tároló használatát teszi szükségessé. Automatikusan kezelt virtuális memória nagy időveszteséggel járna: az adatstruktura általában kis méretü rekordokkal dolgozik, amelyek nagy mennyiségü cim adatot tartalmaznak és a feldolgozás maga is nagy gyakorisággal jelenti a cimláncon való végighaladást.

Kivánatosnak látszik valamilyen félautomatikus segédtároló kezelés: a programozó a modell logikailag szeparálható részeit különböző adatstruktura blokkokban helyezi el. A blokk megnevezése a programozó feladata, mig megkeresése és a blokkon belüli keresés a rendszer feladata.

Az általunk kidolgozott SAS szubrutin rendszert elsősorban számitógépes tervezés céljaira hoztuk létre. Célunk a rendelkezésre álló számitógépeken is használható, könnyen realizálható szubrutinok létrehozása volt. A szubrutinok FORTRAN-ban készültek, tehát könnyen átültethetők más gépre és a felhasználó egyszerüen betekintést nyerhet a rendszer müködésébe.

3. Az adatstruktura szubrutin rendszer részletes ismertetése

Az adatstruktura kezelő szubrutinok használatát először a háttér tároló nélküli változaton mutatjuk be. A program a rendszer megnyitásakor megad egy tömböt, amelyet a rendszer a struktura épitésére használhat fel. A program a

CALL SAS(A,M ,L)

utasitással hivatkozik először a rendszerre. Ezzel egyrészt átadja az előzőleg használt A(M) tömböt a rendszernek, másrészt megadja az adatcella méretét. A felhasználó L szavas adatcellákat hoz létre és ezeket a rendszer rutinjainak segitségével listákká szervezi. A

CALL NEWDAT(K)

utasitás egy új adatcellát vesz elő és cimét K-ba teszi. A

CALL PUTDAT(K,N,A)

CALL GETDAT(K,N,A)

szubrutinok a K cimü adatcella N-edik szavába elteszi A tartalmát, ill. forditva.

A szabad cellák nyilvántartását a rendszer automatikusan végzi. Eközben azonban megsemmisithet olyan cellát, amely nincs kellően védve.

Védve van minden olyan cella, amelynek cime:

1. az A(1), A(2), ... A(10)elemek valamelyikében van
2. a stackben található
3. eleme az univerzális gyürünek, vagy eleme olyan gyürünek, amelynek cime a fent emlitettek valamelyikében található.

A létrehozott adatcellákból gyürüket lehet épiteni. Erre a következő utasitások szolgálnak:

CALL RNGBOT(K,N) A K cimü gyürü aljára tesz az N cimü
CALL RNGTOP(K,N) elemet /jobbra az első/ tetejére.
CALL RNGLFT(K,N,J) A K gyürü teszi az N elemet a J cimü
CALL RNGRGT(K,N,J) mellé balra/jobbra.

Törlésre a következő rutinok szolgálnak:

CALL RNGDEL(K Törli az egész gyürüt
CALL RNGDBT(K) Törli az alsó elemet
CALL RNGDTP(K) Törli a felső elemet
CALL RNGDLT(K,N) Törli az N cimü elemtől balra levőt, ill.
CALL RNGDRT(K,N) jobbra levőt.

A gyürüket a felhasználó rendszerrutinok segitségével szervezi. A felfüzéshez négy pointeres kapcsoló cellák állnak rendelkezésre. Az adatcellához a rendszer mindig hozzátesz négy pointert, amelyek az u.n. felső, ill. alsó gyürü kezdetei. A felső gyürüre füzzük fel azokat a kapcsoló cellákat, amelyek erre az adatcellára hivatkoznak. Az alsó gyürü a leiró gyürü. A felső gyürüt, és igy egy adatcellára való hivatkozásokat a rendszer automatikusan kezeli. Az alsó gyürüt csak a felhasználó kivánságára kapcsolja:

CALL DESRNG(N,K

az N cimü adatcellára csatolja a K cimü gyürüt, mint alsó gyürüt. A gyürükről az adat visszanyerés általában a gyürün való végighaladással történik. Erre a célra egy olvasót /reader-t/ definiálunk.

CALL READER(LR)

a rendszer az LR változóhoz egy reader cimet rendel.

Az olvasó léptetése a következő utasitások segitségével történik:

CALL DRFWRD(LR,K,FL,M Előre, ill. hátra léptetés a gyürün
CALL RDBWRD(LR,K,FL,M) és K-ba kerül a következő elem cime,
FL jelzi tipusát. M a végigjárás módja

A memória gazdálkodás a rendszer feladata, két szabad listát épit fel a kétféle méretü cella részére. Ha valamelyik cellatipus elfogy, megindul a hulladékgyüjtés. Ennek elején minden védett cella marker bitje 1-es lesz. Az igy meg nem jelölt cellák fölöslegesek, újra fölkerülnek a szabad listák valamelyikére.

4. A virtuális memória szervezése

A SAS virtuális memóriakezelés jellemzője az összefüggő memóriablokk mérete és a memóriában egyszerre tárolt blokkok száma. A memóriablokkok közötti hivatkozásokat a rendszer négy pointeres cellák segitségével tartja nyilván. Az A tömb 12 eleme a mindenkori currens blokk referencia számát tartalmazza. A memóriablokk sorszámával kibővitett teljes cim használatának négy alapvető szabálya van:

1. Az eddig ismertetett rutinok továbbra is használhatók.
2. Ujabb memória blokkra való áttérés egyszerü értékadással történik: A(12) = ...
3. A gyürün való végighaladáskor automatikusan új blokkra tér át, amennyiben ez szükséges.
4. Az eddig ismertetett rutinok párjai, minden cimparaméter mellett egy további paramétert is tartalmaz, mégpedig a memóriablokk sorszámát.

Irodalomjegyzék

1./ L.F. Blake, R.E. Lawson, I.M. Yuille: A ring processing package for use with FORTRAN or a similar high-level language
The Computer Journal Febr. 1970.

2./ C.E. Robinson: A data structure for a Computer Aided Design System

3./ J.H. Sexton: A macro system with a data-structure aid for display programming

4./ R.J. Hubbold: Software paging of list data structures for interactive engineering design
Conf. on Computer Aided Design 1972.

A PROBLÉMAMEGOLDÁS-ELMÉLET ALKALMAZÁSA NAGY FELHASZNÁLÓI PROGRAMOK KÉSZITÉSÉT KISZOLGÁLÓ SOFTWARE-RE

Andréka Hajnal
Németi István
NIM IGÜSZI Számolóközpont
Software-fejlesztési Osztály

A cimben megnevezett területen felmerülő kérdések közül ragadjuk ki a problémamegoldáselmélet hurokmentes feladatdekomponálással foglalkozó fejezetét és ennek alkalmazását a softwaren belül modul-könyvtár-fejlesztési elvekre, amelyek a programrendszer készités és fejlesztés részfeladatait hivatottak koordinálni.

Valamely programkészitési feladatot megfelelő "programspecifikációval" rögzithetünk. Egy programkészitési feladatnak általában több programspecifikáció felel meg, ezek közül választunk egyet. Programspecifikáció /PS/ alatt $f:X^* \to Y$ alaku függvényt értünk, ahol X^* az X véges abc-vel generált szabad félcsoport /a program bemenő szövegeinek halmaza/ és Y a program válaszainak /kimeneteinek/ halmaza.

Valamely programkészitési feladat hurokmentes dekompozicióját /HD/ a PS-t, mint automata-leképzést /a továbbiakban: automata/ fogjuk fel. Az $f:X^* \to Y$ automaták HD-jának kiterjedt elmélete van, az előadásban idézett tételek nagy része megtalálható Arbib /69/ VIII. fejezetében.

Definició:
Automaták hurokmentes kompozicióját /HK/ alatt egymásra ható automaták olyan együttesét értjük, amelyben egyetlen automata sem befolyásolja a saját inputját.

A HD programokra való alkalmazásának előnyei:
1/ A részprogramok /modulok/ egymástól függetlenül elkészithetők,

kipróbálhatók és bejárathatók;

2/ A modulok függetlenül fejleszthetők, módosithatók, kicserélhetők stb. A modulok összekapcsolása az egyes modulok input-output specifikációjának alapján elkészithető, megváltoztatható stb. Ezeknek megfelelően a programfejlesztés természetes és kényelmes módon végezhető;

3/ Ha a program rétegezése HD alapon történik, akkor létezik a programnak olyan megszervezése, amely mellett minden rétegcserére legfeljebb egyszer kerül sor;

4/ Ugyanazon programrendszer különböző részei teljesen különböző időpontokban is futtathatók /mentés, megbizhatóság, részletekben rendelkezésre álló gépidő stb./;

A PS-nek megfeleltetett automata dekompoziciója után a kapott komponens-automaták adják a részprogramok specifikációját, amelyekből egy, az eredeti specifikációt kielégitő program felépithető /az automata felépitését követve/.

1. tétel: Az automaták osztálya HK szerint nem generálható végesen.
Megjegyzés: ha nem kötjük ki, hogy a kompozició hurokmentes legyen, akkor végesen generálható.

A továbbiakban véges automata alatt minimalizált véges automatát értünk.

2. tétel: A véges automaták osztályának generátorrendszerei között ven legkisebb a halmaz-tartalmazásra nézve.

Definició: A fenti tételben megnevezett legkisebb generátorrendszer elemeit primautomatáknak nevezzük.

3. tétel: Tetszőleges véges automatához található olyan legkisebb primautomata-halmaz, amely őt generálja. Ezt nevezzük az automata primkomponenseinek.

4. tétel: Tetszőleges véges automatához van legegyszerűbb felépitési mód /amelyben a primkomponensek legkevesebbszer ismétlődnek/.

A 3. tétel értelmében, valamely PS-hoz megkereshetők annak primkomponensei. A HD előnyeinél felsoroltak érvényesülnek, ha először a primkomponenseket készitjük el, majd ezekből /mint modulokból/ épitjük fel a programot. Ha valamelyik primkomponens tul nagy ahhoz, hogy gazdaságos legyen egyetlen modulként elkésziteni, akkor az eredeti PS-n addig módositunk, amig ez a nehézség meg nem szünik. /Ha ez nem oldható meg a PS megengedhető módositásával, akkor a tulnagy primkomponens további dekompoziciója során már nem használhatjuk ki a HD előnyeit./ A PS kis változtatásával a HD tulajdonságok erősen megváltozhatnak.

Modulkönyvtár alatt a softwarenek azt a részét értjük, amely a felhasználó által készitett modulok tárolására, nyilvántartására, felujitására stb. szolgál. A modulkönyvtár felépitése és szervezettségi szintje meghatározó jellegü a programrendszer-készités hatékonyságának tekintetében. Ezért érdemes az idevonatkozó kérdéseket megfelelő elméleti eszközökkel megvizsgálni.

A felsorolt tételekből nem következik, hogy egy programot feltétlenül primkomponenseiből érdemes felépiteni. Ezért hasznos, ha a modulkönyvtár összetett modulok is tárolására is alkalmas, valamint ezek felépitését, az ezzel kapcsolatos adminisztrációt /komponensek felujitása stb./ is elvégzi.

Az 1. tételből következik, hogy lényeges, hogy a modulkönyvtár tetszőlegesen bővithető legyen, nem képzelhető el valamilyen "alapvető" modulok elkészitése után a modulkönyvtár-fejlesztés lezárása. Hasonlóan következik az is, hogy szükség van valamilyen selejtező "modul-

felejtési" rendszerre, amely eleinte müködhet emberi beavatkozás utján, de a modulok számának és összetettségének növekedésével egyre erősebb az automatizálás igénye.

A 3. tételből és a HD előnyeinél felsoroltakból következik, hogy célszerü, ha a modulok önmagukban is futtathatók, hiszen az ilyen futtatás előkészitésére és kiértékelésére az elvi lehetőség megvan, előnyeit pedig már ismertettük. Megjegyezzük, hogy a legegyszerübb megoldás a programokat futtatásra előkészitett moduloknak tekinteni.

Definició: Azt mondjuk, hogy az A automata szimulálni tudja a B automatát - formálisan A/B- ha megfelelő kimenő és bemenő kódolással ellátva az A automatát, azzal elvégeztethető a B automata feladata.

5. tétel: Található olyan háló /H/ és az automaták osztályának leképezése /c/ a H-ra ugy, hogy

1/ H nem véges,
2/ ha A/B, akkor c/A/≥ c/B/,
3/ ha A a P halmaz elemeinek párhuzamos kapcsolásával keletkezik, akkor c/A/=sup cP,
4/ ha A és B komponensei a D automatának, akkor c/D/≥ c/A/; ha továbbá B nem kombinatorikus automata, akkor c/D/≠c/A/.

Definició: Az 5. tételnek megfelelő c függvényt komplexitásmértéknek, c/A/-t pedig az A komplexitásának nevezzük.

Ez a komplexitásmérték, amelyet Arbib /69/ definiál, az automaták primkomponenseiből való felépitésétől függően /4. tétel/ a programok között további megkülönböztetéseket tesz.

Egy programozási feladattal kapcsolatban beszélhetünk annak optimális beágyazásáról. Ez a következőket jelenti: Az A PS-hoz keresünk olyan B beágyazó-automatákat, amelyekre B/A. Ezek közül kiválasztjuk az optimális beágyazó-automatát aszerint, hogy a modulkönyvtárban meglévő automaták közül hányat tudunk felhasználni komponensként, hány és milyen komplexitású uj modult kell készíteni stb. Az optimalizálás nem korlátozódik B keresésére, hanem A felvételére is kiterjed, hiszen mint emlitettük, egy programozási feladatnak általában több PS is megfelel.

A beágyazás-elv alapján olyan további komplexitásmértékek bevezetésére nyilik lehetőség, amelyek figyelembe veszik a meglévő modulkönyvtár és a PS viszonyát, valamint azt, hogy a készitendő uj modulok esetleg más programokban is felhasználhatók-e. /Az utóbbi ugy oldható meg, hogy a komplexitásmértéket programok halmazára definiáljuk./

A PS felvételét és kezelését megkönnyiti, ha azt nem $f:X^* \to Y$ alakban keressük, hanem a következőképpen járunk el. Alkalmasan kiválasztunk egy véges $S \subset X^*$ szótárt. Felveszünk egy J jelhalmazt és egy i megfeleltetést ugy, hogy $S \xrightarrow{i} J$, amely definiálja az $S^* \xrightarrow{j} J^*$-t ugy, hogy $j|_S = i$. /$\to$ a kölcsönösen egyértelmü megfeleltetés jele és $j|_S$ a j függvény korlátozása az S halmazra./ Ezután $g:J^* \to Y$ alakban adjuk meg a specifikációt, ugy, hogy $f = g \circ j|_S^*$ /o a függvénykompozició jele/ S megfelelő megválasztásával elérhető, hogy g megadása csak a programozási feladatra nézve lényeges megkötéseket tükrözze. Tekintve, hogy $S^* \not\subset X^*$, g a program viselkedését csak S^*-ból vett bemenetekre adja meg. Ez megfelel a célnak.

RHODES, J. Tilson B.R. /72/: Improved lower bounds for the complexity of finite semigroups. in Journal of pure & applied algebra 1972.

ARBIB, M.A. /69/ Theories of abstract automata. Princeton-Hall 1969.

A SOFTWARE EKVIVILENCIA- ÉS OPTIMALIZÁLÁS-PROBLÉMÁIRÓL

Gyuris László

NIM IGÜSZI Számológközpont
Software-fejlesztési Osztály

Az optimalizálásnak lényeges szerepe van a software-tervezésben. Optimalizálási problémák szempontjából az ekvivalencia-összefüggések vizsgálata indokolt, mert ezek segitségével összeállithatók olyan algoritmusok, amelyekkel megoldható:

1. Ekvivalens programok /vagy programrészek/ közül - az adott szempontból - optimális program /vagy programrész/ kiválasztása.
2. Az optimális program és az eredeti program /ha nem az 1. szerint optimalizálunk/ ekvivalenciájának megállapitása. /Természetesen e problémában korlátozó feltételek vannak./

Az ismertetendő ekvivalencia-összefüggések megtalálhatók az /5/ munkában.

Megjegyzem, hogy e dolgozat minden eredménye - nagyon egyszerüen, gyakorlatilag hasznossá tehető számológépek segitségével. E célra assembler-nyelveken vagy ALGOL-ban irt programokat javasolok. /A bit- és a byte- "manipuláció" szükségessége miatt./ - Az ilyen programok elkészitésénél is fel lehet használni ezeket az eredményeket. /Azaz már ezeket is optimalizálhatjuk./

Tulajdonképpen a /6/-ban vázolt "reguláris programok nyelvének" részletes ismertetése az "előkészitő" feladat. /A reguláris programok fogalmát V. GLUSKOV vezette be /1/. Itt az eredetitől eltérő definiciót használunk, de könnyen belátható, hogy tartalmilag a két definició ekvivalens./ A nyelv szintaxisát a /2/-ben bevezetett müveletek /ill. azok általánositásai/segitségével difiniáljuk. /E müveletekben/ a müveleti paraméterek jelentkezése lényeges szerepet játszhat programozás /elmélet-/i vizsgálatokban. Ezt illusztrálja és alátámasztja az itt definiált P/$\mathcal{U},\mathcal{L}$/-rendszer felépitésének módja is. A reguláris programok "elmélete" lehetővé teszi szintaktikai és szemantikai /elméleti és gyakorlati/ programozási problémák /4/ együttes vizsgálatát.

Az 1. §-ban definiáljuk a reguláris programokat, a 2. §-ban a P/$\mathcal{U},\mathcal{L}$/-rendszer elemi közötti összefüggésekkel foglalkozunk.

1. §

A reguláris programok rendszerének definiciója előtt röviden indokoljuk a felépités módját.

Mint az a /2/-ben bevezetett mikroprogram-algebrai rendszer definiciójából látható, a mikroprogramok leirására szolgáló nyelv univerzális, azaz a kiindulási operátorok és logikai feltételek

alkalmas megválasztása esetén bármely állapottranszformáció kifejezhető mikroprogramként /ill. mikroprogramok sorozataként/. - Géptervezési és egyéb problémák szempontjából /2/ az $/\mathcal{U},\mathcal{L}/$ -rendszer feltétlenül hatékony eszköz. Az alábbi továbbfejlesztését csak programozáselméleti szempontok /6/ indokolják.

Bonyolult programok ill. programrendszerek leirásánál, és általában szintaktikai programozási problémákkal kapcsolatban szükségesnek mutatkozik, hogy figyelembe vegyük azt a tapasztalati tényt, hogy lényegében egy mikroprogram egy "ciklusom" belüli müveletek elvégzését /és csak azt/ tartalmazza, azaz egy utasitás konkrét leirására szolgál /2/.

Ezért a gluskovi rendszerben előállitható összes lehetséges kifejezések halmazának azt a részhalmazát tekintjük, amelynek elemei ilyen értelemben utasitásoknak feleltethetők meg. - Ebben a halmazban a szorzás, a feltételes diszjunkció és a feltételes iteráció segitségével kapott kifejezések a programok "durvaszerkezetét" /és csak azt/ irják le. Könnyen belátható, hogy tetszőleges program "durvaszerkezete" ilymódon kifejezhető.

Nyilvánvaló, hogy programok "jelentését" /mint állapottranszformációt /4/ a "finomszerkezetet" kifejező mikroprogramok segitségével formalizálhatjuk. Tehát az ilymódon felépülő $P/\mathcal{U},\mathcal{L}/$ -rendszer segitségével szintaktikai és szemantikai programozási problémák egyaránt megfogalmazhatók.

A $P/\mathcal{U},\mathcal{L}/$ -rendszer definiciójának /1/ részét lényegében a /2/-ben definiált $/\mathcal{U},\mathcal{L}/$ -rendszer képezi, bár azt némileg módositanunk ill. specialisálnunk kell. Legyen adva egy M bázishalmaz /egy számológép állapotainak halmaza/. - Amennyiben a rendszer definiciójánál eltekintünk az ezen M halmazra vonatkozó interpretációtól, akkor "mikroprogram-sémákról" és "programsémákról" van szó. E megkülönböztetést illetően $\mathcal{L}$ /4/.

/1/ REGULÁRIS MIKROPROGRAMOK: $/\mathcal{U},\mathcal{L}/$

I. Legyen $\mathcal{U}$ az operátorok /kiindulási mikrooperációk/ véges halmaza: $\{x_1, x_2 \ldots x_n, e\}$. /A bázishalmaz önmagába történő leképezései, e az identikus leképzés jele./ $\mathcal{L}$ a kiindulási logikai feltételek véges halmaza: $\{\alpha_1, \alpha_2, \ldots, \alpha_k\}$. /A bázishalmazon teljesen /azaz nem parciálisan/ definiált logikai feltételek./

II. Alapműveletek:
$\mathcal{U}$ -ban: asszociativ szorzás /operátorok egymásutáni végrehajtását jelenti/, - igy az $\mathcal{U}$ félcsoport.

$\mathcal{L}$-ben: konjunkció, diszjunkció és negáció. /$\mathcal{L}$-ben teljesülnek a Boole-algebra azonosságai./

Külső müveletek:

1. Operátor és logikai feltétel szorzása: /β=X·α feltétel értéke egyenlő az α értékével az X operátor végrehajtása után.

2. α-diszjunkció: Z=/ X$\vee$Y/ operátor α=1 esetén az X operátorral, különben az Y operátorral egyezik meg.

3. α-iteráció: Z= $\{X\}_\alpha$ operátor megegyezik az $\underline{e}$, X, X^2.... X^n.... sorozat első olyan elemével, amelyre már α=1 teljesül.

III. Reguláris mikroprogramoknak nevezzük az $\mathcal{U}$és $\mathcal{L}$ elemeiből a II. müveletek véges számu alkalmazásával kapott kifejezéseket. Nyilvánvalóan beszélhetünk kétféle /feltétel nélküli és feltételes/ reguláris mikroprogramról.

Megjegyezzük, hogy /2/-től eltérően itt nem "algebráról", hanem "rendszerről" beszélünk. Tehát a müveletek nem adott elemek közötti kapcsolatok kifejezésére szolgálnak, hanem adott /kiindulási/ elemekből uj kifejezések konstruálását teszik lehetővé. Az I-III. alapján definiált rendszert a reguláris mikroprogramok rendszerének nevezzük, és /$\mathcal{U},\mathcal{L}$/-vel jelöljük.

A továbbiakhoz pontosan meg kell fogalmaznunk, hogy az /$\mathcal{U},\mathcal{L}$/-rendszerben felirható összes lehetséges kifejezések halmazának mely részhalmaza feleltethető meg utasitásoknak. Utasitásoknak v. elemi reguláris programoknak nevezzük azokat a reguláris mikroprogramokat, amelyek a /2/-ben bevezetett automatakompozició működését irják le a vezérlő automata valamely iniciális állapotától a terminális állapotáig /azaz egy "ciklusra" vonatkoznak/.

Megállapodunk abban, hogy az utasitások között egy speciális E reguláris mikroprogram, amely a bázishalmaz /azaz az állapothalmaz/ identikus transzformációjának felel meg.

/2/ REGULÁRIS PROGRAMOK: P/$\mathcal{U},\mathcal{L}$/

I. Minden utasitás reguláris program.

II. Ha a P_1, P_2 és P_3 reguláris program, akkor az alábbi müveletek

eredménye is reguláris program.

1. Szorzás: $P=P_1 \cdot P_2$. /A P_1 és P_2 által előírt transzformációk egymás utáni elvégzését jelenti ebben a sorrendben./

2. Feltételes diszjunkció: $P=P_1 \vee_{P_3} P_2$ egyenlő P1-gyel, ha a P_3 feltétele teljesül, különben a P_2-vel.

3. Feltételes iteráció: $P=\mathfrak{x}_{P_2}$ /P_1/ egyenlő az E, P_1, P_1^2,..., P_1^n .... közül az első olyannal, amelyre már a P_2 feltétele teljesül.

III. Reguláris programok: az I. és II. által előírt reguláris programok /és csak azok/. P/$\mathcal{U},\mathcal{L}$/-vel jelöljük a reguláris programok rendszerét.

MEGJEGYZÉSEK

1. A /2/ II. 2, 3-ban a műveletparaméterhez asszociált feltételt a feltétel nélküli reguláris programok esetén mindig teljesítettnek tekintjük, a feltételes reguláris programok esetén az általuk leírt feltétel teljesüléséről van szó.

2. Bármely program /ill. mikroprogram/ felírható reguláris programként /ill. mikroprogramként /1//. Mivel /1/ definíciója és az itteni definíció tartalmilag ekvivalens, bármely program előállítható a P/$\mathcal{U},\mathcal{L}$/ -rendszerben.

Tekintsük a következő példát:

Az $\mathcal{U}=P_1 \uparrow^1 A_1 \; P_2 \uparrow^2 \downarrow_1 A_2 \downarrow_2$ "programséma" /3/ interpretálva "programnak felel még. /7/.

$\mathcal{U}$ reguláris programként a következő alaku:

$$\mathcal{U} = /A_1 \cdot /A_2 \vee_{P_2} E// \vee_{P_1} A_2.$$

Jelentését /szemantikai leirását/ ugy adhatjuk meg, hogy elemeit "konkrétan" felirjuk az /1/ nyelvén:

$$A_1 = \ell_{\alpha_1} x_1 \vee x_2 / \cdot \ell_{x_1} x_3 \cdot x_2 / \cdot x_1,$$

$$A_2 = x_1 \cdot x_2 \cdot \ell_{\alpha_1} x_2 \vee x_3 / \cdot \ell_{\alpha_1} x_4 /,$$

$$P_1 = \ell_{\alpha_1} x_2 \cdot x_3 / x_4 \cdot x_1 \cdot \alpha_1.$$

$$P_2 = \ell_{\alpha_1}\, x_1 \vee x_2 / \cdot \ell_{\alpha_2}\, x_2 / \cdot x_3 \cdot \alpha_2 ,$$

E: az identikus transzformációt leíró utasítás: x_1, x_2, x_3, x_4: az állapothalmaz adott leképezései: α_1, α_2: az állapothalmazon értelmezett logikai feltételek.

E példa jól illusztrálja, hogy a P/U,L/-rendszer felépítése programozási szempontból előnyösnek bizonyulhat.

2. §

DEFINICIÓ / /2/ alapján/: Két reguláris mikroprogramot /és így két reguláris programot/ akkor és csak akkor tekintünk ekvivalensnek, ha ugyanazt az állapottranszformációt írják elő.

Nem térünk ki külön az /U,L/-rendszer /2/-ben ismertetett azonosságaira. Az alábbi összefüggések közül némelyek közvetlenül következnek a rendszer definíciójából.

Az összefüggéseket két részre oszthatjuk:

I. Feltételezzük, hogy a műveletparaméterek csak feltételek kifejezésére szolgálnak, állapottranszformációt nem jelölnek ki.

II. A műveletparaméterek tetszőleges reguláris programok.

I.

Legyen Q, Q_1, Q_2 az előző feltételnek megfelelő, - P_1, P_2, P_3 és P_4 pedig tetszőleges reguláris program.

I. /1/ $P_1 \vee_Q P_1 = P_1$.

I. /2/ $/P_1 \vee_Q P_2/ \vee_Q P_3 = P_1 \vee_Q /P_2 \vee_Q P_3/$.

I. /3/ $/P_1\ P_2/\ P_3 = P_1\ /P_2\ P_3/$.

I. /4/ $/P_1 \vee_Q P_2/\ P_3 = /P_1\ P_3/ \vee_Q /P_2\ P_3/$.

Nyilvánvaló, hogy ha a Q feltétele teljesül, akkor mindkét oldal a $/P_1\ P_3/$-nak, ellenkező esetben pedig a $/P_2\ P_3/$-nak megfelelő állapottranszformációt írja le.

I. /5/ $P_1 /P_2 \vee_Q P_3/ \sim /P_1 P_2/ \vee_Q /P_1 P_3/$.

I. /6/ $P_1 \vee_{Q_1} P_2 \sim P_2 \vee_{Q_2} P_1$ csak akkor érvényes, ha a Q_1 és a Q_2 feltétele egymás negáltja.

I. /7/ $/P_1 \vee_Q P_2/ /P_3 \vee_Q P_4/ \sim /P_1 P_3/ \vee_Q /P_1 P_4/ \vee_Q /P_2 P_3/ \vee_Q /P_2 P_4/$.

A feltételes diszjunkcióra vonatkozó asszociativitás /I. /2// miatt célszerű a jobb oldalt felírni pl. a következő alakban:

$///P_1 P_3/ \vee_Q /P_1 P_4// \vee_Q /P_2 P_3// \vee_Q /P_2 P_4/$.

Így már könnyebben megállapítható, hogy ha a Q feltétele teljesül, akkor a $/P_1 P_3/$-nak, különben a $/P_2 P_4/$-nek megfelelő állapottranszformációt jelenti mindkét oldal.

I. /8/ $*_Q /*_Q /P_1// \sim *_Q /P_1/$.

A baloldal a feltételes iteráció definíciója szerint az

$$E, *_Q /P_1/, /*_Q /P_1//^2, \ldots, /*_Q /P_1//^k, \ldots$$

közül az első olyan reguláris program által leírt transzformációt írja le, amelyre már a Q feltétele teljesül. Ha a Q feltétele már kezdetben teljesül, akkor E-ről van szó, különben meg kell vizsgálni a $*_Q/P_1/$-et.

Ez pedig az

$$E, P_1, P_1^2 \ldots, P_1^n \ldots$$

közül az első olyan elem által leírt transzformációt jelenti, amelyre már a Q feltétele teljesül. Ha ilyen elem van, akkor az ennek megfelelő transzformáció a bal oldal által előírt állapottranszformációval azonos: ha nincs, akkor

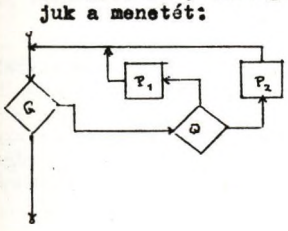
$$\text{sem } /*_Q/P_1//^2, \text{ sem } /*_Q/P_1//^3 \ldots.$$

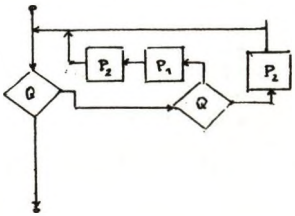
esetén sincs.

A jobb oldalon álló kifejezés jelentését is figyelembe véve, nyilvánvaló az azonosság érvényessége.

I. /9/ $*_Q /P_1 \vee_Q P_2/ \sim *_Q //P_1 P_2/ \vee_Q P_2/$.

Igazolása az I. /8/-hoz hasonlóan történik, ezért nem részletezzük, csak gráfséma-interpretációban illusztráljuk a menetét:





A többi összefüggést csak ismertetjük.

I. /10/ $/\mu_Q /P_1 \vee_Q P_2// \vee_Q /\mu_Q /P_2// = \mu_Q /P_1 \vee_Q P_2/$.

I. /11/ $P_1 E = E P_1 = P_1$.

I. /12/ $\mu_Q /P_1/ = E \vee_Q /\mu_Q /P_1//$.

I. /13/ $/\mu_Q /P_1// \vee_Q /E \vee_Q /\mu_Q /P_1/// = E \vee_Q /\mu_Q /P_1//$.

I. /14/ $\mu_Q /\mu_Q /P_1// = E \vee_Q /\mu_Q /P_1//$.

I. /15/ $P_1 \vee_{Q_1 . Q_2} P_2 = /P_1 \vee_{Q_2} P_2/ \vee_{Q_1} P_2$.

I. /16/ $/\mu_Q //P_1/P_2 \vee_Q P_3// \vee_Q //P_1 \vee_Q P_3/// \vee_Q /\mu_Q //P_1 \vee_Q P_2/ P_3// = \mu_Q ///P_1 P_2/ \vee_Q //P_1 P_3// \vee_Q /P_2 P_3///$.

I. /17/ $\mu_Q /P_1/ = E$, ha a Q feltétele teljesül.

I. /18/ $\mu_Q //P_1 P_2/ \vee_Q P_2/ = /\mu_Q /P_1 \vee_Q P_2// \vee_Q /P_2//$.

I. /19/ $P_1 \vee_{Q_1 . Q_2} P_2 = P_2$, ha a Q_1 feltétele ellenkező esetben teljesül, mint Q_2-é.

I. /20/ $\mu_Q /P_1 \vee_Q /P_1 \vee_{Q_1 . Q_2} P_2// = \mu_Q //P_1 //P_1 \vee_{Q_2} P_2/ \vee_{Q_1} P_2// \vee_Q P_2/$.

I. /21/ $/\mu_Q /P_1// \vee_Q /\mu_Q /P_2// = \mu_Q /P_1 \vee_Q P_2/$.

MEGJEGYZÉSEK:

1. Nem mellőztük a szerkezetileg olyan egyszerű összefüggéseket sem, mint pl. a De Morgan - azonosságok a logikában. /I./21//. Ezek is szükségesnek bizonyulhatnak.

2. A szintaktikai ekvivalenciából következik a szemantikai ekvivalencia is. Ilymódon ezek az összefüggések kapcsolatba hozhatók Ju. Yanov eredményeivel is. /Azonban Yanov formális rendszere nem engedi meg az operátorok tetszőleges ismétlődését./ Emellett, jól megválasztott összefüggések akkor is szükségesek, ha az adott területen formális rendszerrel is rendelkezünk.

IRODALOM

/1/ V. M. G l u s k o v, Avtomatno-algebraicseszkije aszpektü optimazicii mikroprogrammüh upravljajuscsih usztrojsztv, Trudü Mezsdunarodnovo Kongressza Matematikov /Moszkva, 1966./, Izd-vo "Mir" M., 1968., 595-602.

/2/ V. M. G l u s k o v, Tyeorija avtomatov i formalnüje preobrazovanyija mikroprogramm, zsurn. "Kibernyetyika", K., 5, 1965, 1-9.

/3/ A. A. L j a p u n o v, O logicseszkih szhemah programm, szb. "Problemü kibernyetyiki", vüp.1, M., 1958, 46-74.

/4/ A. P. J e r s o v, A. A. L j a p u n o v, O formalizacii ponjatyija programmü, zsurn. "Kibernyetyika", K., 5, 1967, 40-57.

/5/ L. G y u r i s, O reguljarnüh programmah, Studia Sci. Math. Hung. 6 /1971/ 3-4, 307-316.

/6/ G y u r i s L., A gluskovi mikroprogramrendszer módositásáról, MTA Számitástechnikai Központja Közlemények, 3., Budapest 1967. nov., 61-67.

/7/ G y u r i s L., Interpretált algoritmus-sémák ekvivalencia-problémájáról, A Magyar Tudományos Akadémia III. Osztálya Közleményei, 18, 1968, 269-272.

EGY FORDITÓPROGRAMIRÓ RENDSZER ÉS ALKALMAZÁSÁNAK TAPASZTALATAI

Bánkfalvi Zsolt
INFELOR

1. Bevezetés

A magasszintü programozási nyelvek rohamos térhóditása következtében a 60-as évek elején az érdeklődés középpontjába kerülnek a formális kérdések. A fellendülő matematikai nyelvészeti kutatások eredményeként létrejönnek a formális nyelvek szintaxisát leiró egzakt módszerek. A programozási nyelvek szintaxisának formális leirása nem csupán e nyelvek pontosabb definiálásához járult hozzá, hanem egyben előkészitette és megalapozta a forditóprogram-készités automatizálására irányuló törekvéseket is. Olyan forditóprogramok készültek, melyek vezérlésében a szintaktikus elemzést végző felismerők lett a döntő szerepl. Sorra dolgozták ki a szintaktikus elemzés egyre általánosabb és hatékonyabb módszereit, beleértve ebbe a megfelelő felismerők megszerkesztésének algoritmusát is, kizárólag a forrásnyelv nyelvtana alapján. A forrásnyelv szemantikája és a tárgy-nyelv összefüggésének leirása terén, mint ez a kérdés összetettségénél fogva várható volt, csak kevésbé általános eredmények születtek. A kérdéskör megismeréséhez jó bevezetést és gazdag irodalomjegyzéket nyujt J. Feldman és D.Gries tanulmánya az ACM 1968. februári számában.

2. Az UTRA alapelvei

Az UTRA /Univerzális TRAnszlátor/ rendszer alapelveit Dömölki Bálint és Dettrich Árpád fogalmazta meg 1967-68-ban. A rendszerben javasolt elemzési algoritmus Dömölki Bálint korábban kidolgozott szürési eljárása. Ez egy igen egyszerű is mégis nagyon hatékony módszer, melynek lényege abban áll, hogy mind az elemzőt meghatározó, mint az elemzés során nyert információkat bitvektorok alakjában ábrázolva, nagy számu logikai feltétel egyidejű módositását és ellenőrzését teszi lehetővé.

Az UTRA magja egy bármely számitógépen könnyen megvalósitható jelsorozatértelmező automata, melyet jelgenerátornak nevezünk. A jelgenerátor byte-onként vizsgálja a memória tartalmát. Rendre veszi az egymást követő byte-okon tárolt jeleket és e jelek értékétől függően választja meg további tevékenységét.

A jelgenerátor az egy byte-on tárolható teljes jelkészletet az értelmezés módja szerint járom csoportba sorolja:

NORMÁL-jel - átadja a vezérlést egy korábban kivélasztott főkapcsoló rutinnak, mely paraméterül megkapja a normáljelet a tényleges feldolgozás végrehajtása céljából;

VEZÉRLŐ-jel - átadja a vezérlést egy - a vezérlőjel értéke által kiválasztott - vezérlő rutinnak, mely - szükség esetén - a jelsorozatban következő jeleket használhatja fel paraméterül;

POINTER-jel - rátér egy - a pointerjel értéke által kiválasztott - jelsorozatnak a generálására, s egyben az uj jelsorozat normáljelei feldolgozását végző főkapcsoló rutin kiválasztása is megtörténik.

A jelgenerátor a generált jelsorozatok jellemzőinek stackelésével biztositja, hogy a pointerjel hatására megszakitott jelsorozat generálása az uj jelsorozat befejezése után a következő jeltől folytatódjék.

A rendszer felvázolt kereteinek megfelelő tartalommal való kitöltése - az fkp és v rutinok megirása, valamint a jelsorozatok összeállitása - mindekor a felhasználó igényei szerint történik. A rutinok a rendszer géptől függő részét képezik és néhány nyilvánvaló összefüggéstől eltekintve különálló egységekként működnek. A jelsorozatok a rendszer géptől független elemei, melyek a rendszer működési algoritmusának összefüggéseit irják le.

3. Az UTRA megvalósitása

A rendszer tényleges kidolgozása során beigazolódott tervezőinek az a várakozása, hogy viszonylag kis számu jól körülhatárolt tevékenységü fkp és v rutin

felhasználásával a fordítási feladatok nagy része megoldható. Ez azonban nem jelenti azt, hogy az UTRA olyan rutin halmazt nyujt, mely tetszőleges feladat megoldását lehetővé teszi. Az ilyen értelmü teljességre való törekvés a rutinok elaprozódásához és/vagy megsokszorozódáshoz vezetne, ami nehézkessé tenné a rendszer használatát és csökkentené annak hatékonyságát.

A gyakorlati alkalmazások tapasztalatain alapul a rutinok következő csoportositása:

fkp - lerakó, elemző, feldolgozó;
v - vezérlésátadó, táblaelem-képző, előkészitő, szótárkezelő és speciális.

Az UTRA megvalósitásakor kiderül, hogy a közönséges assembly nyelvek nem nyujtanak kielégitő segitséget a rendszer egy sereg lényeges elemének programozásához /jelsorozatok és pointereik, szótárak, elemző rutinok táblái, stb./. Ezen az assembly nyelvek alkalmas bővitésével segithetünk. Az ilymódon nyert jelsorozatnyelvek fordításához célszerü magát a rendszert felhasználni. Ha ezt megszervezzük, az UTRA - részt vállalván saját alkotó részeinek leforditásában - képes önmagát módositani és továbbfejleszteni. Ilyen értelemben nevezzük az UTRA-t önfejlesztő rendszernek.

4. Az EMG FORTRAN

Az EMG FORTRAN az I/O megoldásától eltekintve igen közel áll a FORTRAN IV-hez. A FORTRAN fordító modulonként fordítja a forrásprogramot. A tárgy modul egy SIMPLE nyelvü modul-törzsből és az illető modulban előforduló külső hivatkozásokat összefoglaló modul-fejből áll. Az alkalmas módon kiválasztott tárgy modulokból a SZERKESZTŐ program állit össze egyetlen szintaktikusan helyes SIMPLE fordítóval alakitunk át gépi programmá. A futó program felhasználja a FIOS szolgáltatásait.

A FORTRAN foridtó egy modult egészében egy, egy sort pedig egy vagy két menetben dolgoz fel. Egymenetesek általában a deklarációk és az egyetlen alapnévből álló utasitások, két menetes valamennyi további utasitás. A tárgykódból kihagyásra kerülnek a különböző müveleteknek megfelelő tárgyprogram szakaszok egymás után helyezésével előállt redundans utasitáspárok.

A több fázisban való fordités okozta kényelmetlenségeket csökkenti az EMG FORTRAN-mágnes-szalagos változata, mely a fordítókat és a közbülső fázisok outputjait mágnes-szalagon tárolva a felhasználó számára "load and go" üzemmódot biztosit.

5. Az UTRA alkalmazása

Mind a FORTRAN fordító, mint a SZERKESZTŐ program

az UTRA rendszerben iródott. Bár feladatuk meglehetősen eltérő, számos közös rutint tartalmaznak.

A fordító alapjelsorozata soronként olvassa és szüri a forrásprogramot. Valamennyi további forditási tevékenységre a szürő valamely felismerésének következményeként kerül sor. A rendszer 8 szürője közül egy az egyemenetes sorok feldolgozását, egy a kétmenetes sorok első menetét, 6 pedig - az utasitás tipusától függően - a kétmenetes sorok második menetét vezérli.

Az első menet a forrás- és tárgy-nyelv alapelemei közötti megfeleltetést oldja meg. Ez a folyamat lényegében a programozó által bevezetett nevek szótári válaszának fokozatos kiépitése utján megy végbe.

A második menet a forrásnyelvi sorok szerkezetének feltárásával és az annak megfelelő tárgykód előállitásával foglalkozik. Mindezt a szürő elemző tevékenységét egésziti ki, a másik az egyes müveletek operandusaira való hivatkozásokat állitja össze a tárgykód számára.

6. Összefoglalás

Az UTRA jelentőségét az adja meg, hogy segitségével programjainkat a géptől független, és tetszés szerint bővithető jelsorozatnyelven fogalmazhatjuk meg. Ehhez járul még az a tapasztalat, hogy egy

körültekintően megválasztott rutin készletet az egyes alkalmazásokkor viszonylag kevés feladattól függő rutinnal kell kiegészíteni.

Az UTRA fő erénye és egyben gyengéje lezáratlansága. Felhasználása kétségtelen előnyöket nyujt az olyan programozóknak, akik a tervezés szintjén is biztosan mozognak, de hátráltatja és megzavarja az önálló elképzeléseket nélkülözőket.

Az UTRA alkalmazhatósága nem korlátozódik a fordító-programok irására. Minden olyan feladat megoldásánál hasznos lehet, ahol tulsulyban van a jelsorozatkezelési igény. A szörő tábláinak dinamikus változtatását megszervezve például igen egyszerüen irhatunk olyan javító programokat az UTRA rendszerben, melyek a beszurás, ill. kihagyás helyét karakterstringek segitségével azonositják.

FORDÍTÓPROGRAM-LEÍRÓ NYELV, FORDÍTÓPROGRAM-ÍRÓ PROGRAM, RAGNYELVTANOK

Bedő Árpád, Laborczi Zoltán
NIM IGÜSZI Számolóközpont

1 Bármely szövegkörnyezettől független nyelv (V_N, V_T, S, P) nyelvtana átalakítható az eredetivel ekvivalens olyan nyelvtanná, amelyben elegendő a kezdő szimbólumot és a szabályhalmazt megadni:

i A szabályok jobboldalain az egymásutánírás kétváltozós müveletét jelöljük ,-vel

ii Hagyjuk el a fölösleges[1] szabályokat

iii $(A;B) \in P$-t jelöljük A:B.-tal

iv Vonjuk össze az A:B. és A:C. szabálypárokat rendre A:B;C:-tá /B és C az A változatai/; ezt ismételjük addig, míg minden összevonás megtörténik.

V_N elemei a : előtti szimbólumok, V_T elemei a ,-vel, ;-vel .-tal határolt és V_N elemeitől különböző szimbólumok lesznek, a kezdő szimbólum marad S.

Feleltessünk meg minden egyes $V_N \cup V_T$-beli szimbólumnak egy-egy elemző /logikai függvény-/ eljárást[2] a következő módon: V_T-belieknek olyanokat, amelyek a szimbólum ábrázolását olvasva igaz, egyébként hamis értéket adnak; igaz érték esetén a szimbólum ábrázolását beolvassák; V_N-belieknek olyanokat, amelyek pontosan akkor adnak igaz értéket, ha az adott-szimbólum-baloldalú szabály valamelyik változata következik éppen az olvasandó mondatban.

/pl. $A:b_1,b_2,\ldots,b_n;c_1,\ldots,c_m;\ldots;z_1,\ldots,z_k.$ esetén az A-nak megfelelő eljárás a $B_1 \wedge B_2 \wedge \ldots \wedge B_n \vee C_1 \wedge \ldots \wedge C_m \vee \ldots \vee Z_1 \wedge \ldots \wedge Z_k$ logikai értéket adja, ahol B_i, C_j és Z_h a b_i-nek, c_j-nek és z_h-nak megfelelő elemző eljárások hívásai/

1) A szabály fölösleges, ha nincs a nyelvtanban olyan levezetés, melyben alkalmazni kellene.
2) Algol 60 müszavakat fogunk használni.

2 A felülről lefelé való elemzés a kezdő szimbólumnak megfelelő eljárás végrehajtása. Az elemzés során -- az adott nyelvtan egyedi szerkezetéből adódóan -- előfordulhat, hogy valamely V_N-beli elem egyik változatának elemzésekor beolvassuk a mondat néhány szimbólumának ábrázolását mire kiderül, hogy ez a változat nem megfelelő; ekkor a mondat olvasását nem folytathatjuk, vissza kell lépnünk az elé a szimbólum-ábrázolás elé, amit ezen változat elemzés-próbálkozásakor először olvastunk be. Emlékezőnek nevezzük a felülről lefelé elemző módszert, ha hamis eredmény esetén minden eljárása visszaállítja az olvasásihelyzet-jelzőt, mintha nem is olvasott volna; a nememlékező módszer visszalépésre nincs felkészítve.

3 A mondatot /programot/ felépítő szimbólumok jelentését a fordítóprogramok egyrészt adatrendszerük /táblázataik/ szerkezetében és tartalmában őrzik, másrészt a lefordított /tárgynyelvi/ programba építik be. Az egyes szimbólumok jelentését a felülről lefelé elemző program "megértheti" olyan eljárások /tevékenységek/ végrehajtása révén, amelyek akkor hívódnak meg, ha a megfelelő szimbólum elemző eljárása igaz értéket ad. Értelmező eljárásokat természetesen minden egyes $V_N \cup V_T$-beli szimbólumhoz kapcsolhatunk. Ezen értelmező eljárások hívásait úgy köthetjük a szabályokban az egyes szimbólumokhoz, hogy a szabály jobboldalán az adott szimbólum után beszúrunk a szabályba egy veszszőt és utána az eljárás nevét.

Az értelmező eljárások számára általában kevés az az információ, hogy melyik szimbólumhoz kapcsolódnak. Az elemzés során kialakult helyzetről, valamint a szimbólum konkrét tartalmáról is "tudniok" kell. Az előbbit globális változókkal és tömbökkel, az utóbbit az egyes eljárásokra vonatkozóan lokális változókkal és paraméterekkel lehet megoldani. Bővítsük az elemző és az értelmező eljárásokat paraméteres eljárásokká; így a paramétereken keresztül információkat tudnak átadni egymásnak. A nyelvtan szabályaiba a paramétereket szúrjuk be a szimbólumok, illetve ne-

vek után, mindegyiket + jellel megelőzve. A szabályhalmaz felsorolása előtt deklaráljuk a globális változókat és tömböket, a lokális változókat - jellel megelőzve közvetlenül a : előtt soroljuk fel.

4 A fordítóprogram-leíró nyelv /Compiler Description Language, CDL/ a fenti gondolatok megvalósítása révén fordítóprogramok s így nyelvek definiálására alkalmas. A CDL-nek öt fő alkotórésze van:

1. úSzövegkörnyezettől-független-alakú, paraméterekkel és lokális változókkal ellátható nyelvtani szabályoknak /kijelentéseknek/ a fenti formában írt sorozata /a szimbólumokat azonosítókkal ábrázolva/ adja a CDL program legfontosabb részét;

2. Az értelmező eljárások /tevékenységek/ definíciói a kijelentésekkel keveredhetve, velük teljesen azonos formában szerepelnek /, következésképp előre jelezni kell, hogy mik lesznek a tevékenységek/ a CDL programban;

3. Paraméteres makrók segítségével lehet definiálni a készítendő fordítóprogram programrészleteit azon a nyelven, amelyen a fordítóprogramot kapni akarjuk;

4. A külső környezet /szabványos eljárások/ jelenlétét jelző előírások is szerepelnek a CDL programban;

5. "Láthatatlan" rész: a makrók megfelelő nyelvi szerkezetekbe való illesztésével a kijelentésekből mechanikusan logikai függvényeket, a tevékenységekből a kijelentésekéhez teljesen hasonló módon érték nélküli eljárásokat készítő program, amely végül kiadja a kívánt, felülről lefelé elemző fordítóprogramot.

Mivel a CDL program fordítóprogrammá fordul le, a CDL alkalmas nyelvek teljesen formális leírására.

5 A CDL fordítóprogramja fordítóprogram-író program. E fordítóprogramot s így a CDL-t Koster CDL-ben definiálta/1/, 160 szabály megadásával. A NIM IGÜSZI Számológközpontban meglévő változatban az 5. rész Algol 60 kód előállítására van felkészítve. A CDL saját definíciójának egyszerű módosításával elérhető, hogy az 5. rész gépi kódú szerkeze-

teket hozzon létre, mégpedig a következő módon: jelenleg van egy F:CDL→A60 Algol 60-ban írt, CDL-ről Algol 60-ra fordító program és megvan ennek a CDL-definíciója: $F_{CDL,A60}$. Elkészítjük az $Fl_{CDL,PLAN}$ definíciót; alkalmazzuk F-et: $F(Fl_{CDL,PLAN})=Fl$, ahol Fl Algol 60 program, Fl:CDL→PLAN; alkalmazzuk Fl-et: $Fl(Fl_{CDL,PLAN})=F2$, ahol F2 PLAN program és F2:CDL→PLAN. Ha F2 megvan, akkor ha definiálunk egy nyelvet CDL-ben, akkor egyúttal két /Algol 60 és PLAN/ fordítóprogramját is elkészítjük. A CDL-definíciók fordítóprogram-családok gyors megvalósítására is lehetőséget adnak.

6 Ragnyelvtannak nevezzük a $(V_N,V_T,A_N,A_T,Q,E,R,S,P)$ kilencest, ahol V_N a nyelvtani-, V_T az alapszimbólumok véges, nem üres halmaza; A_N a nyelvtani-, A_T az alapragok szimbólumainak véges halmaza; Q az alapvető kijelentések szimbólumainak véges halmaza /Ezek a halmazok páronként idegenek és egyik sem tartalmazza ω-t, a tiltott szimbólumot./; $E\in V_N$ a kezdő szimbólum; $R\subset A_N\times(A_N\cup A_T)^*$, a ragszabályok véges halmaza /Minden a A_N-re a $G_a=(A_N,A_T,a,R)$ négyes szövegkörnyezettől független nyelvtan. Legyen $L=\bigcup_{a\in A_N}L_a$./; $S=\{S_x \mid x\in Q\cup V_N, S_x=(x,N_x,\tau_x,\alpha_x,F_x)\}$ a vezérlés, ahol x-et fejnek nevezzük; N_x az x raghelyeinek a száma; $\tau_x\in\Pi^{N_x}\{\delta,\iota\}$, a raghelyek típusai, ahol δ és ι különleges szimbólumok; $\alpha_x\in\Pi^{N_x}A_N$ a raghelyek tartományai; $F_x:L_{\alpha_x,1}\times\ldots\times L_{\alpha_x,N_x}\to\{e,\omega\}$ egy teljesen rekurzív, az x-hez kapcsolt függvény /ha $x\in V_N$, akkor F_x-nek nincs jelentősége/; $P=\{(U,V) \mid U=(v,a_1,a_2,\ldots,a_{N_v},v\in V_N,a_i=\alpha_{v,i}$ /S_v-ből/, $V=(m_1,m_2,\ldots,m_k), k\geq 0, m_i\in M\cup V_T, M=(Q\cup V_N)\times\Pi^*(A_N\cup L)\}$ a szabályok halmaza; M a ragkifejezések halmaza.

A ragnyelvtanbeli levezetés:

Ha a ragkifejezés minden raghelyén L-beli elemek állnak, akkor a ragkifejezést konkrét ragkifejezésnek nevezzük. A kezdő szimbólum konkrét ragkifejezés. A kijelentés vagy konkrét ragkifejezés -- ekkor termelőnek nevezzük --, vagy alapszimbólum, vagy a tiltott szimbólum, vagy a semmi /e/. Az alapvető-kijelentés-fejű konkrét ragkifejezés

közvetlen terméke[1] a hozzá kapcsolt függvénynek a raghelyeken lévő ragokkal vett értéke. A nyelvtani szimbólum-fejü konkrét ragkifejezés kötvetlen terméke kijelentéseknek olyan listája, amelyhez található $(U,V)\in P$, valamint $\bar{m}_i$ $i=1,\ldots,n$, melyekre fennáll, hogy minden $\bar{m}_i \in L_{a_i}$ és a_i-ken kívül más nyelvtani rag nem szerepel (U,V)-ben, és U-ban és V-ben minden előfordulása helyén a_i-t $\bar{m}_i$-vel helyettesítve U-ból az adott konkrét ragkifejezést, V-ből a szóbanforgó listát kapjuk. A konkrét ragkifejezés termékének vagy közvetlen termékét nevezzük, vagy kijelentéseknek olyan listáját, amelyet az adott ragkifejezés valamely termékében az egyik termelő kijelentés közvetlen termékével való kicserélésével kapunk. A konkrét ragkifejezés végterméke olyan termék, amelyben minden egyes kijelentés vagy alapszimbólum, vagy a semmi. A ragnyelvtan mondatának nevezzük a kezdő szimbólum bármely végtermékét. A ragnyelvtan nyelve mondatainak a halmaza. A ragnyelvtan a CDL általánosítása a következő /durva/ megfeleltetéssel: kijelentések — szabályok; tevékenységek — alapvető kijelentések, ragszabályok, kapcsolt függvények; globális változó és tömb, lokális változó — alaprag; paraméter — nyelvtani rag. A ragok bevezetésével egyrészt a szintaxis szövegkörnyezettől való függetlenségén lehet túllépni, másrészt az alapvető kijelentésekkel és a kapcsolt függvénnyel a szemantikát lehet nyelvtanszerüen megfogalmazni. A ragnyelvtan generáló ereje megegyezik a Turing gépekével/2/.

Irodalom

/1/ C.H.A. Koster: A Compiler Compiler /MR 127, Matematisch Centrum, Amsterdam/

/2/ Laborczi Zoltán: Programnyelvek teljesen formális leírása, fordítóprogram-iró program /Eötvös Loránd Tudományegyetem TTK matematikus szakdolgozat/

1) Angolul direct production.

ELEMZŐ VEREM-AUTOMATÁK SZINTÉZISE A NYELV FORMÁLISAN MEGADOTT SZINTAXISA ALAPJÁN

Solymosi András
INFELOR

Legyen X egy ábécé /betük véges halmaza/. Az $X^* = \bigcup_{n=0}^{\infty} \{x \mid x:\{1,2,\dots,n\} \to X\}$ halmaz elemeit X feletti szavaknak nevezzük. A 2^{X^*} /X^* összes részhalmazainak halmaza/ elemeit X feletti eseménynek nevezzük. Ebben a halmazban bevezetjük a 3 ismert bináris müveletet: ; /egyesités/, , /szorzás/, * /iteráció/. R_0-lal jelöljük az elemi események /azaz a V nullabetüs szót és az összes egybetüs szót tartalmazó egyelemü események/ halmazát.

2^{X^*} legkisebb R_0-t tartalmazó részhalmazát, amely a fenti 3 müveletre nézve zárt, az X feletti reguláris események halmazának nevezzük. Jelölése: $R(X)$. Reguláris kifejezéseknek a $\Sigma = X \cup \{;\ ,\ *\ (\ )\ V\}$ ábécé feletti, bizonyos szintaktikai követelményeknek eleget tevő szavakat nevezzük. Triviális a megfeleltetés a reguláris események és a reguláris kifejezések között. Pl: az $\tilde{A}=a\ ;\ b\ ,\ c\ *(a\ ;\ b) \in \tilde{R}(X)$ reguláris kifejezésnek megfelel az $A=\{a\ a\ bc\ bcac\ bcbc\ bcacac\ bcacbc\ \dots\}$ reguláris esemény. $\tilde{R}(X)$-szel jelöljük az összes X feletti reguláris eseménynek megfelelő reguláris kifejezések halmazát.

R-nyelvtannak a $G=\langle X,Y,I,P\rangle$ négyest nevezzük, ahol $X \cap Y = \emptyset$, $I \in Y$, $P: Y \to \tilde{R}(X \cup Y)$. A G-nyelvtan által generált nyelvnek az $L(G)=\{x \in X^* \mid I \models x\}$ halmazt nevezzük, ahol a $\models$ jellel jelöltük /a CF-nyelvek elméletéből ismert fogalommal ekvivalens/ "levezethető" fogalmat.

Tekintsük például a következő R-nyelvtant:

$X = \{$ <u>if</u>, <u>then</u>, <u>else</u>, ¬, ±, (,), <u>true</u>, ≠, =, <u>betü</u>, <u>szám</u>, , ⊥ , $\}$

$Y = \{K,E,Z,S\}$

$P(K) = V\ *(\underline{\text{if}}\ ,\ K\ ,\ \underline{\text{then}}\ ,\ E\ ,\ \underline{\text{else}}),\ E$

$P(E) = (V\ ;\ \neg\ ;\ \pm),(($, K , $)\ ;\ \underline{\text{true}}\ ;\ Z\ ,\ (V\ ;\ \neq\ ,\ E);\ S)$ $*(\geq\ ;\ +)$

P(Z)= betü , V * (betü ; szám),(V ;[, E *$_1$,]
P(S)= szám * V

A P-hez tartozó reguláris kifejezések gráfjainak szerkezetét hiven tükrözi a következő /számitógépben jól előállitható/ "szintaktikus gráfsémának" nevezett jelsorozat. Az alábbi táblázatot sorfolytonosan kell értelmezni, ahol bármely elem indexét megkapjuk, ha az elem sora és oszlopa élén álló számokat összeadjuk. A < jelölés "elágazást", a ↓ "folytatást", a + pedig "ugrást visszatéréssel" jelöl. Az ezen jelek után álló számok indexekre való hivatkozást jelölnek, a . jel pedig a részgráf végét. Az összes többi elem a G nyelvtan alapjele /X-be tartozik/.

00	1	2	3	4	5	6	7	8	9
00	<	12	if	+	1	then	+	15	else
10 ↓	1	+	13	.	<	20	¬	↓	23
20 <	23	±	<	31	(	+	1	)	↓
30 34	<	36	true	↓	45	<	47	+	60
40 <	45	≠	+	15	↓	49	+	83	<
50 59	<	56	=	↓	57	+	↓	23	.
60 betü	<	71	<	68	betü	↓	69	szám	↓
70 61	<	82	[	+	1	(	81	$\perp$	↓
80 74	]	.	szám	<	88	↓	83	.	

Ezen szintaktikus gráfséma alapján lehet szintetizálni a következő automatát. A táblázat sorai az automata állapotainak felelnek meg, oszlopai pedig a bemenő jeleknek. A sorok és oszlopok metszetében az automata átkapcsoló függvényének értékei találhatók. Ha ez egy tört, akkor a tört számlálója az automata következő állapota,

nevezőjét pedig be kell irni a verembe. Ha a mező nincs kitöltve, akkor az átkapcsolófüggvény e helyen nincs értelmezve. Ilyenkor, ha az illető sor száma be van karikázva /azaz az automata végállapotban van/,akkor a verem csucsáról vett értéket kell módositani az utolsó bemenőjellel, hasonlóképpen. Ha az automata nem végállapotban van, akkor "leáll", ami azt jelenti, hogy az elemzett szó nem tartozik a nyelvhez. Ha az automata az összes betüt beolvasta, akkor meg kell vizsgálni, állapota és a veremben található összes állapot végállapot-e. Ha igen, a beolvasott szó megfelel az automatához tartozó nyelvtan szintaxisának.

A táblázat első és második oszlopában található számok a szintaktikus gráfséma elemeire hivatkoznak. Az első oszlop az adott állapothoz tartozó "kiinduló csucsok" sorszámát, a második pedig az adott állapotot jellemző csucsok sorszámát tartalmazza. A számlálókban álló csucsokba kell "visszatérni". Az állapot végállapot, ha a jellemző csucsok között található pont / . /. /ld. a cikk utolsó oldala/

Ellenőrzésképpen megvizsgáljuk, hogyan müködik az automata a következő, a fent ismertetett L(G) nyelvbe tartozó szó hatására /ebben a szóban a szemléletesség kedvéért a betü és szám szimbólumok helyett a vagy b, ill. 1 jeleket irtunk/.

a 1 1 [if a ≠ b + 1 then 1 else a] + 1

Mivel az utolsó jel hatására az automata végállapotba billen, és a verem valamennyi állapota végállapot, ez a szó eleme az L (G) halmaznak.

A témával kapcsolatos eddig elért eredmények a következők:

1. A reguláris események és reguláris kifejezések elméletének egy az automataszintézis céljaira alkalmas formalizálása.
2. Az R-nyelvtanok és az általuk generált nyelvek formalizálása.
3. Az R-nyelvtanok és az automaták állapotainak közvetlen kapcsolata.
4. Egy algoritmus, amely egy feltételt kielégitő R-nyelvtanok esetén a nyelvheztartozást a szó hosszával arányos számu lépésben eldönti; az algoritmus helyességének bizonyitása.
5. Ezen algoritmus ALGOL-68 nyelvü megválasztása.
6. A veremautomata müködése ALGOL-68 nyelven.
7. A veremautomaták elméletének szintézis céljaira alkalmas formalizálása.
8. Veremautomaták szintézise ALGOL-68 nyelven az R-nyelvtanból.
9. Veremautomaták müködése ALGOL-68 nyelven
10. A szintaktikus gráfsémák elméletének formalizációja
11. A nyelvheztartozás eldöntése a szintaktikus gráfséma alapján ALGOL-60 nyelven.
12. A szintaktikus gráfséma előállitása az R-nyelvtanból ALGOL-60 nyelven
13. Veremautomata szintézise a szintaktikus gráfsémából ALGOL-60 nyelven.

			if	then	else	¬	±	∫	∂	true	≠	≡	beta	gam	(	→	]
1.	0,9	$3 \cdot \frac{17 \cdot 22 \cdot 25 \cdot 33 \cdot \frac{60}{39} \cdot \frac{83}{48}}{13}$	1/2			3/4	3/4	1/5,4		6/6			7/8,4	9/6,4			
2.	5	6		10/11													
3.	17,23	$25 \cdot 33 \cdot \frac{60}{39} \cdot \frac{83}{48}$						1/5		6			7/8	9/6			
(4.)	13,81	0															
5.	27	28							6								
(6.)	33, 45, 49, 48	53 · 56 · 0					3										
(7.)	60	65 · 68 · 73 · 0											7	7	1/12		
(8.)	39	42 · 53 · 56 · 0					3				10/6	3					
9.	83	83 · 0												9			
10.	6	$\frac{17 \cdot 22 \cdot 25 \cdot 33 \cdot \frac{60}{39} \cdot \frac{83}{48}}{8}$				3	3	1/5		6			7/8	9/6			
11.	8	9			1												
12.	75	79-81														1/12	4

SZÁMITÓGÉP HARDWARE ÉS SOFTWARE ÖSSZEHANGOLT FEJLESZTÉSE

Braun Péter
Villamosenergiaipari Kutató Intézet

Az Intézet 1968-ban vásárolta meg a szovjet gyártmányu RAZDAN-3 számitógépet. Az intézeti beruházási alapok koncentrációja volt szükséges ehhez a beszerzéshez és alapvető érdek füződött ahhoz, hogy a kétségtelen kockázatot jelentő beruházás gazdaságos és lehetőleg nyereséges legyen.

A számitógép munkaspektrumát két tényező határozta meg. Az Intézetben már korábban is folyó, bérelt gépeken futtatott számitások adták az egyik részt, illetve a nagy számitógép adta lehetőségek kihasználásával uj munkák elvállalása vált lehetővé, és ez adta a munkaspektrum másik részét.

A kialakuló munkaspektrum és az induló számitógép hitelének megteremtése szükségessé tette a berendezés megbizhatóságának nagymértékü növelését és időről-időre a szolgáltatások növelését olymódon, hogy az országban dolgozó többi számitóközpontokhoz képest az erkölcsi kopást ellensúllyozzuk. Az alábbiakban azokról a müszaki és egyéb intézkedésekről számolunk be, melynek segitségével a fenti célokat kivántuk elérni.

A számitógép lyukszalag perifériáit Facit gyártmányuakkal cseréltük le. Valamennyi input rutint ennek megfelelően start-stop üzemben karakterenkénti olvasással és egy adott szótár szerinti konvertálással épitettünk fel. Második lépésben mind az inputot, mind az outputot 8 csatornás szalagok kezelésére is alkalmassá tettük és ezzel

egy időben kibővitettük a kezelő rutinok szótárait. A lyukkártya olvasás megbizhatóbbá tételére 1000 kártya/perc sebességü start-stop üzemü lyukkártya olvasót illesztettünk a számitógéphez és az input rutinok közé ennek kezelő programját is felvettük.

Ezekkel a bővitésekkel a számitógép 5 csatornás telex és tele kódban, valamint 8 csatornás ISO és Eichner kódban lyukasztott szalagokat fogad minden átkapcsolás vagy konzol üzenet nélkül. Konzol üzenet segitségével beviteli csatornaként a lyukkártyaolvasó is kijelölhető.

Az egyre bővülő feladatkör szükségessé tette egy korszerü operációs rendszer felépitését. Ennek megvalósitásához egy sor hardware és software feladatot kellett megoldani. Átalakitottuk a számitógép megszakitási rendszerét, hogy az eddig megállást okozó variációk /pl. túlcsordulás/ az operációs rendszerre történő ugró utasitásaként müködjenek. Az operációs rendszert tulságosan elbonyolitotta volna, ha a megszakitásokat tetszőleges mélységben tettünk volna lehetővé, ezért egy megszakitás után ujabb megszakitást nem kezdeményezhetünk, mig az előzőt az operációs rendszer le nem kezelte. Vezérlőirógépként egy IBM gömbfejes irógépet alkalmaztunk, mely az integrált áramkörökből felépitett illesztő egységgel együtt megbizhatóan müködik.

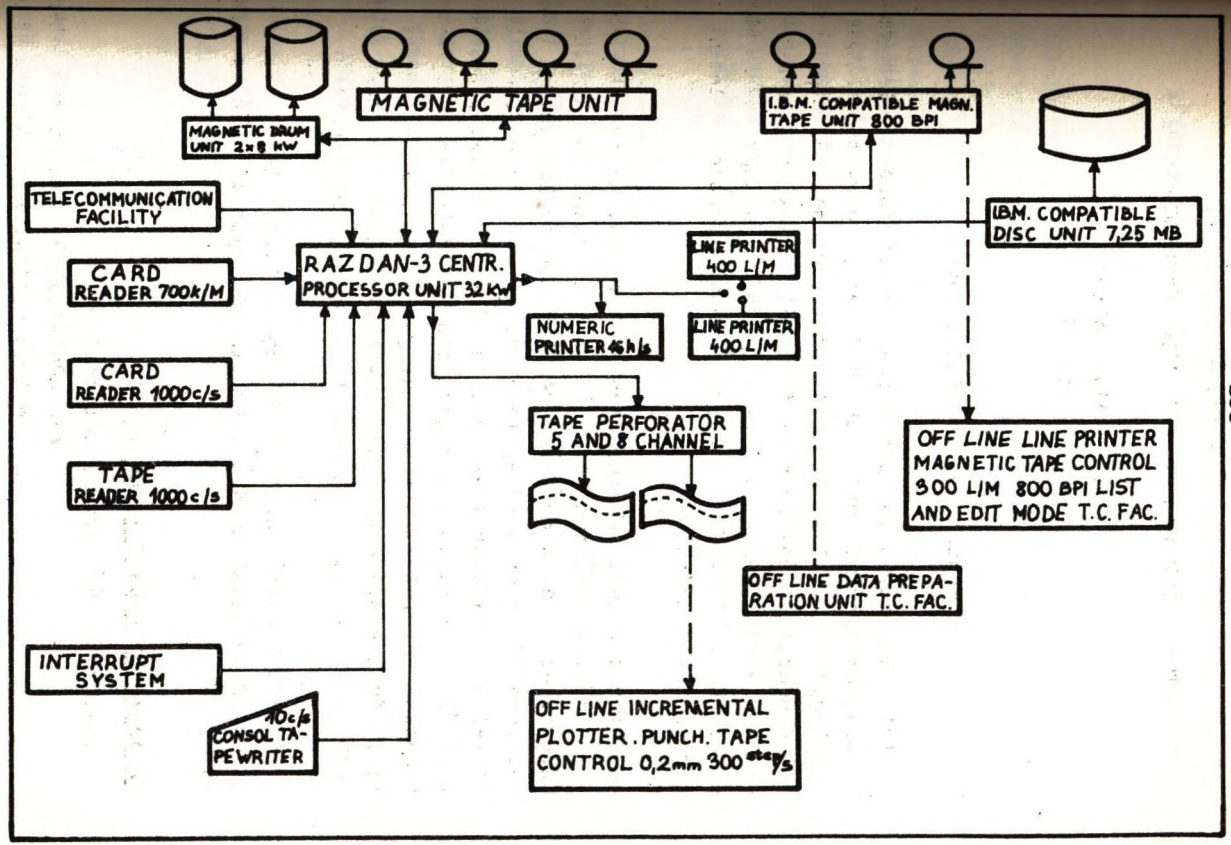
A jelentősen megnövekedett nyomtatási igény kielégitésére egy off-line rendszert hoztunk létre, mely IBM kompatibilis mágnesszalagokból és mágnesszalag vezérelt sornyomtatóból áll. A müszaki munkáknál felmerülő grafikus

eredményközlési igényeket egy lyukszalagvezérlésü 0,1 pontosságu plotterrel elégitjük ki.

A jelentősen kibővült periféria készletet az operációs rendszer automatikusan kezeli, az eredményeket az on-line vagy az off-line sornyomtatóra lehet küldeni és egyszerü módon használhatók a plottert vezérlő rutinok is.

A start-stop vezérlésü lyukkártyaolvasó felhasználásával lyukkártya job leirások adhatók és a számitógép ilyen esetben a vezérkártyák által meghatározott módon müködik.

A számitógép rendszer vázlatát a mellékelt ábra mutatja. A használható nyelv egy assembler szintü nyelv és az ALGOL. Az adatelőkészitő gépek kimélése céljából mágnesszalagos programtároló és programjavitó rendszer is rendelkezésre áll.


MAGNETIC DRUM UNIT 2×8 kw
MAGNETIC TAPE UNIT
I.B.M. COMPATIBLE MAGN. TAPE UNIT 800 BPI
TELECOMMUNICATION FACILITY
I.B.M. COMPATIBLE DISC UNIT 7,25 MB
CARD READER 700k/M
RAZDAN-3 CENTR. PROCESSOR UNIT 32 kw
LINE PRINTER 400 L/M
NUMERIC PRINTER 46 h/s
LINE PRINTER 400 L/M
CARD READER 1000c/s
TAPE PERFORATOR 5 AND 8 CHANNEL
OFF LINE LINE PRINTER MAGNETIC TAPE CONTROL 300 L/M 800 BPI LIST AND EDIT MODE T.C. FAC.
TAPE READER 1000c/s
OFF LINE DATA PREPARATION UNIT T.C. FAC.
INTERRUPT SYSTEM
10c/s CONSOL TAPEWRITER
OFF LINE INCREMENTAL PLOTTER. PUNCH. TAPE CONTROL 0,2mm 300 step/s

VIDEOTON 1010B TERMINÁL VEZÉRLŐ PROGRAMRENDSZERE

Keresztély Zsoltné
Vödrös Csabáné

Videoton Fejlesztési Intézet

Az 1010B kisszámitógép kommunikációs lehetőségeit kihasználó programrendszer, mely lehetővé teszi, hogy tetszőleges távolságban levő ICL System 4 számitógép szolgáltatásait a 1010B számitógépből és perifériáiból kialakitott terminál felhasználója igénybe vegye. Lehetőség van nagy tömegü adat ki és bevitelére, feldolgozására a nagy gépben, valamint job-ok futtatására.

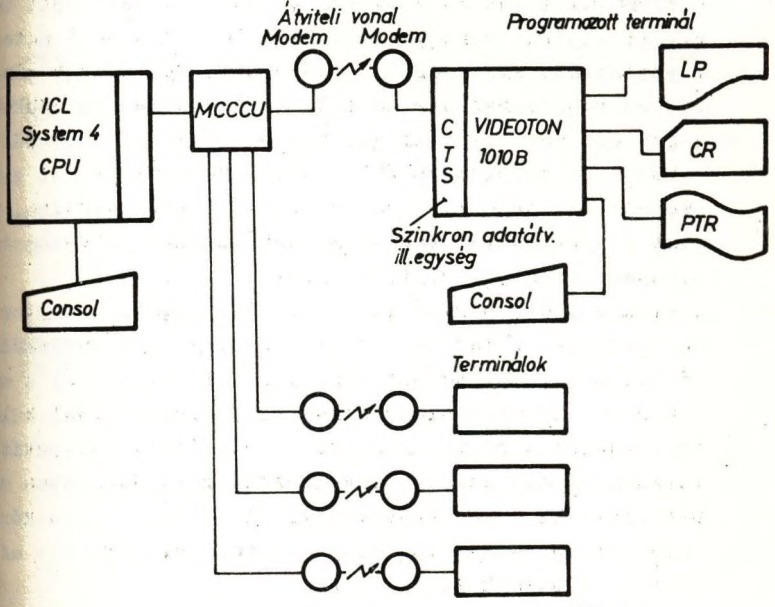
Az 1010B terminál hardware konfigurációja, az összeköttetés jellemzői, valamint a két gép közötti adatátviteli eljárás ismertetése után a dolgozat részletesen foglalkozik a terminál Vezérlő Program felépitésével, a program funkcionális részeinek feladatával és realizációjával. A kisgép sajátságait /cimzési rendszer, megszakitás feldolgozás/ a programrendszer tárgyalása során mindvégig ismertnek tételezzük fel. Végül röviden utalunk a System-4 oldali software feltételekre, valamint a rendszer tesztelési lehetőségeire.

A két gép közötti kapcsolat hardware feltételeit, valamint az 1010B terminál hardware konfigurációját az 1. ábra szemlélteti.

Az összeköttetés jellemzői:

Az összeköttetés tipusa: két pontos
/Az adatátviteli algoritmus lehetővé tesz multi-pontos tipusu összeköttetést is./

Alkalmazott vonal tipusa: bérelt telefonvonal
Átviteli sebesség: 1200/2400 bps
Átviteli mód: szinkron, félduplex
Kód: ASCII
Hibaellenőrzés: Karakter és blokkparitás
Blokkhossz: változó, max. 80 karakter/blokk



1. ábra

Az adatátvitel jellemzői:

a/ Üzenetformátum

Az adatátvitel folyamán két fajta üzenet kerülhet a vonalra:

adatblokk formátuma: SYN-SOH Heading STX Data ETX/ETB BP

vagy STX Data ETX/ETB BP

vezérlőblokk-hét különböző tipusa lehetővé teszi a ter
nál és a terminál egy perifériájának megcimzését; ter
nál illetve periféria status üzenet, valamint adatblok
nyugtázó üzenetek küldését.

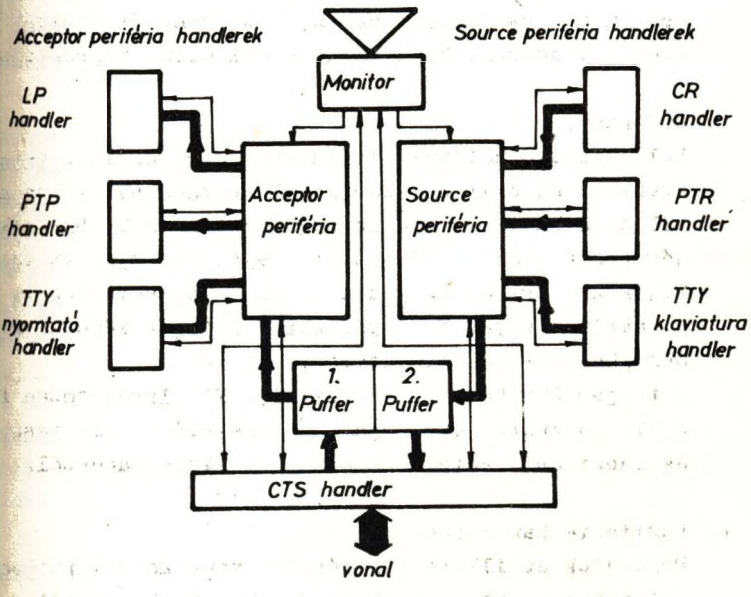
b/ Az adatátvitel algoritmusa

A VIDEOTON 1010B és a System 4 között alkalmazott adat
viteli eljárás megegyezik az ICL 7020 tipusu Renote Da
Terminál-nál alkalmazott eljárással, igy a nagy gép op
rációs rendszerében semmiféle változtatás nem szükség
A két gép közötti dialógus három szekvenciából épül fe
A központi számitógép /CPU/ által kezdeményezett cimzé
szekvencia során létrejön a terminál kiválasztása, val
mint a terminálról kapott Status karakter CPU-ban tört
elemzése után a periféria kiválasztása.
A kiválasztott periféria lehet adat fogadó acceptor ti
su vagy adat küldő source tipusu, igy az algoritmus ké
féleképpen folytatódhat. Acceptor algoritmusnál a nagy
a master, 1010B a slave állomás, a master által küldött
adatblokkra a slave az acceptor periféria állapotát je
lemző nyugtázó üzenettel válaszol, mely üzenetben mód
periféria hiba jelzésére, átvitel felfüggesztés kérel
terminál operátori interrupt jelzésére, valamint az ad
átvitel hibáinak kiszürésére.
Source algoritmusnál a két gép szerepe felcserélődik.
Az adatátviteli hibák kiszürése hossz és keresztparitá
vizsgálattal, blokk sorszámozással és blokkismétléssel
történik.

A terminál vezérlő program

Moduláris felépitésü, igy lehetővé válik a perifériáli
kiépitettség rugalmas változtatása. /2. ábra/

A program az adatátviteli algoritmus egyes szekvenciáinak megfelelő három vezérlőmodulból /monitor, acceptor, source/, az adatátvitelt lebonyolító CTS handlerből, valamint a periféria handlerekből épül fel.



2. ábra

a/ Vezérlőmodulok

A monitor indítását megelőzően a 1010B gép a System 4-től teljesen független helyi számítógépként működik. A monitor indításakor rendszergenerálást, majd a CTS vonalra-

kapcsolását, és vételbeállitását végzi, ezután 6 jel
byte folyamatos teszteléséből álló ciklusba kerül.
Az acceptor és source modulok a nagygép és a terminál
egy perifériája közötti adatcsere szervezését, a puff
rek kezelését, valamint szükség esetén kódkonverziót
geznek. Mindhárom vezérlőmodul a fentieken kivül leke
li az operátori megszakitást, hibák /vonal, periféria
esetén a hibaelemzést végez, hibaüzenetet nyomtat ki
konzol irógépen. Mindegyik vezérlőmodul főprogramban

b/ CTS handler
Feladata az adatátvitel lebonyolitása az ismertetett
átviteli algoritmus szerint. Interface byte-okon kere
tül érintkezik a vezérlőprogramokkal, mely byte-okban
jékoztatást ad az adatátvitel egyes befejezett fázisa
Tájékoztatást ad az adatátvitel során keletkezett oly
hibákról, melyek fellépésekor a két gép közötti kapcs
megszüntetettnek tekinthető.
A teljes CTS handler 1. osztályu IT alprogramban fut,
a CTS hardware adottságának megfelelően 3 fő részből
az időzitési, vételi és adási IT alprogramokból.

c/ Periféria handlerek
Feladatuk az illető perifériára vonatkozó input vagy o
put müvelet lebonyolitása. 2. illetve 3. osztályu mega
kitási alprogramban futnak.

Software feltételek az ICL System 4-ben

A terminál vezérléséhez a System-4 központi egységében sz
séges egy privilegizált üzemmódban futó szubrutincsomag,
a fenti vonali algoritmust megvalósitja, és felhasználói
ramban supervisor hivással hivható. Ilyen az ICL készen
tó CCP programcsomagja.

KIS SZÁMITÓGÉP, A KOMPUTEREK ADATELŐKÉSZITŐJE

Dobó Csabáné
Kőolaj- és Gázipari Tervező Vállalat

1. Bevezetés

A Kőolaj- és Gázipari Tervező Vállalat a magyar szénhidrogéniparon belüli széleskörü tevékenységhez szükséges müszaki és gazdasági tervek készitésével foglalkozik. Ezen belül fő területekként megjelölhető a kitermeléssel, feldolgozással, tárolással és szállitással foglalkozó uj létesitményekre, felujitásokra és bővitésekre vonatkozó tanulmányok, beruházási programok és kivitelezési tervek készitése.

A különböző iparágaknak gyors és dinamikus fejlődése megkövetelte a tervezők munkájában is a növekvő igényekhez alkalmazkodást, a hagyományos munkamódszerek átváltását a hatékonyabbakra. Fokozatosan fellendült a számitástechnika, és 1971-ben az addigi bérelt géphasznált megtartása mellett, egy kis számitógép - Hewlett-Packard 9100/B modell, 9101/A tipusu háttérmemoriával - és hozzátartozó perifériák megvásárolása után, saját berendezéssel is dolgozunk.

Az előadás további részében a fentiekből következő kettős rendszerből kiaknázható előnyökről és tapasztalatokról lesz szó, programozási szempontok szerint.

2. Eredmények és lehetőségek

Az előadás gyakorlati hasznosságát növelendő, konkrét alkalmazásokon és példákon domboritjuk ki az - előzőleg ismertetendő megfontolások során kialakult - alapelvek eredményeit.

Az alapelvek a következők:

a/ a kisszámitógépen történő programozásnál a programszervezés legfontosabb korlátjainak tudatos kihasználása révén előnyös program-

szerkezet alakitható ki.

A gyakorlati alkalmazások közül erre vonatkozóan méretezési feladatról lesz szó.

Meghatároztuk csővezetékekben és készülékekben használt kihuzott nyaku T-idomok falvastagságát az összes járatos csőméret, a nyomásfokozatok és a T-idomok gyártásához használt alapanyagok különbözö anyagminőségének figyelembevételével. A program ugy szegmentált, hogy lehetővé teszi a futtatás kezdését csak attól a ciklustól, amelyben szereplő változók értéke az uj alapadatokból kerül kiszámitásra.

b/Komputerekre irt programok számára - előzetes számitás során - adatok nyerhetők a kisgépen, s ennek kapcsán a komputerek programjai egyszerüsithetők vagy a ciklusok szükithetők.
Példaként részletesen ismertetjük a függvényközelitéseket végző kisgépi és Elliott 803 számitógépre irt programok kapcsolatát és esetenkénti felhasználásukat meghatározó követelményeket - amelyeket önálló vagy egymást követő futásokkal tudnak kielégiteni.

Konkrétan szembe és egymás mellé állitható két azonos cimü program közelités polinommal - amelyek közül a kisgépi változat a legjobban közelitő 12-ed foku polinom együtthatóit szolgáltatja; mig az Elliott 803-as program bemenő adatként igényli a közelitő polinom fokszámának alsó és felső határát. Ez a program hibaszámitást is végez és az erre vonatkozó korlátok vizsgálata alapján dönt a fokszám növeléséről. Következik ebből, hogy ha szükséges a bérelt gép igénybevétele, akkor is lehetőség van a fokszám előzetes kitünő becslésére. Adott esetben viszont teljesen elegendő a számitást csak a kisgépen elvégezni.

c/A Hewlett-Packard gépre alkalmazható algoritmusok közül a választást elsősorban befolyásoló tény - a memóriakapacitás - jelentősége a programlépések és az adatok különböző helyigénye alapján.
E tényből következő kompromisszumokat is ismertetjük szemléltetésként a mérési eredményeket feldolgozó statisztikai programok készitése kapcsán.

Például lineáris egyenletrendszer megoldásán alapuló feladatoknál előnyös a Gauss-elimináció alkalmazása a viszonylag alacsony müveletigény és a matrix-tároláshoz szükséges hely viszonya alapján.

d/A bevezetőben emlitett tervfázisok a számitások modelljeiben eltérő mélységként is jelentkezhetnek. Igy ugyanazt a számitást nagy gépen vagy a Hewlett-Packard calculatoron is végezzük, a különböző mélység- és formaigények szerint.
Ezt illusztrálandó egy gazdasági számitást végző, ICT 1900-as tip. gépre irt algol program Hewlett-Packard változatait emlitjük meg. Döntő jelentősége van a felhasználóktól beérkező kivánságnak, mely a számitási eredmények kiiratási képére vonatkozik. A Hewlett-Packard perifériákon nem tudunk alfanumerikus, táblázatszerü nyomtatást produkálni, de sokszor - ha korai tervfázisban, különböző döntéselőkészitésekhez szükséges - elegendő az eredmények számszerü megjelenitése.

e/A bérelt gépeken futó programok átfutási ideje és a saját gép gyors hozzáférése miatt létrejövő "időtényező" szerepének fontossága a mérnöki munkában.

Az időtényező jelentőségét egy olyan példán keresztül világitjuk meg, ahol a Hewlett-Packard gépi számitás - ha csak ellenőrző, közelitő jelleggel is - használható eredményt szolgáltat abban az esetben, ha az ICT 1900-as gépen a részletes számitást produkáló program futtatása a kért időre nem biztositható.

A feladat a technológiai tervezéseknél gyakran szükséges fázisegyensulyszámitás, melynek egyéb lényeges programszervezési problémáiról is szó lesz /alapadatok folytonos bővitése/.

A VT 1010/B TÖBB-IRÓGÉPES INTERAKTIV ALKALMAZÁSÁNAK PÉLDÁJA

Gerlits J.-né - Kisdi Gábor

INFELOR

Amikor a CII 10010 elektronikus számitógép megjelent hazánkban, a Számitástechnikai Koordinációs Intézet felkérésére kisérletet végeztünk a gép interaktiv üzemmódban való alkalmazásának vizsgálatára. Ennek keretében rendelkezésünkre bocsátottak egy több-irógépes konfigurációt, amelyen az alább vázolt time-sharing-szerü rendszert elkészitettük.

A feladatot két kisérleti programmal oldottuk meg, az első egy barkochba játék /több változatban: magyar, angol, német, és orosz nyelven/, a másik pedig egy asztali számológép szimulátor. A célunk az volt, hogy ezen programok közül bármelyiket egyidőben, bármely, a géphez kötött irógépen használni lehessen.

Több irógépes beszélgetés szervezése:

A programok és a gépelő partner közötti információcsere szervezéséhez modulárisan felépitett fizikai input-output rutinokat használtunk fel. Kisérletünkben az irógépeket az egyik megszakitási osztályra kötöttük. Minden irógéphez hozzárendeltünk egy "Handler"-t, amely az adott készüléken az input, illetve output tevékenységet vezérli, az információkat a hozzárendelt pufferterületre gyüjtve, illetve onnan kiadva.

A "Handler"eket egy felügyelő-program vezérli: beinditja, I/O felfüggesztések esetén a müvelet

befejezését ellenőrzi és visszaadja a szót, ciklikusan.

Ha a "Handler" az input tevékenységet befejezte, akkor tetszés szerinti programra átadható a vezérlés. Ha ez a program egy jól definiált véges tevékenység után valamilyen output információt a pufferba helyezve visszaadja a szót a "Handler"-nek, akkor több irógép esetén is biztositva van, hogy kb. egyenlően osztjuk el az időt közöttük.

Barkochba:

Kisérletünkben a fenti rendszer által "felfüggeszthető egyik program" a barkochba játék szervezése volt.
Készitettünk egy speciális értelmező programot, amellyel jelsorozatokat lehet összeállitani, s eközben bármely gépi tevékenységet is inicializálhatunk, tetszőleges mélységben /veremtechnika/.
A kérdés-felelet szövegsort faágszerüen épitettük fel, ahol az egyes csomópontokat az irógépen feltett kérdések képviselték, mig a választ szótárazással azonositjuk. Azonosság esetén a szótári válasz adja meg a következő szöveg kezdőcimét.

A jelsorozat kezelésben alkalmaztunk egy szükitett belső kódot is. A barkochba szövegét Astrol-8 nyelven lehetett irni ugy, hogy a nyelv szintaxisától eltérő jelsorozatokat egy preprocessor készitette elő a forditóprogram számára. Az információ faágszerü ábrázolása az egyes ágak javitását is rendkivül egyszerüvé teszi.

Asztali számológép szimulátor:

A fenti rendszerhez kapcsolódó másik programunk lehetővé teszi, hogy a programozó az irógépen olyan számitási müveleteket végezzen, amelyekben a négy alapmüveletet használja fel. A kifejezéseit számokból és betükből állithatja össze, amelyek egy-egy értéket reprezentálhatnak. Az irásban felhasználható a négy alapmüvelet jel, továbbá a zárójelek, a matematikában megszokott értelmezéssel. Használhatjuk még az egyenlőség jelet, részértékek lekérdezésére, továbbá a # számjelet, amely az értékadás jelölésére szolgál, ha egy kifejezés értékét egy betühöz akarjuk rendelni.
A betüket a kifejezésekben csak akkor használhatjuk, ha azok már korábban értéket kaptak, és lehet ugy ujra inditani, hogy a betük értéke törlődjön.

Befejezés:

A bemutatott rendszert tovább lehet fejleszteni. Szerkezeti fejlesztésének lehetősége egyrészt, hogy csak egyetlen "Handler"-t készitünk, és a készülékhez táblázatokat rendelünk, amelyeken keresztül vezérli a "Handler" az input-output tevékenységet; a másik módositási lehetőség pedig, hogy a végrehajtandó programot diszken tartjuk és csak a megfelelő részeit hozzuk be esetenként.

A számolási részben pedig a számolás pontosságát lehet növelni és be lehet vezetni az elemi függvényeket.

MŰSZAKI TERVEZÉSRE ORIENTÁLT PROGRAMRENDSZER MODULELVŰ KIÉPITÉSE

Szőts Miklós

Építőipari Számitástechnikai és Ügyvitelgépesitési Vállalat

1. Bevezetés

Programrendszer kiépitése minőségileg más problémát vet fel, mint egyes programok irása. A legfontosabb feloldandó ellentmondások a következők:

Az egész programrendszer és az egyes programok megtervezése korlátos forrásokból származó információk alapján történik és a megirással egyidejüleg lezárul, viszont az elkészülő programok tág körben, hosszabb időre való felhasználása természetes igény. Tehát a programrendszerrel szemben támasztott alapvető igény annak rugalmassága: uj vagy az adott pillanatban fel nem ismert problémák és megoldási módok könnyü beolvasztási lehetősége a programrendszerbe - azaz az egyes programokba.

Ugyanilyen természetes igény a programrendszer teljessége, azaz hogy az adott problémakörben felmerülő összes vagy majdnem összes problémára adjon a gyakorlatban megfelelő megoldási lehetőséget. Ez a programrendszer méretét megnöveli, igy az elkészitéshez szükséges idő irreálisan megnőhet.

Mindkét ellentmondás szerintünk leghatékonyabban a modulelvü programozás felhasználásával oldahtó fel. Maga a modulelvü programozás több számitógép operációs rendszeréből ismert /IBM 360 OS, ICL System 4, stb./.

Itt azonban többről beszélünk: problémára orientált modulrendszerről, azaz olyan modulkönyvtárról, amelyből

építőkő-elv szerint összeállítható az igényelt progra
Jelen előadásban ilyen modulrendszer kiépítésének ala
elveivel foglalkozunk.

2. Modulrendszer kiépítése

2.1 Modulfelosztás megtervezése

A programokon belül a modulok hierarchikus felépítésb
foglalhatók. Programrendszer tervezése esetén bizonyo
fokig célszerünek látszik a modulokat osztályozni a
hierarchiában elfoglalható helyük szerint:

a/ Alapmodulok. Ezek egyszerü matematikai vagy problé-
mára orientált /müszaki/ alapfeladatokat oldanak
meg. Minden, vagy majdnem minden tényleges számitá-
si munkát ezekre bizunk. A felsőbb szintek moduljai
ezek illetve egymás behivását szervezik.

b/ Szervező modulok. Az alapmodulok segitségével rész-
feladatok szervezését végzik.

c/ Feladatszervező modulok. Ezek már egy feladat komp-
lex eredményét szolgáltatják, illetve szolgáltat-
hatják, lehetnek a programrendszert alkotó progra-
mok fő rutinjai.

2.2 Adatkezelés

A modulok egyszerü kapcsolhatósága kivánja meg az
adatkezelés egységesitését. A probléma két ágra osz-
lik: a modulok egymásközti kapcsolatának megoldására,
valamint a program és a felhasználó közötti kapcsolat
kiépitésére / input/output /.

A modulok egymásközti kapcsolatát a következő módokon lehet megoldani:

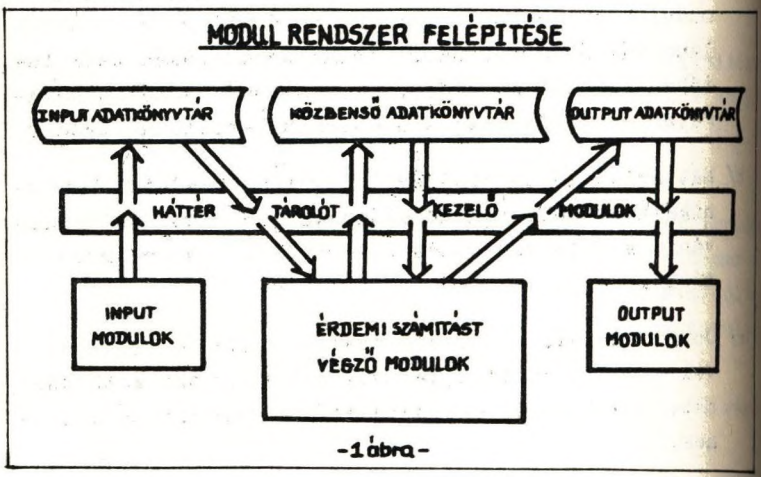
a/ Formális paraméterekkel történő megoldás adja a legtisztább képet. Kizárólagos használatukkal a modulok teljesen függetlenithetők, igy ez az általánosság és a többszöri felhasználhatóság tekintetében kedvező megoldás, viszont sok formális paraméter a rutint teljesen áttekinthetetlenné teheti.

b/ Háttértároló segitségével nagytömegü adatok kommunikálását célszerü megoldani. Ez a megoldás megkivánja a file szervezés bizonyos foku szabványositását.

c/ Common mezők, illetve globális változók segitségével elért kapcsolat jelenti a legnagyobb kötöttséget, ezért ennek használatát nem tartjuk célszerünek.

Program és a felhasználó közti kapcsolat ellentmondásos feladat elé állitja a programtervezőt. A nagytömegü input rendkivül kényelmetlen feladatokat ró a felhasználóra, tehát az adatmegadásban jelentkező egyszerüsitési lehetőségeket okvetlenül fel kell használni. Azonban ez a lehetőség általában specifikumokon keresztül jelentkezik, s igy ellentmond a modulokkal szemben megkövetelt újra felhasználhatósági igénynek. Ezt az ellentmondást az input müveletek és a tényleges számitás teljes elkülönitésével tudjuk feloldani. Nagytömegü output kezelése nehézkes, viszont az esetleg szükséges adatokat is meg kell őrizni. Ezért az

outputot is célszerü elválasztani a tényleges számitá-
tól. Az elkülönitès háttértárolón létrehozott adat-
könyvtár segitségével érhető el.



-1 ábra-

Az előzőek alapján a modulrendszert az 1. ábra szerint
oszthatjuk fel. A modulokból alkotott programok három
adatkönyvtárt kezelnek:

a/ Input adatkönyvtár. Itt található az érdemi számi-
tási munkához szükséges összes adat olyan elrende-
zésben, amely a számitáshoz legcélszerübb. Redundáns
információk tárolásának elkerülése nem lehet szem-
pont.

b/ Közbenső adatkönyvtár. Nagytömegü közbenső számitási eredmények tárolására létesitendő. Ezen keresztül biztositható a modulok közti kapcsolat.

c/ Output adatkönyvtár. Itt jelenik meg a számitás eredménye, amelyből további részeredmények egyszerüen számithatók.

A három adatkönyvtár szerkezetét célszerü ugyanolyannak választani, igy kevesebb modullal megoldható az adatcsoportokhoz való hozzáférés. Hierarchikus felépités, indexelt hozzáférés alkalmazása látszik célszerünek.

Maguk a modulok a következő csoportokba oszthatók:

a./ Adatkönyvtár kezelő rutinok

Az adatkönyvtárak csak ezeken keresztül érhetők el, igy a többi modul függetlenithető magától az "adatkezelés"-től.

b./ Az érdemi számitást végző modulok

c./ Input modulok
Feladatuk a lehető legkényelmesebb adatmegadásból kiindulva az input adatkönyvtár felépitése. Periféria kezelés mellett előkészitő számitásokat és adatrendezést végeznek.

d./ Output modulok
Az output adatkönyvtárból vett értékeket és az ezek egyszerü feldolgozásából kapott eredményeket nyomtatják ki.

3/ Összefoglalás

A modulrendszer kiépitése a következő pontokba foglalt előnyöket biztositja a programrendszer irásához:

a/ A programozói kapacitást legtakarékosabban és legcélszerübben használhatjuk fel, elősegiti a kollektiv munkát.

b/ Ujonnan felmerülő probléma vagy megoldási mód kevésszámu uj modul megirásával beolvasztható a rendszerbe.

c/ Az input/output és a háttértároló kezelés elválasztása az érdemi számitástól minimalizálja a gép kiépitésében történő változás okozta munkát, vagy akár más gépre való átállást.

Felhasználás tekintetében a következő momentumot szeretnénk kiemelni:

A fentiekben vázoltak segitségével elérhető a tervezés menetéhez való jobb alkalmazkodás. A háttértároló ismertetett felhasználási módja szerint a kiinduló, közbenső és végadatok megőrizhetők, esetleg szerkeszthetők és igy összeállitható olyan program, amely néhány adat megváltoztatása esetén csak ezek hatását számitja ujra, igy a felhasználó közelebb juthat ahhoz az állapothoz, hogy a számitógépet "logarléc"-ként használja.

PROGRAMRENDSZEREK MÉRNÖKI SZÁMITÁSOKHOZ

Jancsó Ferencné
Út- Vasúttervező Vállalat
/UVATERV/

A tervező mérnök számára segitséget adni számitógép segitségével többféle módon lehet. Ki lehet ragadni munkájából egy-egy számitásigényes részletet és csupán a szémitásokat asztali számitógép helyett computerre bizni. Az ilyen irányu tevékenység az ugynevezett müszaki számitások elvégzése. Az alkalmazott computernek megfelelően kisebb vagy nagyobb terjedelmü feladatokat, de mindig csak egy leszükitett munkarészt lehet ily módon megoldani. A tervezői munka azonban egy tevékenységi folyamat, amelynek különféle szinteken döntési pontjai vannak. A rutin döntéseket is magában foglaló, láncszerüen egymásba kapcsolódó, automatikus adatkapcsolattal kidolgozott számitási és rajzi munkákat is ellátó programok csoportját tervezői programrendszernek nevezzük. Ezek a programrendszerek valósitják meg a tervezésgépesitést.

A tervezői programrendszerek kidolgozása hosszadalmas, sokszor évekig tartó munka. Első programrendszerként vállalatunknál 1965-ben az autópályák vonaltervezéséhez kezdtük meg a tervezői programok kidolgozását, majd 1967-ben kezdtük rendszerré épiteni. Az első tervezői programrendszer kialakitásánál szerzett tapasztalatok nyomán mind több szakágban dolgozunk ki programrendszereket. Igy ma már a metro vonalvezetéséhez, a geodéziai tervezésekhez, közlekedésfejlesztés tervezéshez /forgalomszámlálás kiértékelése, hálózatfejlesztés/, többnyilásu feszitett és nem feszitett hidszerkezetek tervezéséhez részben vagy egészen kész programrendszerek állnak a tervező mérnök rendelkezésére.

Az alábbiakban összefoglaljuk milyen szempontokat érvényesitünk a programrendszerek kialakitásánál.

A rendszer a feladat komplex megoldását adja, melyet csak komplex munkacsoport -team- tud kidolgozni. Az UVATERV-nél egy team tervező mérnökből, programtervezőből és programozóból áll.

A programrendszer kialakitása során részletesen elemezzük a hagyományos tervezési folyamatot. Ennek során átvizsgálásra kerülnek a tervezési szabályzat oda vonatkozó részei. Arra törekednek, hogy a tervezési folyamat és az azt kiszolgáló programrendszer a legteljesebb összhangban legyenek. Azaz a programrendszer feladat orientált szemléletben készül. Ha szükséges, javaslatot tesznek a folyamat megváltoztatására, vagy például a tervezési szabályzat pontositására.

A tervezési folyamatnak két jól megkülönböztethető fázisa van:

- a tervvariánsok kidolgozása és
- az elfogadott variáns részleteinek kidolgozása.

Mivel a második fázis az elsőre épül és sokszor az elsőben végzett lépéseket részletesebben ujra el kell végezni, a két fázishoz egyetlen programrendszert dolgozunk ki.

Az input és output adatok megtervezésénél a következő szempontokat érvényesitjük:

- a tervező mérnök egy tipusu adatot csak egyszer adjon meg nyomtatványon, ha ujra ugyanarra az adatra szükség van
- a programrendszer egyes tagjai automatikus adatátvitellel gondoskodjanak annak jelenlétéről és
- a programok input-output igény szempontjából is vezérelhetők legyenek.

A tervezői adatszolgáltatást úgy szervezzük meg, hogy az adatlapok egyszerű szerkezetüek, magyarázó ábrákkal ellátottak legyenek, azaz a felhasználó tervező mérnök a gépi program ismerete nélkül is egyszerüen ki tudja tölteni.

A programok vezérelhetőségét és a programok közötti automatikus adatáramlást együttesen kell megoldani. A vezérelhetőség azt jelenti, hogy az elvégzendő számitásokat a tervező egy programon belül még tovább megválaszthatja, azaz a futtatandó modulokat kijelölheti. Erre a modul kijelölésre azért van szükség, mert adott esetben nem szükséges az összes modul futtatása. Ez komoly gépidőköltség megtakaritást jelent és megkiméli a tervezőket a felesleges információk tömegétől. Hasonlóképpen biztositani kell, hogy a programfutás ujra elvégezhető legyen egy adott ponttól, anélkül, hogy az eddig elvégzett számitásokat meg kellene ismételni. Erre például egy későbbi időpontban felmerülő részletesebb számitási igény esetén kerülhet sor.

Az output rendszer megtervezésénél nemcsak a kapcsolódó számitás, hanem a rajzoló programok input adatainak előállitására is gondolunk.

Az eredménytáblákat lehetőség szerint olyan formában tervezzük meg, hogy azok a mérnök tervdokumentációjához átalakitás nélkül csatolhatók legyenek.

A programok matematikai modellje felépitésben és alkalmazott számitási módszerekben nem a hagyományos kézi eljárást követi. Ez ma már legtöbb szakágban készült programnál igy van, de elég hosszu időbe telt és csak a konkrét eredmények győzték meg nagyon sok esetben a tervezőket arról, hogy a programtervező igy hatékonyabb eredményt tud számára szolgáltatni. A leggyorsabb előrehaladást általában azoknak a programrendszereknek készitésénél értük el, ahol a matematikai módszer megválasztását a tervező mérnök a programterve-

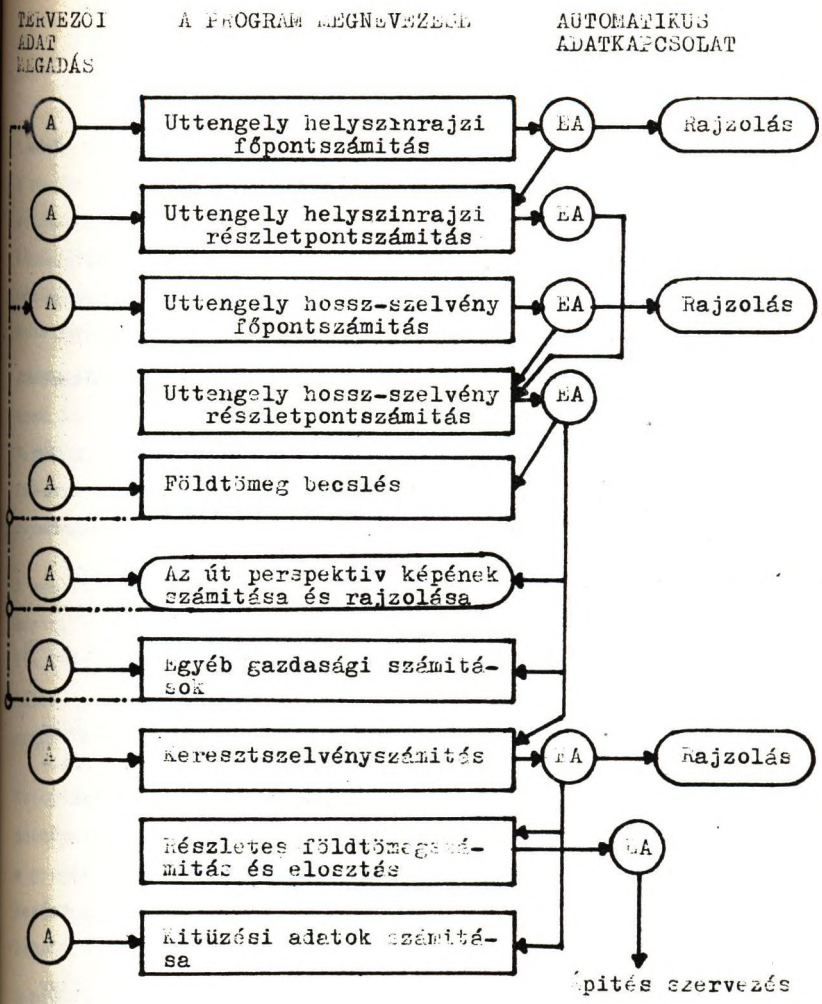
zőre bizta. A programokat modulrendszerüen épitjük fel,mint erre már a fentiekben utaltunk.

A programdokumentálás kérdését a programkészitési munka szerves részeként tekintjük.A program üzemszerü felhasználása a programot készitő team-től függetlenül történik.Ezért a teamban résztvevő tervező mérnök a programtervező közremüködésével olyan"tervezés ismertetés" elnevezésü leirást készit,amely más felhasználó mérnök számára részletesen rögziti,hogyan szervezze munkáját,hogy a programot tervezésnél alkalmazni tudja, továbbá előirásokat tartalmaz a program adatlapjainak kitöltésére vonatkozóan,ismerteti a program müködését,minta eredményeket tartalmaz,előirja,hogy az eredmények miként kerüljenek a tervekben alkalmazásra.

A programtervező részletes adatrögzitési,futtatási utasitás üzemeltetési leirást készit a programok üzemszerü futtatásával megbizott programkönyvtáros részlegnek. Ez utóbbi tér ki például arra, hogy a programba épitett hibajelzések esetén mi a programkönyvtáros teendője. A programdokumentáció tartalmazza még a program blokkdiagramját, matematikai modelljét, a forrásnyelvi programot és minden olyan információt, amely a további fejlesztést vagy a program moduljainak más területen történő felhasználását segiti.

Az előadás során ismertetjük
- az autópálya vonaltervező és a
- metro vonaltervező programrendszert.

Ezek segitségével mutatjuk be a fentiekben összefoglalt szempontok érvényesitését.



AUTÓPÁLYA TERVEZŐRENDSZER PROGRAMJAINAK ADATKAPCSOLATAI

OPTIMALIZÁLÓ MEMORIABEOSZTÓ ÉS KEZELŐ SZUBRUTINRENDSZER

Békefi József

Volán Tröszt Elektronika

Bonyolult, sok tömbbel /vektorral, matrixokkal/ dolgozó algoritmusok számítógépes programjainak elkészitésekor általában nagy problémát jelent a rendelkezésre álló memoriaterületek szétosztása a tömbök között. A tömbök elhelyezésekor általában az a cél, hogy a gépi program futási ideje rövid legyen.

Gondoljunk csak egy - lineáris programozási feladatot megoldó - módositott szimplex-algoritmusra. El kell helyezni a számitógép valamilyen memoriájában /központi memoriában, vagy egyéb háttérmemorián/ az alapmatrixot, a célfüggvényt, a korlátvektort, és egyéb input tömböket, valamint ki kell jelölni a bázisinverz helyét, és helyet kell lefoglalni a számitásokhoz és eredményközléshez szükséges egyéb tömbök számára is.

A tömbök elhelyezésére általában három lehetőségünk van: vagy központi memoria, vagy mágneslemez, vagy mágnesszalag. A nem adatfeldolgozó programok jórésze általában a szekvenciálistól eltérően használja tömbjeit, és a nem szekvenciális használat megoldása mágnesszalagon elég körülményes. Ráadásul a számitógépen megoldandó feladatok nagy része nem is igényel akkora memoriát, amely egy vagy két diszk kapacitását meghaladná.

Mindezeket figyelembe véve, olyan memoriabeosztó szubrutinrendszer kidolgozását tüztük ki célul, amely a rendelkezésre álló memoriaterületekre vonatkozó, és az igényelt memoriaterületeket /tömböket/ jellemző adatok felhasználásával elkészíti egy program memoriabeosztását, és a felhasználó program igényei szerint kezeli is a tömböket /azaz a felhasználó program kérésére hozzáférhetővé teszi a

tömbök tartalmát/.

A szubrutinrendszer csak központi memoria és mágneslemezmemoria kezelésére készült fel. Felhasználja ezen két memoriatipusnak azt a közös tulajdonságát, hogy mindkettő cimezhető, és minden nehézség nélkül alkalmassá tehető a beosztó rendszer egyéb cimezhető memoriák kezelésére is.

A beosztó és kezelő rendszer által nyújtott előnyöket természetesen csak kismértékben használhatjuk ki, ha a probléma megfogalmazása "adatfeldolgozás irányultságú": például; valamilyen file recordjaival kell olyan müveleteket végezni, amelyekhez nincs szükség egyéb recordokra, és igy elhelyezési probléma tulajdonképpen nincs is.

A memoriabeosztó rendszer használata különösen akkor előnyös, ha az adatok elhelyezéséhez szükséges memoriaterületek méreteit az aktuális feladat határozza meg /például; egy LP esetén a változók és feltételek számából, és az algoritmus felépitéséből egyértelmüen kiszámitható az igényelt memoriaterület/.

Az optimalizáló memoriabeosztó- és kezelő szubrutinrendszer egy adott program olyan memoriabeosztását készíti el, amely biztositja a program adatkezelésének maximális gyorsaságát, ezenkivül a rendelkezésre álló memoriaterületek maximális kihasználását, és ezzel a lehető legnagyobb méretü feladat lefuttatását.

A memoriabeosztó- és kezelő rendszer FORTRAN programok memoriabeosztásának elvégzésére készült, mivel a számitástechnikai programok nagy része ezen a nyelven iródik, de nincs akadálya annak, hogy tetszőleges programok adatkezelésére felhasználható legyen.

A memoriabeosztás elkészitéséhez szükséges információk

A memoriabeosztó rutin a tömböket vagy a központi memoriában, vagy diszken helyezi el. Ismernie kell tehát a központi memóriában, illetve diszken rendelkezésre álló memoriaterület méretét. Ha valamely tömb diszken van elhelyezve, akkor biztositani kell számára olyan központi memoriaterületet, amelybe felhasználása előtt kerülhet, ugyanis müveleteket csak a központi memoriában lévő adatokkal lehet végezni. Ezt a központi memoria-területet a továbbiakban pufferterületnek fogjuk nevezni. Egy tömb pufferterülete lehet azonos méretü a tömb teljes méretével, de lehet annál kisebb is.

1. Tömbméretek

Valamely tömb jellemzése a következő adatokkal történik:

a/ A tömb egyetlen elemének memóriaigénye a gép memoriájának alapegységében /byte, szó/

b/ A tömb teljes /deklarált/ mérete,

c/ Darabolás - méret: a tömb legkisebb, önállóan kezelt részletének mérete.

d/ A pufferterület tulajdonságaira vonatkozó információk.

2.. Tömb-együttállások

A memoriabeosztó rendszer azon tömbök számára, amelyek a központi memoriában már nem férnek el diszken jelöl ki helyet, és a központi memoriában pedig pufferterületet foglal le. Az algoritmus azonban általában meghatározza, hogy mely tömbökkel kell egyidőben müveleteket végezni.

A pufferterületek kijelöléséhez szükség van olyan információkra, amelyekből megtudjuk, hogy egyes programrészek mely tömböket

használnak egyidejüleg. Fel kell tehát bontanunk a programot részekre, és minden programrészben fel kell sorolnunk a programrészben együtt szereplő tömböket.

3. A tömbök fontossági sorrendje

A program futási ideje szempontjából nem mindegy, hogy mely tömbök vannak a központi memóriában elhelyezve, és melyek diszken. A program alapjául szolgáló algoritmus általában meghatározza, hogy mely tömbökkel végzünk gyakrabban müveleteket, és melyekkel csak ritkán. Az algoritmus ismeretében megadható egy fontossági sorrend. A memoriabeosztó rendszer figyelembe veszi ezt a fontossági /pontosabban hozzányúlás - gyakorisági/ sorrendet a tömbök elhelyezésénél.

A memoriabeosztási eljárás

A memoriabeosztási algoritmus két fő részből áll. A következőkben vagyvonalakban ismertetjük ezeket a részeket.

1. A központi memoriában rendelkezésre álló terület felosztása

A központi memoria felosztása során az algoritmus a tömböket fontosságuk sorrendjében a központi memoriában helyezi el mindaddig, amig az el nem helyezett tömbök számára tud pufferterületet kijelölni. A pufferterület nem lehet kisebb, mint az illető tömb darabolás-mérete. A központi memoriában helyhiány miatt el nem helyezhető tömbök diszkre kerülnek, és számukra a központi memoriában pufferterületet jelöl ki az algoritmus, az együttállások figyelembevételével.

2. Mielőtt az algoritmus másik fontos részéről irnánk, pár szóval kitérünk a mágneslemez használati módjára.

Feltételezzük, hogy létezik egy TRANSZ /T, M, KC, BE/

szubrutin, amely a cimfolytonos diszk-filet és a központi memória között végez adattranszfert, éspedig a T/M/ tömb és a file KC cimü, megfelelő méretü területe között. Ilyen rutin biztosan irható bármely gépen.

A diszk-területek szétosztásának feladata tehát a következő: Legyenek adottak a $\{T(I): 1 \leq I \leq M\}$ méretü területek, kezdőcimeikkel, és az $\{A(J): 1 \leq J \leq N\}$ terület-igények. Legyen $X(I,J)$ definiciója:

$$X(I,J) = \begin{cases} 1, & \text{ha a J-edik igény az I-edik területre kerül;} \\ 0, & \text{egyébként.} \end{cases}$$

Keresendő a

$$\sum_{J=1}^{N} A(J) * X(I,J) \leq T(I) \quad ; \quad I = 1,\dots,M,$$

$$\sum_{I=1}^{M} X(I,J) = 1 \quad ; \quad J = 1,\dots,N,$$

$$X(I,J) \in \{0;1\} \quad ; \quad I = 1,\dots,M \; ; \; J = 1,\dots,N,$$

egyenlőtlenségrendszer egy lehetséges megoldása.

Az OAP - rendszer

Több alprogramból álló programok esetén gyakran előfordul, hogy az alprogramok egymástól csak kevés tömböt vesznek át, ezért ajánlatos ezen alprogramokat önálló adatkezelésüekké tenni. Az önálló adatkezelésü programrészek /OAP-ok/ között természetesen biztositani kell a kapcsolatot. A szubrutinrendszer tartalmaz ezt a funkciót ellátó rutinokat.

A szubrutinrendszer nyelve FORTRAN. Használata akkor válna igazán kényelmessé, ha beépitenénk valamilyen compilerbe.

INFORMÁCIOVISSZAKERESÉS AZ EGYETEMES TIZEDES OSZTÁLYOZÁS ALAPJÁN

Makay Árpád

József Attila Tudományegyetem Kibernetikai Laboratóriuma

1. A MINSZK-22 számitógépen megvalósult információvisszakereső rendszer az Országos Müszaki Könyvtárral együttmüködésben készült és kisérleti jellegü. Kisérleti jellege elsősorban az Egyetemes Tizedes Osztályozásnak (ETO), mint általánosan elterjedt tartalmi leiró rendszernek információvisszakereső rendszerekben való alkalmazhatóságára vonatkozik. A visszakeresés hatékonyságának számitástechnikai eszközökkel való fokozása éppen ezért háttérbe szorult az ETO rendszer formális és algoritmikus tulajdonságainak felhasználása mellett. Következésképpen nem kerestünk olyan, a MINSZK-22-nél alkalmasabb számitógépet, amelyen az információvisszakeresés üzemszerüen és gazdaságosan lenne végezhető, továbbá későbbi időpontra halasztottuk a visszakeresés idejének csökkentését szolgáló programrendszer-elemek beépitését (elsősorban invertált-file felhasználását). A kisérleti futásokat a Müszaki Könyvtár által feldolgozott ötezres dokumentum minta-anyagon hajtottuk végre.

2. Az ETO az alá- és fölérendelt fogalmak hierarchikus rendjét a tizedestörtek elvén felépitett számjelzetekkel jelöli. Megkülönböztet főtáblázati számokat és alosztásokat. Egy ETO-jelzet főtáblázati számokból és alosztásokból épül fel a rendszer szabályainak megfelelően. Az alábbiakban a jelzetek néhány számunkra érdekes tulajdonságát soroljuk fel.

2.1. Legyen $x = x_1x_2$ főtáblázati szám. Ekkor az x jelzet által jelölt fogalomnak x alárendeltje az ETO hierarchikus rendjében. Pl.: $x = 621.3$ (elektrotechnika), $x_1 = 62$ (műszaki tudományok).
(Megjegyzendő, hogy a "."-nak a főtáblázati számokban csupán formai szerepe van.)

2.2. Az alosztásokat formai szempontból két csoportba oszthatjuk: valamilyen elhatároló (-,=,.0,.00) után tizedesszám, vagy tizedesszám zárójelpárban (zárójel az idézőjel,"(0" ill."(=" és a")"pár is). Mivel az ETO szabályai szerint alosztás után főtáblázati szám már nem következhet a jelzetekben, a zárójelpárok második tagja elhagyható anélkül, hogy a jelzet többértelművé válna. Az ETO jelzeteken ezt a formális átalakítást elvégezve az alosztásokra is érvényes marad, hogy ha $x = +x_1x_2$ alosztás (ahol + valamilyen elhatároló vagy bal-zárójel), akkor a $+x_1$ alosztás fölérendeltje x-nek. Pl. $x = 945.11$ (magyar nyelv), $x_1 = 945$ (uráli nyelvek).

3. A fentiek alapján mind két főtáblázati szám közötti, mind két elosztás közötti hierarchikus fogalmi rendet formálisan megállapithatjuk: ugyanis ha az y egyszerű jelzetnek (főtáblázati számnak vagy alosztásnak), mint jelsorozatnak az x egyszerű jelzet, mint jelsorozat kezdőszelete, akkor az x fogalomnak y alárendeltje az ETO rendszerben. Általánositsuk a jelsorozatokra vonatkozó kezdőszelet fogalmát olyan, általánositott kezdőszeletnek nevezett részsorozattá, hogy bármely x,y egyszerű vagy összetett ETO-jelzetről akkor mondhassuk, hogy az x fogalomnak az y fogalom alárendeltje, ha x általánositott kezdőszelete y-nak. Általánositásunk csak akkor lesz sikeres, ha az ETO szabályrendszeré-

3.2. A fenti definiciót az ETO-jelzetekre ugy alkalmazzuk, hogy betüként a számjegyeket, terminatorokként az ott terminátoroknak és balzárójeleknek nevezett jeleket tekintjük. Természetesen a jelzetekből elhagyjuk az elválasztójel-szerepü "."-ot és a jobbzárójeleket, ami egy formális eljárással elérhető.

3.3. Viszonylag egyszerü számitógépi algoritmus konstruálható, amely bármely két x,y szóról eldönti, hogy x általánositott kezdőszelete-e y-nek vagy nem. Ugyanez az algoritmus nyilvánvalóan alkalmas annak eldöntésére is, hogy x részsorozata vagy kezdőszelete y-nak. Ugyanis az Σ halmaz speciális választásával az általánositott kezdőszeletek részsorozatokká ($\Sigma = \Phi$) illetve kezdőszeletekké ($\Sigma = V$) válnak.

4. Információvisszakereső rendszerünkben mind a dokumentumok leirására, mind az elemi igények megfogalmazására ETO-jelzeteket használunk. Összetett igények ETO-jelzetektől a logikai müveletek segitségével képezhetők.

A fentiek alapján annak megállapitásához, hogy egy elemi igénynek egy adott dokumentum megfelel-e, az szükséges, hogy az elemi igény ETO-jelzetéről megállapitsuk, általánositott kezdőszelete-e a dokumentumhoz rendelt valamelyik (vagy egyetlen) ETO-jelzetnek. Ezzel tulajdonképpen azt döntöttük el, hogy az elemi igény által kifejezett fogalomkörbe beletartozik-e a dokumentum tartalmát lejró ETO-jelzettel kifejezett fogalom.

Az ETO hajlékony rendszerében az általánositott kezdőszelet mechanikus alkalmazása természetesen felvet néhány, még meg nem oldott problémát. Elsősorban amiatt, hogy ugyanazt a fogalmat (az alosztások sor-

hez még egy szabályt hozzáveszünk. Ugyanis az ETO nem irja elő az alosztások sorrendjét, csupán egy ajánlott sorrendet közöl. Az általánositott kezdőszelet fogalma akkor felel meg céljának, ha az ajánlott sorrendet kötelezően előirjuk. Az általánositott kezdőszelet fogalmát akkor, jelsorozatokra vonatkozóan a következőképpen definiálhatjuk.

3.1. Legyen V véges halmaz, Σ' pedig V-nek részhalmaza. V, Σ', V-Σ' elemeit rendre jeleknek, betűknek illetve terminátoroknak nevezzük. Jelölje V^* a V halmaz fölötti szabad félcsoportot, elemeit nevezzük szavaknak. Az x szóban előforduló terminátorok számát $\|x\|$-val jelöljük.

Legyen $x,y \in V^*$. Az általánositott kezdőszelet fogalmát $\|x\|$-ra vonatkozóan teljes indukcióval definiáljuk.

a.) Ha $x \in \Sigma'^*$ azaz $\|x\| = 0$ és x kezdőszelete y-nak, akkor x általánositott kezdőszelete is y-nak.

b.) Tegyük fel, hogy $\|x\| = n + 1$ $(n \geq 0)$, továbbá

$x = x_1x_2$, $y = y_1x_2y_2$, $x_2 = ti_1i_2 \ldots i_m$,

ahol $x_1, x_2, y_1, y_2 \in V^*$, $t \in V - \Sigma'$, $m \geq 0$ és $i_1, i_2, \ldots, i_m \in \Sigma'$.

(Nyilvánvalóan $\|x_1\| = n$.) Ha x_1 általánositott kezdőszelete y_1-nek, akkor x is általánositott kezdőszelete y-nak.

Másszóval ez azt jelenti, hogy ha x azon részsorozatai, amelyek az elejétől az első terminátorig illetve terminátortól terminátorig (csak az első terminátort hozzászámitva) vagy a végéig tartanak, előfordulnak y szeleteiként ugyanolyan sorrendben, mint x-ben, akkor x y-nak általánositott kezdőszelete.

rendjétől eltekintve is többféleképpen lehet kifejezni: egy-egy könyvtárban szakozási "szokások" ("nyelvjárások") alakulnak ki, amelyen belül bátran használható a fogalmak hierarchikus rendjének eldöntésére az általánositott kezdőszelet, de azon kivül nem. Információvisszakereső rendszerünkre ez azzal a következménnyel jár, hogy ugyanaz a csoport szakozza a dokumentumokat, amely az igényeket is megfogalmazza.

Irodalom

Egyetemes Tizedes Osztályozás. Segédtáblázatok. Közgazdasági és jogi Könyvkiadó. Budapest, 1980.

SAMPO ADATKEZELŐ PROGRAM

Hajós Tamás
KERINFORG

1. A program feladata, hogy az input-adatokból előre megadott szempontok szerint kiválassza és átrendezze a szükséges adatokat, és ezekkel szükség szerint aritmetikai és logikai műveleteket is végezzen. Az input-adatok és output-adatok egyaránt kerülhetnek mágnesszalagra vagy mágneslemezre. A program futtatásához maximálisan egy lemezre és két szalagra van szükség.

   A végrehajtandó feladatokat vezérlőnyelv segitségével közöljük a programmal. Két tipusu művelet van. Az I. tipushoz tartoznak a rekordon belüli műveletek. Ekkor egyidejüleg csak egy rekord adataival számolunk, minden rekordra nézve egyformán. Az eredmények is az illető rekordba kerülnek vissza. A II. tipusu műveletek a rekordok közötti műveletek, amikor két vagy több rekorddal végzünk műveleteket a rekord összes elemére, vagy csak megadott elemekre. Itt a számitás során uj eredményrekordok keletkeznek.

2. A számábrázolás kétféle lehet: vagy integer, pld. 358; vagy floating point, pl. 3.1415 . Az adathalmaz egy rekordján belüli adatra pl. x17-el hivatkozhatunk. Ez a rekord 17. elemét jelenti. A rekord minden adatához /az összes rekordra természetesen egyformán/ egy-egy változónév is tartozik. Az egyes rekordokra külön nem lehet hivatkozni, de erre nincs is szükség.

3. I. tipus Rekordon belüli műveletek

3.11. Aritmetikai müveletek: + összeadás, - kivonás, * szorzás, / osztás, ** hatványozás, = értékadás. Igy felirhatjuk a következő utasitást:

```
x17 = x1 + x2/3.1415 **2 - 5.+x61 **x1
```

Ezeket a müveleteket minden rekordra elvégzi. Az aritmetikai müveletek tetszőlegesen zárójelezhetők. Az eredmény-adathoz tartozó változónevet meg is változtathatjuk: pl. x17/TERÜLET/ x1**2*3.1415 . Vagyis az uj változónév a TERÜLET lesz.

3.12. Logikai müveletek: .EQ. egyenlő, .NE. nem egyenlő, ; LT. kisebb, .LE. kisebb egyenlő, .GT. nagyobb, .GE. nagyobb egyenlő, .OR. vagy, .AND. és. Ezen müveletek segitségével tetszőleges bonyolultságu logikai kapurendszert épithetünk fel.

3.2. Sorrend megváltoztató utasitással megváltoztathatjuk a rekord adatainak sorrendjét, és egyben ki is hagyhatjuk a nem szükségeseket.

```
pl. SORREND (2, 10, 5, 1, 48)
```

Tehát az 1. adat helyére a 2. fog kerülni, a 2. helyére a 10., a 3. helyére az 5., 4. helyére az 1. stb...

3.3. Az ugró utasitás segitségével eltérhetünk az utasitások megadott sorrendjétől.

```
pl.  GO  TO  12
```

Ekkor a következő utasitás a 12. lesz.

3.4. Ciklikus ismétlésre is van lehetőség.

pl. DO 9 x51 = 2,4o,3

Ezen utasitás hatására addig ismétli a 9. utasitásig bezárólag levő utasitásokat, amig x51 értéke nem haladja meg a 4o-et. Az x51 kiinduló értéke 2, és ciklusonként 3-al növekszik. A DO ciklusok egymásba ágyazhatók.

3.5. Feltételes utasitás hatására a feltétel teljesülésekor végrehajtja a vele egy sorba irt utasitást. Ha a feltétel nem teljesül, a következő sorba irt utasitás következik.

pl. IF(x1.GT.1Ø.AND.x1.GT.x2) GO TO 7

Azon rekordok esetében, ahol az első adat nagyobb 1o-nél és egyben a másodiknál is nagyobb, ez utasitás után a 7. következik.

3.6. Rekordkihagyó utasitás a SZELEKT, melynek hatására kihagyja az adattömegből az éppen soron levő rekordot, ha a SZELEKT előtt álló feltétel nem teljesül.

pl. IF (x1.LT.x2) SZELEKT

Összefoglalóan a 3. pont alatt szereplő utasitások abban különböznek pl. a FORTRAN egy-egy utasitásától, hogy egy utasitás hatására nemcsak egy müveletet hajt végre, hanem egy müveletet annyiszor, ahány rekord van.

4. II. tipus Rekordok közötti müveletek

4.1. Két rekord közötti müveletek

4.11. Rekordok összeadása

Az utasitás alakja:

pl.RPLUSZ (x3,x4=69,72/5,6,1o,18/KIMENET=0)

Ezen utasitás hatására azon két rekord 5,6,1o,18 sorszámu elemét adja össze, melyek közül az egyik negyedik eleme 69, és a másiké 72, ugyanakkor a harmadik elem azonos mindkét rekordra nézve. A KIMENET=0 esetén a kimeneten rekordpáronként egy eredményrekord keletkezik. KIMENET=1 esetén az eredményrekord előtt a két operandusrekord is fog szerepelni, melyekkel tehát a müveletet végeztük. Azok az adatok, melyek nem vettek részt a számitásban, melyek nincsenek felsorolva a két törtvonal között, az eredményrekordban meg fognak egyezni az első operandusrekord ilyen sorszámu adatával.

4.12. Rekordok különbsége

Az RMINUSZ utasitás a második operandus-rekord kijelölt adataiból kivonja az első operandus-rekord megfelelő adatait.

4.13. Rekordok szorzása

Az RSZOR müvelet a kiválasztott két rekord elemeit összeszorozza.

4.14. Rekordok hányadosa

Az RPER utasitás a második operandus-rekord kijelölt elemeit elosztja az első operandus-rekord megfelelő elemeivel.

4.15. Rekordok bázisra vetitése

Az RBÁZIS utasitás az első operandus-rekordot bázisnak tekinti /1oo %-nak/ és a második operandusrekordot er-

re a bázisra vetiti. A 4.12-4.15 utasitások alakilag megegyeznek a 4.11. utasitással.

4.2. Kettőnél több rekord közötti müveletek.

4.21. Több rekord összege

A REKPLUSZ müvelet hasonló az RPLUSZ-hoz, csak több rekord kijelölt elemeit összegzi.

4.22. Több rekord átlaga

Az REKÁTLAG müvelet több rekord kijelölt elemeinek az átlagát számitja ki. A 4.21.-4.22 utasitások hasonlóak alakilag a 4.11. utasitáshoz.

5. Minden vezérlő utasitást külön kártyára kell irni. Ha egy utasitás nem fér ki egy kártyára , folytatás-kártyát lehet alkalmazni /csak egy darabot/. Ekkor az utasitásokat folyamatosan kell irni a 79. karakterig. A 80. karakterre „,” kell lyukasztani.
Az utasitások maximálisan három ciklusból állhatnak. Egy utasitásciklusba tartoznak az egymásután következő I. tip. utasitások, vagy egy darab II.tip. utasitás.

6. A program működését vázlatosan az alábbi rövid, két ciklusu vezérlőutasitások segitségével tekinthetjük át:

```
EX*SAMPO
   001 ADATFELDOLGOZÁSI PRÓBA
   SORREND(2,3,5,6,7,10)
   XI=2.5**x2+x3-x4
   RPLUSZ (x5,x6=69,72/1,2,3,4/KIMENET=0 )
* ENDATA
* ENDRUN
```

A bemenő adattömeg álljon loo rekordból, rekordonként 12 adattal. A feladat tehát: rekordonként 12 adatból a kijelölt 6 adatot kiválasztani, ezekkel végrehajtani a megadott aritmetikai müveleteket, majd a rekordok összegét kell képezni az első négy elemre. A vezérlés a programbehivó kártyából, a három végrehajtandó utasitásból, a cimkártyából és két programzáró kártyából áll.

A cimkártyán tüntettük fel, hogy a feldolgozás egyszeres, csak egy input adattömeg van, mely mágnesszalagon érkezik. Az output adatok mágneslemezen lesznek. A végrehajtandó utasitásokból az 5. pont értelmében az első kettő alkotja az első ciklust, a harmadik utasitás a második ciklust. A program-beirás után a program beolvassa az input-adattömeg első rekordját. Ezen végrehajtja az első ciklus utasitásait és az eredményt kiirja egy külső tárolóra. Ezután a második rekordon hajtja végre az első ciklus utasitásait, majd a többi rekordon. Csak akkor kerül sor a második ciklus RPLUSZ utasitására. Ezen utasitás számára az input az előzőleg külső tárolón keletkezett adattömeg lesz. A kiválasztás és összegzés után az input adatoktól függően egy vagy több eredményrekord keletkezik. Az outputot a program a követelményeknek megfelelően most mágneslemezre irja fel.

SZÖVEGES INFORMÁCIÓK TÁROLÁSA ÉS VISSZAKERESÉSE A TRAMPS PROGRAMRENDSZER HASZNÁLATÁVAL

Komoróczy György
DATORG RT

A programrendszer kidolgozása az un. Külkereskedelmi Tájékoztatási Adatbank kiszolgálása céljából történt. A rendszer és a hozzátartozó gépi programok kidolgozásában a szerző mellett részt vettek:

Bennó Pál
Csóka Károly
Czakó Erzsébet

1. A TRAMPS szöveges információ visszakereső és kezelő programrendszer általános jellemzői

A programrendszer általános jellege folytán különböző információ forrásokat használó tájékoztatási rendszerekben alkalmazható. /Pl. könyvek, folyóirat-,ujság cikkek, müszaki dokumentáció, szabadalmi leirások, jogszabályok, hangfelvételek, elektronikus számitógépi programleirások, stb./Az információ források alapegységeit dokumentumoknak nevezzük. Tárolásukról, visszakeresésükről a tájékoztatási rendszer gondoskodik.

A dokumentumok alapján az un. feltáró /indexelő/ munkafázis során célinformációk készülnek, melyek tartalmazzák a

- dokumentum azonositó adatait
- tartalmi jellemzést és a visszakeresést biztositó mennyiségi és minőségi jellemzőket /descriptorokat/
- a tartalomra utaló, leiró jellegü szövegeket /nem kötelezően/.

A feltárás /és visszakeresés/során használt leképzési rendszer

/a célinformációk kialakitásának szabályai/ kialakitása a tájékoztatási rendszer követelményeitől függően különböző lehet. Egyaránt lehetőség van un. osztályozási rendszerek /pl. ETO/ vagy descriptor /indexelő/ nyelvek használatára.

A TRAMPS szempontjából a descriptorok maximálisan 52 jelből álló jelsorozatok, melyek általános alakja az alábbi:

aspektus:descriptor

Az "aspektus" jelsorozat kifejezi az osztályozási ismérvet, használata nem kötelező. Amennyiben használjuk, összetett descriptorról beszélünk; ha elmarad, szabad descriptorról van szó.

Pl: VILÁGGAZDASÁG /mint közgazdasági fogalom/
SAJTÓ:VILÁGGAZDASÁG /mint egy sajtótermék cime/
MEGJELENÉS: 1970
HOSSZUSÁG: 1970

A descriptorok között - a feltárás és visszakeresés összhangjának megteremtése és a visszakeresés hatékonyságának növelése érdekében az alábbi kapcsolatok megteremtésére van lehetőség:

- szinonim kapcsolatok az azonos értékü descriptorok összekötésére /pl. óceán, tenger/
- hierarchikus kapcsolatok az alá és fölérendeltségi viszony jelzésére /pl. Magyarország, Európa ill. KGST országok/
- rokon vagy asszociativ kapcsolatok /pl. fagyasztás, hideg/

A TRAMPS rendszer által tárolt descriptorok-, valamint a descriptorok közötti kapcsolatok együttesen alkotják az un. thesaurust. A thesaurus mind a feltárás, mind pedig a visszakeresés során rendkivüli fontossággal bir, megfelelő szinvonalu kialakitása a tájékoztatási rendszerek egyik alapkérdése.

A visszakeresés un. kereső kérdésekkel történik. A kereső kérdés az adott problémát leiró descriptorok logikai müveletekkel és határjelekkel összekapcsolt együttese.
A használható logikai müveletek:

ÉS
VAGY
ÉS NEM

A visszakeresés során a szinonim descriptorok kezelése automatikusan történik. Lehetőség van a hierarchikus kapcsolatok automatikus hasznositására is, az un. "MINDEN" funkció használatával.

/Pl. a "MINDEN" EURÓPA descriptor az EURÓPA - mint generikus - és az európai országok - mint specifikus - descriptorok együttes reprezentációja./

Az alkalmazott tárolási struktura biztositja a visszakeresési idő függetlenségét a tárolt célinformációk számától. A gyors visszakeresés-, a descriptorok közötti kapcsolatok gépi realizálásának biztositása érdekében un. komplex tárolási struktura került kialakitásra, mely a tárolási alapstrukturák mindegyikét magába foglalja.

A programrendszer a DATORG RT meglévő konfigurációjához alkalmazkodik. A programok nagy része csoportos /batch/ üzemmódban működik lyukkártya vagy mágnesszalag inputtal és sornyomtató vagy mágnesszalag outputtal. Az információk visszakeresése vagy párbeszédes üzemmódban, konzolirógépen keresztül, vagy csoportos üzemmódban történhet. A programok FORTRAN és assembler nyelven iródtak, a moduláris felépités lehetőséget ad az egyedi felhasználási igényekhez való gyors alkalmazkodásra.
A TRAMPS használatához szükséges minimális konfiguráció:
- SIEMENS 4004/45 központi egység,
  64 K byte-tal/

- gépkezelői konzol irógép
- 1 lyukkártya olvasó egység
- 1 sornyomtató egység
- minimálisan 2 mágneslemez egység
- minimálisan 2 mágnesszalag egység.

2. A programrendszer elemei:

- thesaurus kezelő programok
  /thesaurus építése, különféle listák készitése/
- célinformáció-kezelő programok
  /bevitel, ellenőrzés, listázás, selejtezés/
- visszakereső programok
  /lekérdezés interaktiv nyelv használatával, kártyás visszakeresés/
- adatadminisztrátor segédprogramjai
  /alkalmazási rendszerek definiálása, statisztikai adatok kiirása, átszervezés, stb./

3. A rendszer kidolgozásának megszervezése

Az általános számitógépes gyakorlatnak megfelelően a rendszer kidolgozására és üzembeállitására rendelkezésre álló idő rendkivül rövid volt. A rendszer első változatának üzembeállitására mintegy fél év állt rendelkezésre. Ennek megfelelően az átfogó /általános/ rendszerterv elkészitése után azonnal megkezdődött az alapvető - a FORTRAN nyelvet kiegészitő, karakterkezelő assembler - programmodulok irása. A részletes rendszerterv és a programok készitése, ellenőrzése a továbbiakban párhuzamosan történt.

BELKERESKEDELMI ADATBÁZIS KEZELŐ RENDSZER

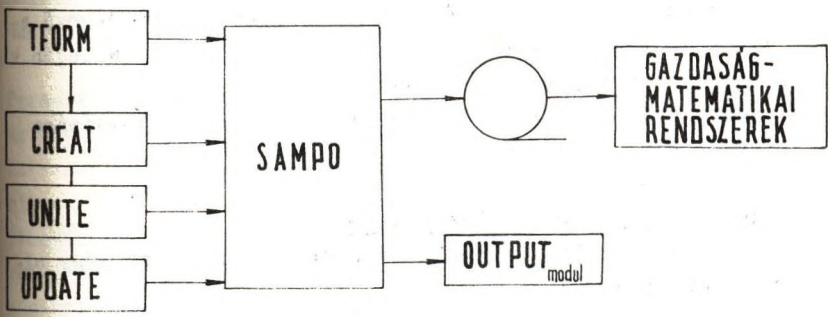
Garai Péter
KERINFORG

A gazdasági mikro- és makrostruktura szervezeteinek korszerű irányitásában egyre nagyobb szerepet kap a rendelkezésre álló hatalmas adattömegek kezelése és elemzése. A számitógép célszerű felhasználásával ez lehetővé válik, s a vezetők bármely időpontban, bármely információhoz hozzáférhetnek további döntéseik elkészitéséhez.

Ezzel el is jutottunk napjaink divatos témájához az adatbázis-kezelő rendszerekhez, melynek szükségszerüségét felismerve már hazánkban is szinte mindegyik országos hatáskörü szerv létre kivánja hozni saját adatbankját.

A Belkereskedelmi Minisztérium főosztályai is nagymennyiségü adattal rendelkeznek.
Ez az adatbázis a kis-, és nagykereskedelmi vállalatok kötelező statisztikai jelentésein /áruforgalmi jelentések, költségjelentések, stb./ és a mérlegbeszámolókon alapszik. Az adatok fokozatos számitógépre vitele nemcsak a manuális munka alól mentesit, hanem gyorsabbá és sokoldalubbá teszi a vezetés számára szükséges információk biztositását.

Az elkövetkezendő feladatokra történő felkészülés, valamint az adatbank koncepció megvalósitásának jegyében a KERINFORG Rendszertechnikai osztályán figyelembe véve a jelenlegi hardware és software adottságot és korlátokat elkészitettünk egy mágneslemez-orientált adatkezelő rendszert, mely szorosan illeszkedik az általunk fejlesztett gazdaságmatematikai programrendszerekhez is.



Ez a rendszer alkalmas a jelenlegi belkereskedelmi beszámolási rendszer alapján szolgáltatott adatok tárolására, kezelésére és feldolgozására, figyelembe veszi a népgazdasági szinten később kialakitandó országos adatbank-hálózat koncepcióit, valamint a KSH és a PM adatbázisával való kompatibilitást.

Megteremtettük annak a lehetőségét is, hogy az Intézetünknél gépi adatfeldolgozást végeztető nagykereskedelmi vállalatok /RAVILL, PIÉRT, BUTORÉRT, TRIÁL/ adatait is tárolhassuk és a rendelkezésünkre álló gazdaság-matematikai programokkal a tárolt adatok belső összefüggéseit elemezhessük.
Az adatbázis-kezelő rendszer igénye először a Minisztérium statisztikai feldolgozásai során merült fel. Köztudott, hogy általában a külföldi cégek adatkezelő-rendszerei, igy a Honeywell által ajánlott rendszer /DATA BASE SUBSYSTEM/ sem alkalmazható közvetlenül /elsősorban a hardware-konfiguráció igénye miatt/, ezért először egy mágnesszalag-orientált rendszert dolgoztunk ki. E rendszer szekvenciális

lévén igen lassu volt és vezérlőnyelve is eléggé bonyolu
nak bizonyult.

A további fejlesztés előtt tanulmányoztuk a különböző a-
datbázis-kezelő rendszereket s a tapasztalatok alapján ki
dolgoztuk a fejlesztési tervet.
Ennek során a következő lényeges problémák megoldására
került sor:

Egységes input-modul kialakitása
Az adatbevitellel kapcsolatos alábbi funkciókat négy al-
program látja el:

- tetszőleges formátumu adatok beolvasása
- adatellenőrzés
- vállalati szintü adatok egyesitése
- báziskreálás
- adatok karbantartása

File-kezelési technika
Az adatbázis-kezelő rendszert FORTRAN nyelven programoztu
Ez részben megkötötte a lehetőségeket, más részről viszon
többféle, rugalmas megoldásokat tett lehetővé /pl. az in-
put és output modul esetében/.
A Honeywell-FORTRAN lemezen tárolt adatokhoz kétféle hoz-
záférést biztosit:

- szekvenciális
- random

Természetesen szükségünk volt az indexelt szekvenciális é
a láncolási technika alkalmazására is, ezeket azonban pro
ramozni kellett.

Vezérlőnyelv továbbfejlesztése
A rendszer kialakitásához olyan információs nyelvet is ki
kellett dolgozni, amely lehetővé teszi a visszakeresésen k
vül akár egyes adatok, akár az adatok bizonyos csoportjai

között elvégzendő aritmetikai és/vagy logikai müveleteket. Intézetünknél kialakitott "report language" kezdetben számozott utasitásokból állt, melyet a FORTRAN utasitásokhoz hasonló szimbolikus nyelvvé fejlesztettünk.

Paraméterezhető output-modul

Az adatkezelő rendszer által szolgáltatott adatok táblázatszerü kiiratását paraméterezhető output-modul biztositja.
Lehetőség van tábla-generálásra, amikor a program a kiirandó adatokkal automatikusan állapitja meg a mezőszélességeket, biztositja az adatok szimmetrikus elhelyezését, vagy kivülről beolvasott formátumvezérlő kártyákkal definiálhatók a kivánt output táblázatok.
A táblázatok adatai között illetve után tetszés szerinti megjegyzés helyezhető el.

Gazdaság-matematikai programokkal való kapcsolat biztositása

Az adatkezelő rendszer által szolgáltatott adatok mágnesszalagon vagy mágneslemezen is kérhetők s igy közvetlen inputként használhatók a konferencián ismertetésre kerülő gazdaság-matematikai rendszerekhez.

TAPASZTALATOK AZ ICL PLUTO PROGRAMRENDSZERÉNEK HAZAI VÁLLALATNÁL TÖRTÉNŐ ALKALMAZÁSÁRÓL

Füle Károly
NIM IGÜSZI Számológközpont
Software-fejlesztési Osztály

1. Bevezetés

A hazai vállalatok közül is egyre többen ismerik fel a termelésirányitás komplex gépesitésének előnyeit. A NIM IGÜSZI Számológközpontjában is készül olyan számológépes rendszer, amely távlatilag integrált vállalati termelésirányitási rendszer alapját is képezheti. A megrendelés alapján egy gépipari vállalat részére készülő rendszer az ICL 1900-as gépek PLUTO elnevezésü programcsomagjára támaszkodik.

Beszámolónkban először vázoljuk a PLUTO információtároló és visszaszolgáltatási rendszerét, majd összegezzük a programcsomag alkalmazásának eddigi tapasztalatait.

2. A PLUTO-rendszer vázlatos ismertetése

A PLUTO-rendszer olyan gépi adatbázis létrehozására, karbantartására, alapszintü feldolgozására és átszervezésére alkalmas, amelyet az adatok nagy tömegén kivül az adatcsoportok közötti logikai kapcsolatok hatalmas mennyisége és sokfélesége is jellemez.

2.1 PLUTO-alapfogalmak

A PLUTO-rendszerü adatbázisban kétféle adatcsoport /rekord/ szerepel: elem /unit/ és kapcsolat /structure/ jellegü adatcsoport.

- Az elem adatcsoportját - részben vagy egészben - tetszőleges számu másik elem viszonylatban fel lehet használni. Az

elemnek azonositója - saját neve - van. A - rendszerint rövid - név teszi lehetővé, hogy az elem - rendszerint hosszu - adatcsoportjára több viszonylatban is csak utalás történjék, az adatok ismételt tárolása helyett. Az elemek elemtömböket /master file/ képeznek.

- A kapcsolat adatcsoportját mindenkor csak a két elem viszonylatában lehet felhasználni, amelyekkel a kapcsolatot létrehozzuk. A kapcsolatnak saját neve nincsen, önállóan nem használható fel. A kapcsolat két elem olyan egymáshoz rendelése, amely viszonynak saját - egyik elemhez sem tartozó - adatai is lehetnek. A kapcsolatokat kapcsolattömbökben /structure file/ tároljuk.

A PLUTO-rendszer szerkezetnek tekinti a kapcsolatok olyan sorozatát, amelyek egyetlen elemhez tartoznak. Minden egyes kapcsolat egyszerre kétféle szerkezetben vesz részt:

- az elsőként megadott - fölérendelt - elem /parent unit/ összetételi szerkezetében /component structure/; valamint

- a másodikként megadott - alárendelt - elem /subunit/ beépülési szerkezetében /used-on structure/.

Az 1. ábra A1, A2, A3 elemei gyártmányok, B1, B2, B3, B4 és B5 elemei anyagok. Közöttük összesen 10 kapcsolat áll fenn. Tegyük fel, hogy a kapcsolatokban rendre a gyártmányokat adjuk meg az első helyen. Valamely gyártmány összetételi szerkezete ilyenkor azt fejezi ki, hogy a gyártmány milyen alapanyagokból készül. Egy-egy alapanyag beépülési szerkezete megadja, hogy az anyag milyen gyártmányoknál kerül felhasználásra. A kapcsolat megadásakor a gyártmányok rendre alárendelt elemként is szerepelhettek volna. Ebben az esetben csupán az "összetételi", illetve "beépülési" szerkezet önkényes elnevezése hatna zavarólag.

A logikai kapcsolat elkülönítése a két elem adatcsoportjától több szempontból is előnyös.

- Tetszőleges, és elemenként különböző hosszu szerkezet képezhető, fölösleges helyfoglalás nélkül. A szerkezet hosszát ugyanis az elemtömbre nézve sem egységesiteni, sem maximálni nem kell.

- A kapcsolatnak - a logikai viszonyon túlmenően - több saját adata is lehet. Mindig szerepel az alapmennyiség és a sorrendkulcs, előfordulhatnak feltételi kódok, és más egyéb adatok is. Az 1. ábrán bemutatott kapcsolatok alapmennyisége például az az anyagmennyiség, amelyet az anyagból egyetlen gyártmányhoz fel kell használni. A kapcsolat adatszerkezetének további részletezésére itt nem vállalkozhatunk.

Két elemtömb a kapcsolattömbbel együtt egy adatmodult képez. Az adatmodul sematikus jelölése a 2. ábrán látható. Az adatmodul az adatbázis egy menetben feldolgozható része. Ilyen feldolgozásra példa lehet az 1. ábra alapján az anyagszükséglet számitása a - mondjuk 5 db A1 és 2 db A3 gyártmányból álló - gyártmányprogramból kiindulva.

2.2 Integrált adatbázis

Az adatmodulok összekapcsolásával integrált adatbázis jön létre. Az integráció lehetőségeit a 3. ábrán keresztül mutatjuk be, amely egyben az általunk kidolgozott vállalati adatbázis sematikus vázlata is. Ez az adatbázis 4 elemtömbből és 4 kapcsolattömbből áll. Az alkatrészek elemtömbje az anyagjegyzékek kapcsolattömbjén keresztül a szerelési és alapanyagok elemtömbjéhez ugyanugy kapcsolódik, mint a 2. ábra egyszerü adatmodulja. Két esetben a kapcsolattömbhöz nem csatlakozik alárendelt elemtömb. A szerelési, illetve a megmunkálási müveletek kapcsolati adatcsoportjai külön alárendelt elemekre: munkahelyekre is hivatkozhatnának.

A munkahelyeknek saját adatai itt nem lévén, nevüket egyszerüen beiktattuk a kapcsolat adatcsoportjába. A darabjegyzékek kapcsolattömbje viszont négy alárendelt elemtömbhöz is csatlakozik. A végtermékek és szerelvények összetételi szerkezete ugyanis heterogén elemekből: alkatrészekből, kereskedelmi gyártmányokból, szerelési anyagokból, sőt más szerelvényekből tevődik össze. Ha a kapcsolattömb fölé- és alá rendelt elemtömb igy egybeesik, az adatmodulban többszintü szerkezetek - gyártmánystrukturák - is tárolhatók és feldolgozhatók. Egy közbenső szerelvénynek ilyenkor egyszerre van összetételi és beépülési szerkezete is. Mindkét fajta adattömb esetén gyakorlatilag általában korlátlan a hozzá csatlakozó adattömbök száma, mindkét irányban. A négy eset közül egy kivétel: egy kapcsolattömbnek csak egyetlen fölérendelt elemtömbje lehet. Ez a megszoritás biztositja, hogy az adatmodul egy menetben feldolgozható legyen.

A fenti adatbázisra épülő információszolgáltatási rendszer adott gyártmányprogramból kiindulva - a többszintü gyártmánystrukturák lebontásán keresztül - különféle vállalati szükségleteket - alkatrészszükségletet, anyagszükségletet, szerelési és gyártási időszükségletet - határoz meg.

2.3 A PLUTO közvetlen hozzáférésü rendszere

Szellemes megoldása miatt érdemes néhány szóban érzékeltetni az adatcsoportokban tárolt cimek rendszerét. Minden uj adatcsoport a mágneslemezes adattömb végére kerül, ennek megfelelően kapja meg a cimét.

- Az elem csupán az első kapcsolat cimét tárolja, mind az összetételi, mind pedig a beépülési szerkezetére nézve. Az elemtömbökben külön cimtáblázatok röviditik meg az elem név szerinti keresését.

- A kapcsolat a két elem neve helyett csupán rövidebb cimüket

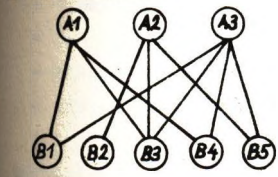
tárolja. A kapcsolatban tárolt további négy cim tovább, illetv visszafelé vezérli a kétféle szerkezet szerinti feldolgozást a szomszédos kapcsolatokra.

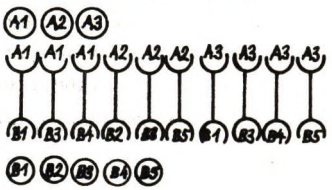
A cimeket a PLUTO-rendszer automatikusan betölti, Integrált adatbázisban az adatcsoportok cimrésze - adattömbönként egységesen - általában megsokszorozódik. Az egész adatbázis feldolgozása - a kezdeti szakasztól eltekintve - közvetlen hozzáféréssel zajlik le.

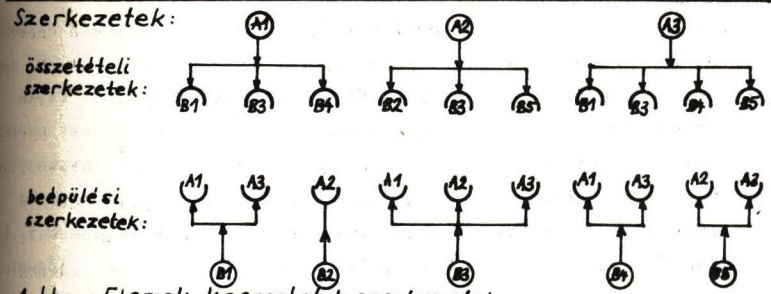
3. A PLUTO-rendszer alkalmazásának tapasztalatai

A PLUTO-rendszer nemcsak önálló programcsomag, hanem - kissé átalakitott formában - az ICL 1900-as gépek NIMMS elnevezésü integrált termelésirányitási rendszerének is része. Ez a felépitmény magyarázza a PLUTO-rendszer rendkivüli általánosságát és rugalmasságát. Egyben biztató alapul szolgál az általunk kidolgozott rendszer továbbfejlesztésére, akkor is, ha ez esetleg nem is a NIMMS magasabb szintü moduljaira fog támaszkodni. A PLUTO-rendszer önállósága tette lehetővé, hogy a termelésirányitás információs rendszerét a vezérlési feladatoktól elkülönitve, előre dolgozzuk ki. Az igy szervezett adatbázis minden bizonnyal a vezérlési feladatok magasabb szinvonalu integrációját fogja majd szolgálni.

Hazai viszonylatban igen előnyös sajátosság, hogy a PLUTO-rendszer csak háttértárolóról történő be- és kivitellel foglalkozik, valamint, hogy zömmel szubrutinokból áll. Óvatosságra int azonban az a tény, hogy a PLUTO a feldolgozás hatásfokát a felvitel rovására emelte maximális szintre. Előreláthatólag az előadás időpontjában már üzemelési adatok ismertetésére is sor kerülhet.

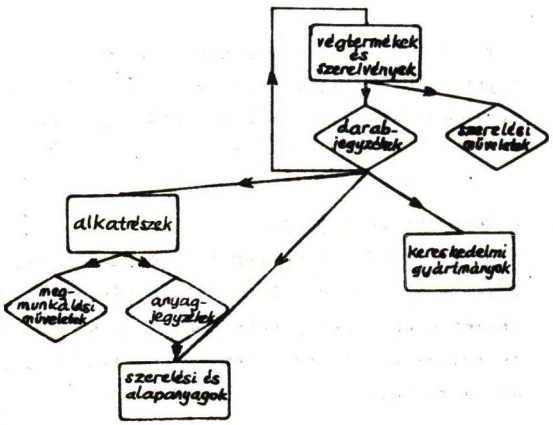






1. ábra: Elemek, kapcsolatok, szerkezetek

2. ábra: Adatmodell sematikus vázlata



3. ábra: Integrált adatbázis

MÁGNESSZALAGOK ÉS MÁGNESLEMEZEK NYILVÁNTARTÁSI RENDSZERE

Vásárhelyi Péter
NIM IGÜSZI Számolóközpont
Software-fejlesztési Osztály

1. Bevezetés

A NIM IGÜSZI Számolóközpontban az ICL 1903/A számológép üzemeltetése a GEORGE operációs rendszer alatt történik. A GEORGE operációs rendszer hatékony használata nagyfoku fegyelmet kiván többek között a géptermi mágnesszalagok és mágneslemezek naprakész nyilvántartása tekintetében is. Ebből a célból létrehoztunk külön egy mágnesszalag és külön egy mágneslemez nyilvántartó rendszert. E rendszerek azonkivül, hogy a mágnesszalagokról, illetve mágneslemezekről, azok bérlőiről, a bérlés kezdetének időpontjáról és fizikai jellemzőiről jelentést adnak, felhasználhatók a havi bérleti dijak kiszámitásán és a számlák elkészitésére is.

A továbbiakban, mivel müködési elvük azonos, csak a mágnesszalagok nyilvántartási rendszerével foglalkozunk; a mágneslemezek nyilvántartási rendszerénél előadódó eltérésekre külön kitérünk.

2. A rendszer felépitése

A nyilvántartási rendszer tulajdonképpen egy lemezen tárolt index-szekvenciális file, amelyen már kezdetben kijelöltük az összes mágnesszalag-sorszámot, a későbbi esetleges bővitéseket is beleértve. Az index-szekvenciális file kulcsa a mágnesszalagok sorszáma.

Egy mágnesszalagra vonatkozóan a következők találhatók ezen a file-n:

- sorszám,
- témakód, amelyre a mágnesszalagot kiadták,
- jel - a mágnesszalag fizikai állapotára mutat,
- kiadás dátuma,
- vezető témakód, számlázási célokra,

Mágneslemez esetén a rendszer-file a következőket tartalmazza:

- lemezsorszám,
- mutató, amely megmutatja, hogy az alábbi csoport hányszor fordul elő:

- témakód,
- kiadás dátuma,
- cylinderszám.

. A rendszer karbantartása

A rendszer karbantartásának alapkövetelménye az volt, hogy könnyen kezelhető, gyors legyen. A karbantartást paraméterkártyákkal végezzük, amelyek első pozíciójában lévő kód a paraméter funkciójára mutat. A paraméterkártya funkciója mutathat változást a mágnesszalag fizikai állapotában /pl. egyik szalag megszünt/ és a témakódra vonatkozóan /pl. adott témakódra kiadott mágnesszalagot visszaadtak/. A rendszer egyszerre csak egyfajta változást engedélyez és ellenőrzi, hogy a változtatás véghezvihető-e. Formai vagy logikai hiba esetén hibajelzést ad, hibátlannak talált paraméter esetén a megfelelő módositást végrehajtja és az uj állapotról jelentést ad.

. A rendszer kiegészitő file-jai

A rendszer működéséhez segédfile-ok szükségesek az alábbiak számára:

1. A felujitást végző paraméterek tárolása.
2. A témakódról törölt mágnesszalagok tárolása /számlázás céljából/.
3. A havi számla elkészitése.
4. Rendezések.

A fenti szükségletek céljaira soros file-okat használtunk.

5. Havi számlázás

Az adott témakódokra való kiadásnak és a hónap során a témakódokról való törléseknek megfelelően a tárgyhót követő hónap első napján elkészithető a számla, amely a vezető témakód és azon belül témakód bontásban a felhasználók számára a bérelt mágnesszalagok ill. mágneslemezek bérleti díját kiszámitja.

6. Összefoglalás

A vázolt rendszer nagymértékben kihasználja a közvetlen hozzáférésü file-ok előnyeit. Gyorsan átvezethetők rajta a rendszerben bekövetkezett változások. Használata a GEORGE operációs rendszer alatt történik.

INFORMÁCIÓKERESÉS MÁGNESSZALAGOS ADATTÁRAKBÓL, KÜLÖNÖS TEKINTETTEL A SZAKIRODALMI TÁJÉKOZTATÁSRA

Horváth Iván

MTA Központi Fizikai Kutató Intézet

1. Bevezetés

Adott információtömeg valamely speciális témáju részhalmazának kiválasztása kézi erővel gyakorlatilag kivihetetlen, ha az adattömeg nagy mennyiségü. Ez a probléma elektronikus számológép és megfelelő háttértár alkalmazásával megoldható.

Az MTA Központi Fizikai Kutató Intézetben ICL 1900-as gépekre kidolgoztunk egy programcsomagot adott profilu információknak mágnesszalagos adattárakból való kiválogatására. Ez a programok funkcióját tekintve, négy egységből áll:

a) adattárszerkesztő programok (készitő, módositó, konvertáló),
b) információkereső program,
c) a kiválogatott információ másodlagos feldolgozását végző programok (rendezés, módositás stb),
d) a feldolgozott információt kilistázó programok.

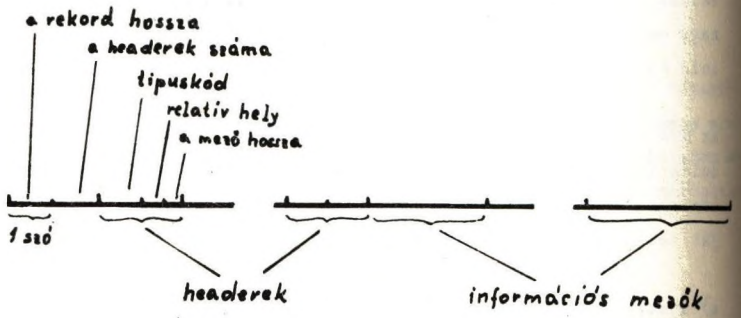
A rendszert elsősorban szakirodalmi tájékoztatásra használjuk, külföldi készitésü adattárak felhasználásával.

2. Az adattár szerkezete

Fizikailag az adattár változó hosszuságu blokkokból és minden blokk egész számu fizikai rekordból áll. Fizikai rekordon olyan fizikai információegységet értünk, amelynek az első szava az információegység hosszát tartalmazza valamilyen formában (pl. karakterekben vagy szavakban). Egy logikai egységbe tartozó fizikai rekordok egy

logikai rekordot alkotnak.

Minden fizikai rekord egy vagy több változó hosszuságu mezőből tevődik össze. Ezek a mezők tartalmi szempontból egyező vagy különböző tipusuak lehetnek, s formailag szintén (bináris, karakter stb). Ezek a mezők tartalmazzák a feldolgozandó információt. Minden mezőhöz egy-egy kétszavas, un. "header" tartozik, amely a tipuskódokon kivül a mező hosszát és a fizikai rekord kezdetéhez viszonyitott relativ helyét tartalmazza. A fizikai rekord szerkezete látható az alábbi ábrán:



Minden fizikai rekord az egyik információs mezejében a megfelelő logikai rekord azonositóját tartalmazza.

A különböző adattárakat feldolgozás előtt a fenti alakuvá konvertáljuk.

3. A profilok szerkezete

A témák profiljai, amelyek alapján a logikai rekordok kiválogatása történik állandó jelleggel a memóriában vannak egyirányu listába füzötten. Minden profil lényegében kereső szavak, szótöredékek - term-ek - halmaza, amelyekre a logikai ÉS, VAGY és NEGÁLÁS müveletét alkalmazzuk.

Ha a term-eket T-vel és a "negált" term-eket N-nel jelöljük, akkor egy profilnak megfelelő logikai kifejezés a következő alaku:

$$((NvNv...vN) \wedge (TvTv...vT)) \wedge (TvTv...vT) \wedge ... \wedge (TvTv...vT)$$

Az un. "sulyozás" lehetőséget nyujt bizonyos fokig egyes term-ek közötti AND kapcsolat szimulálására (pl. egy tényezőn belül).

Minden profil egy un. headerből és egy listából áll. A header az azonositó kódot, a logikai tényezők számát (ti. a listán belül), a lista elemeinek számát és az első elem hosszát és cimét tartalmazza. A lista egy-egy eleme egy-egy term karaktereit, a term tipuskódjait, az őt tartalmazó logikai tényező sorszámát és a következő elem hosszát és cimét tartalmazza. A profilok kezdőcimeit külön táblázat tartalmazza.

4. A keresés módja és szempontjai

Tekintettel arra, hogy a programrendszer elsősorban szakirodalmi tájékoztatás céljából készült, elsődleges szempont a gazdaságosság volt. Másrészt, mivel általában igen nagy mennyiségü információ feldolgozásáról van szó, a programnak gondoskodnia kell arról, hogy képes legyen ujra inditani önmagát, bármikor történt is a futás félbeszakitása. Ez utóbbi probléma könnyen megoldható volt, mivel a program mindössze két mágnesszalagot használ. Az első probléma végeredményben a felhasznált gépidő csökkentésével oldható meg, anélkül azonban, hogy a profilok számát csökkentenénk. Az alábbiakban ismertetünk néhány olyan megvalósitási módot, amelyekkel ezt a célt kivántuk elérni.

Az adattár és a profilok összehasonlitása szekvenciális módon történik: minden logikai rekordot összehasonlitunk egymás után minden profillal. Ebből a célból minden logikai rekord feldolgozása egy táblázat készitésével kezdődik, amelyben mezők szerint minden szó hossza és kezdőcime feljegyzésre kerül. Hasonlóképpen minden term tartalmazza első szavának hosszát. Minden összehasonlitás a szó és

a term első szava hosszának az összehasonlításával kezdődik. Ha a term hosszabb, akkor a program az összehasonlítást nem végzi el, s ha nem hosszabb, akkor is csak addig végzi az összehasonlítást, amíg a még összehasonlításra váró szótöredék nem lesz rövidebb. Másrészt a profilok szerkezete olyan, hogy az egy logikai tényezőhöz tartozó term-ek egymás után következnek - a negált term-ekkel kezdve. A program minden tényező feldolgozása után ellenőrzi, hogy az adott tényező nem "üres"-e, s ha igen, akkor a következő profil feldolgozásával folytatja a munkát. Az összehasonlítás egyként karakterenként történik.

Az adattárak, illetve tekercsek számára, a profilok méretére és számára vonatkozólag semmilyen lényeges korlátozás nincs (a számológép kapacitásán kívül).

Irodalom

[1] Standard Distribution Format Technical Specifications Revised, Chemical Abstracts Service, Columbus, Ohio, 1971. SF-1020(3-71)

[2] Data Content Specifications for CA Condensates in Standard Distribution Format Revised, Chemical Abstracts Service, Columbus, Ohio, 1971. SF-1021(3-71)

[3] ZUHK program leírása
KFKI Programkönyvtár, M9.0001/P

SZÁMOLÓGÉP SZIMULÁLÁS, MINT A SOFTWARE-FEJLESZTÉS EGYIK ESZKÖZE

Nagy Mihály
MTA Központi Fizikai Kutató Intézet

1. Bevezetés

A számológép felhasználása ma már elképzelhetetlen megfelelő software-rendszer nélkül. Minden számológép-gyártó cég azon igyekszik, hogy minél jobb és effektivebb software-eszközöket tudjon nyujtani a géppel együtt a jobb kihasználás és a versenyképesség érdekében. Ezért a software-fejlesztésnek általában meg kell előznie a tényleges gép megépitését, hogy az időre elkészüljön. Ehhez nyujthat nagyon hasznos segitséget az illető gép szimulátor programjának elkészitése egy megfelelő kiépitettségü közepes vagy nagy számológépen. A szimulált gép használata sok szempontból előnyös:

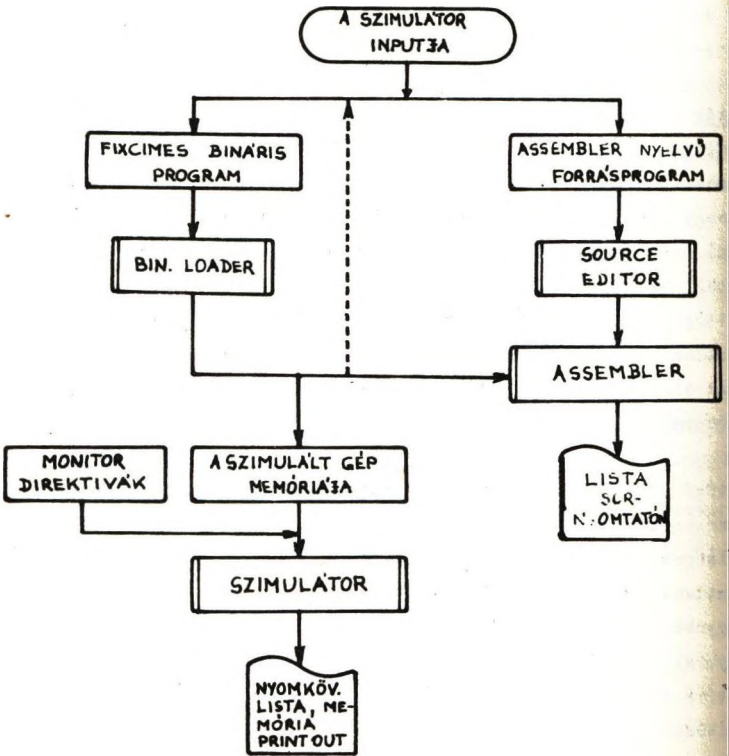
- lehetővé teszi a software fejlesztés megkezdését még a tényleges gép megépitése előtt;
- segitséget nyujt a gép megtervezésében esetleg előforduló hardware pontatlanságok felderitéséhez;
- a nagyobb számológép hardware és software előnyeit kihasználva, meggyorsitható a software kidolgozása;
- lehetővé teszi megfelelő nyomkövetési és hibakeresési rendszer kiépitését.

Az adott gépre megirt szimulátort célszerü ellátni megfelelő operációs rendszerrel, és a fejlesztéshez feltétlenül szükséges utility programokkal. Ezek lehetnek önálló programok, de lehetnek a szimulátor program szubprogramjai is.

A KFKI Számitástechnikai Osztályán üzemelő ICL 1905-ös számológépen két kisszámológép szimulátorát valósitottuk meg. Ezek a VT 1010/B és a fejlesztés alatt álló TPA-70.

2. A szimulációs rendszer felépitése

A szimulációs rendszer komponensei a következők:



1. ábra

Az 1. ábrán látható rendszer sematikusan szemlélteti a KFKI-ban software-fejlesztésre kidolgozott szimulációs rendszert. A futtatni kivánt programot vagy fixcimes bináris formában vagy forrásnyelven (az illető gép assembler nyelvén) lehet megadni. Az assembler nyelven megirt program lehet lyukkártyán, lyukszalagon vagy

mágnesszalagon. A lyukszalagon, illetve mágnesszalagon tárolt programok javitására az ICL software megfelelő maintenance programjait használhatjuk. A szimulátor programmal egybeépitett assembler lehetővé teszi a fixcimes bináris program előállitását lyukszalagon, valamint lehetővé teszi a szimulált memóriába való közvetlen betöltést is, mivel az assembler nem a szimulált gép nyelvén iródott, és igy nem foglal el külön helyet.

A betöltött program kipróbálása és futtatása a szimulátor programnak megadható direktivák segitségével történik; ezeket a következőkben monitor-direktivákнak nevezzük. A szimulátor szolgáltatásainak nagy része ilyen monitor direktivák segitségével vezérelhető, igy például a monitor direktivák beolvastatása is stb.

3. A szimulátor operációs rendszere (monitor direktivák)

A monitor direktivák használata a szimulált programok kezelését, funkcionális ellenőrzését, valamint a szimulációs rendszer egyéb tevékenységeinek vezérlését kivánja egyszerüsiteni és kényelmesebbé tenni.

A monitor-direktivákat lyukszalagon vagy lyukkártyán adhatjuk meg; amelyek beolvasásáról és értelmezéséről a szimulátor-program egyik szubrutinja gondoskodik, megfelelő operátori beavatkozás hatására.

A monitor direktivákkal az alábbi főbb szolgáltatások kérhetők a szimulátortól:

- a szimulált program elinditása adott cimről;
- megállás és postmortem kiadása adott cimeken;
- adott cimtől kezdve, adott számu utasitásról nyomkövetési információ kérése a sornyomtatón (szimulált regiszterek tartalmának kiiratása);
- a mindenkori utolsó 'n' számu végrehajtott utasitás és a hozzátartozó regiszterállapotok tárolásának igénylése (postmortem információ kiadása céljából);

- nyomkövetés felfüggesztése adott cimen;
- felfüggesztett nyomkövetés ujrainditása adott cim elérése esetén;
- adott cim elérése esetén megadott memória területek kilistázása hexadecimálisan, és/vagy postmortem kiadása;
- megállás és postmortem kiadása adott cimre való hivatkozás esetén;
- eltérő konfiguráció igénylése;
- ujabb monitor-direktivák beolvastatása adott cim elérése esetén;
- a memória adott rekeszeinek megváltoztatása;
- szimulált perifériák használati módjának megadása (buffer méret, az input vagy output módja stb);
- forditási paraméterek megadása az assemblernek;
- mágnesszalagról való forditás igénylése.

Az itt felsorolt tevékenységek némelyike ICL operátori beavatkozással is elvégezhető (inditási pontok és kapcsolók segitségével).

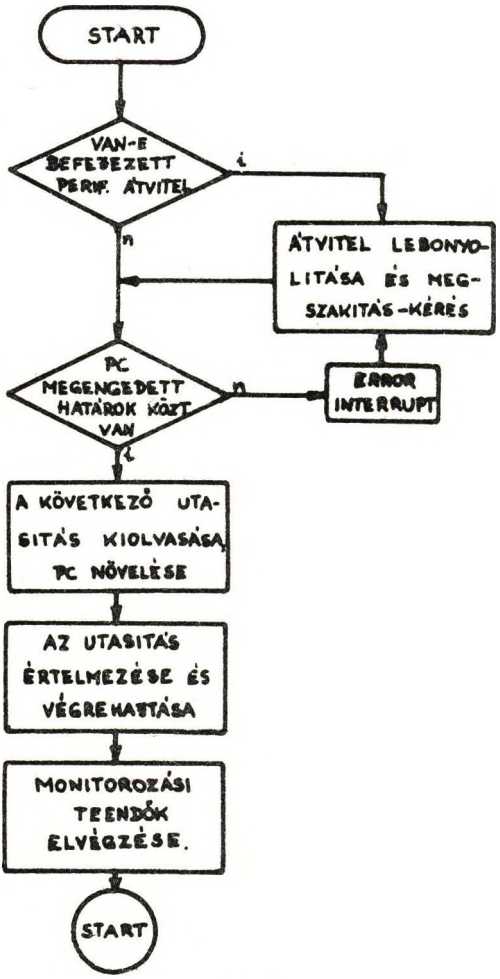
4. A szimulátor müködésének rövid ismertetése

A fent emlitett két kisgépre megirt szimulátor a következő hardware és software lehetőségeket valósitja meg:

- az alapgép utasitásainak szimulálása (külső interpreter segitségével);
- 8-16K byte kapacitásu memória;
- perifériális és interrupt rendszer szimulálása; ezen belül:

  a) időarányos szimuláció (belső óra) megvalósitása)
  b) interrupt rendszer
  c) teletype irógép
  d) gyors lyukszalagolvasó és lyukasztó
  e) FEX3 tipusu minidiszk (korlátozott kapacitással)
  f) tárvédelem
  stb

A 2. ábra vázlatosan szemlélteti a szimulátor működését.



2. ábra

UJ SZÁMOLÓGÉPEK ARCHITEKTURÁJÁNAK TERVEZÉSE, ALAP SOFTWARE-JÉNEK KIDOLGOZÁSA, SZÁMOLÓGÉPES SZIMULÁCIÓVAL

Dettrich Árpád - Csaba Margit
INFELOR

Bevezetés

Már az első elektronikus számológépek megjelenésekor kialakult az a gyakorlat, hogy a további gépek tervezésében mint hathatós segédeszközt alkalmazzák.

A számoló-tulajdonságát kihasználva elsősorban a gép elemeit tervezték meg /elektronikusan, mechanikusan/.

Másik alkalmazási terület a hardwer elemek szimulálása, a gép architekturájának megtervezésében, mig a harmadik, a software elemek elkészitéséhez ugyancsak hardware szimuláció, természetesen más meggondolásokkal. Végül nem ritkán találkozunk software elemek szimulációjával is.

Hardware tervezés

Az első lépés azoknak az áramköri elemeknek a megtervezése, amelyekből a számológép felépül. Itt tulnyomó részt logikai müveleteket kell végezni, s a kezdeti időszakban ez lépésről lépésre meg is történt.

A második generációs gépek idejére azonban kialakult az a gyakorlat, hogy nagyobb logikai egységeket tipikus funkciókkal egy elembe foglaltak össze, s igy a tervezés már csak a rendszernek ezen elemeiből való

összeállitására szoritkozott.

Lényegesen nagyobb problémát jelent azonban az ilymódon felépitett rendszer müködésének vizsgálata. A rendszer egyes elemeinek hatását, mivel diszkrét automatát szimulálunk diszkrét automatán, jól tudjuk szimulálni, és a jelenségeket ezen szimulációval nyomon követhetjük, amig determinisztikus folyamatokról van szó.

A gép egyes részei között azonban nemcsak determinisztikus kapcsolatok vannak,hanem igen gyakran láthatunk sztohasztikus folyamatokat is. Itt most nem beszélünk az élettartammal kapcsolatos problémákról, csupán olyan jelenségekről, mint pl. a perifériális készülékek és a központi egység kapcsolata, ahol a válaszidő eloszlás-függvénye viszonylag jól meghatározható, vagy olyan esetről, amikor kivülálló tényezők hatását (pl. a Real-Time) kell megfigyelni.

Az ilyen szimulációnál az egyes logikai elemek, blokkok közötti interface-t kell elsősorban megtervezni. Miután ismerjük a blokkok I/O jeleit, definiálni kell a megfelelő transzformációs függvényeket. Ezen függvény nemcsak a jelek közti kapcsolatot határozza meg, hanem a transzformáció végrehajtásnak idejéről is ad tájékoztat. Ez az időadat általában jól meghatározott determinisztikus folyamatoknál, mig sztohasztikus folyamatoknál statisztikai analizissel meghatározott eloszlási függvényt kell alkalmazni.

Ilyen módon állithatunk fel különböző konfigurációkat és próbálhatjuk azokat különböző elképzelt szituációban. Rendszerint ezen kisérlet outputja egy igen sokrétü statisztikai analizis.

A software-tervezés

Igen gyakran felmerülő probléma, hogy egy számológép megtervezésénél a hardware struktura rögzitése után azonnal hozzá kell kezdeni az alapsoftware kidolgozásához. Igy az első gépek piacra kerülésekor már egy jól müködő rendszert tudnak eladni a kezdeti hibáktól eltekintve.

A fenti célra ki kell választani egy olyan gépet, (Alapgép), amelyen a kérdéses Tárgygép hardware-jét szimuláljuk olyan szinten, mint ahogyan azt a programozó látja. Itt figyelembe kell venni azt is, hogy a T gép egy-egy utasitásának szimulálása az A gép több utasitásával történik, ezért a tervezésben lehetőleg jól ki kell használni az A gép és a T gép tulajdonságai közötti hasonlóságokat. A szimulátor tervezésénél elsősorban az információábrázolás módját kell rögziteni.
A T gép legkisebb megcimezhető információegysége az M-ben többféleképpen ábrázolható, asszerint, hogy milyen az információ-egységek hosszának viszonya.

Az ábrázolást azonban az is befolyásolhatja, hogy milyen egyszerü cim-kiválasztási lehetőséget ismerünk. Ennél az eljárásnál figyelembe kell venni még a második gép számábrázolást és utasitásábrázolási formáit.

A következő probléma a T gép regisztereinek ábrázolása. A gyakorlat azt mutatja, hogy nemcsak azokat a regisztereket kell szimulálni, amelyekhez a programozó utasitással hozzáférhet.
A másik észrevételünk pedig, hogy az A gép megfelelő regisztereit /ha ilyenek vannak/ nem célszerü a szimulálás céljaira felhasználni, azok a szimulátorban mint eszközök szerepelnek.
A következő probléma a cim-kiszámitás megoldása. Itt alsósorban a megfelelő módositó kódok segitségével kell leágazást szervezni az egyes cim-kiszámitási módokhoz. A második probléma, hogy a kapott cimeket ellenőrizni kell, hogy bele esnek-e a szimulált tárintervallumba.

Ezután az utasitás-végrehajtás ciklusát kell megszervezni, amely két fő részből áll; az egyik az utasitás kódja alapján szétágazik megfelelő tevékenységekre, és a másik ezen tevékenységek, mint szubrutinok elkészitése.
Most az egyes utasitások kidolgozásának problémáiról nem beszélünk, itt tartjuk magunkat ahhoz az elvhez, hogy az A gép utasitáskészletét minél jobban fel tudjuk használni. Külön szót kell emelni az I/O utasitások szimulációjáról, mert abban van a legnagyobb különbség a két rendszer között. Háromféle módszert alkalmazunk. Az első a megszakitási metodika szimulálása olymódon, hogy fix válaszidőt állitunk be, a második módszer, hogy a válaszidő számlálására egy, a végrehajtott utasitásokkal időarányos értékeket használunk. A harmadik módszer pedig, hogy a periférikus készülékek válaszidejére valamilyen eloszlás-

függvényt alkalmazunk. A szimulátoron készülő programok tulnyomó része azonban nem kivánja közvetlenül a fizikai I/O tevékenységet, ezért valamilyen követett uton a logikai I/O-t szimuláljuk.

A szimulátor az utasitás végrehajtásakor a rész-tevékenységről pontos információt tartalmaz, amelyeket a felhasználó kivánságára kilistázhatunk. Itt többféle un. nyomkövetési módszer ismeretes. Szokás csak a vezérlő utasitások helyeit jelezni, de szokás az is, hogy az utasitásról és annak eredményéről teljes információt adnak, beleértve az egyes regiszterek állapotát is.

Befejezésül megemlitjük néhány szóban azt, hogy a szimulátoron programot futtatni, próbálni igen nehéz volna akkor, ha önmagában állna, és nem vennénk körül megfelelő kiszolgáló rendszerrel. Ez a kiszolgáló rendszer tulajdonképpen már software szimulációt végez, mert az A gép eszközeivel biztositja mindazokat a lehetőségeket, amelyeket majd a T gép az üzembeállitás pillanatában a programkészitő, programkipróbáló rendelkezésére ad. A tapasztalataink leszürésével keressük azt az általánositási lehetőséget, amellyel egy olyan rendszer dolgozható ki, amelynek nyelvén bármely gép szimulátora viszonylag könnyen megirható.

A KALMÁR-FÉLE FIKTIV ELEKTRONIKUS SZÁMITÓGÉP SZIMULÁTORA MINSZK-2[3] GÉPEN

Simon Endre

MTA Matematikai Logikai és Automataelméleti Tanszéki Kutató Csoportja

1. A Kalmár-féle fiktiv elektronikus számitógépet /későbbiekben fiktiv gép/ Kalmár László akadémikus definiálta oktatási célokra. A programtervező matematikus szakos hallgatók gépi kódu programozási előadásain nem lett volna célszerü egy konkrét gépkonfigurációt választani és mint etalonra, erre a gépre irni fel a különböző oktató programokat. Ezek ugyanis akarva-akaratlanul kihasználják az illető géptipus speciális adottságait.

Konkrét gép esetén problémát okozott volna az a mai napig egyértelmüen meg nem válaszolható kérdés, hogy az etalonnak kiválasztandó gép, egy- két- vagy háromcimes legyen.

A fiktiv gépet tehát ugy kellett definiálni, hogy annak legyen egy- két- sőt háromcimes változata is.

Igy egy hallgató, aki a fiktiv gépen tanult programozni, konkrét gép mellé kerülne viszonylag gyorsan képes az elsajátitott programozástechnikai módszerek alkalmazására a gyakorlatban is.

Nyilvánvaló hátránya a fiktiv gépen való programozásnak, hogy nincs lehetőség a felirt programoknak a való kipróbálására, futtatására. Azon tul, hogy ez a tény megfosztja a hallgatókat a didaktikai szempontból semmiképpen nem lebecsülendő sikerélménytől, lehetetlenné teszi a vissza-

csatolást.

Mindezek a problémák adták az ötletet, hogy készitsük el a fiktiv gép összes változatának szimulátorát a JATE Kibernetikai Laboratóriumában üzemelő MINSZK-22 elektronikus számitógépre, és adjunk lehetőséget a hallgatók által irt programok ezen, szimulált gépen történő futtatására.

2. A fiktiv gép utasitásrendszere

Megállapodás szerint a gép egyes regisztereit a következőképpen jelöljük:

E: eredményregiszter /gyüjtős gépnél mindig a gyüjtőt jelenti/
R: szorzó/osztó regiszter
T: tulcsordulás regiszter
U: utasitásszámláló regiszter
I_k: k-adik indexregiszter

A fiktiv gépnek két fő tipusa definiált:

2.1 egycimes indexregiszter nélküli gép

ekkor egy gépi utasitás általános alakja a következő:

⊖ a ahol

⊖ a müvelet jele /mindig egy betü/

a a cimrész

például:

A a : (E) + (a) ⟹ (E)
B a : (a) ⟹ (E)

2.2 indexregiszteres gép

ennek létezik egy-, két- és háromcimes változata, az uta-

sitások általános formája ennek megfelelően:

ν⊖ a, i
ν⊖ a, i b, j
ν⊖ a, i b, j c, k

ahol

ν a változat jele /szám, esetleg kiegészitő jelekkel/
⊖ a müvelet jele /egy vagy két betü/

a, b, c — címek
a, ab, abc — címrészek
i, j, k — indexregiszter hivatkozások

ν⊖ együtt a gépi utasitás müveleti kódja

például:

1A a, i : (E) + (a + (I_i)) ⟹ (E)
2AT a b : (a) + (b) ⟹ (E) , (a)
1OF a b c : a ⟹ (U) ha (b) ≥ (c)
1BX a, i : a ⟹ (I_i)

A 2.2-ben leirt gép mindhárom változata esetén érvényes a következő:

- az indexregiszterek cimrész hosszuságuak
- cimrész után az indexhivatkozás opcionális
- ha a hivatkozás elmarad, ez azt jelenti, hogy a 0 cimü indexregiszter van az illető cimrészhez hozzárendelve

- a 0 cimü indexregiszterben nulla van és a beleirás tiltott

2.3 Utasitás formátumok

A fiktiv gép gépi kódu programjait előre adott formátumu kód ürlapra irjuk. Kétcimes gép esetén a helyes utasitás formátumokat a példa mutatja:

Cimke	Cim	Müv. kód	I.cimrész tart.	I.cimrész cim	II.cimrész tart.	II.cimrész cim	Megjegyzés
BELÉP:	100)	2AT	PAR1	200	PAR2	201	(PAR1) + (PAR2) ⇒ (PAR1)
	+1)	2BX	EGY	1,10		0,2	1 ⇒ (I_1) 0 ⇒ (I_2)
C1:	+2)	2FX	C2:	103,10			→ C2: ha (I_{10}) > 0
C2:	103)	1SP		0			STOP 0 ⇒ (E)
	+4)	1SP		104,0			STOP utasitás ⇒ (E)
PAR1:	200)	117					konstans1
PAR2:	+1)	77					konstans2

A cimke, megjegyzés és tartalom mezőkitöltése opcionális.

3. A szimulációs rendszer

Szimulációs rendszeren értjük a továbbiakban a szimulátort a hozzákapcsolt segédprogramokkal együtt. Mivel a szimulátor müködésekor feltételezi, hogy a futtatandó program a fiktiv gép memóriarekeszeiben már elhelyezésre került, szükség van egy beolvasó programra.

A rendszert aktiváló hallgatókról nem tételezzük fel a MINSZK-22 ismeretét, tehát része egy vezérlőprogram, mely a külvilággal egy ASR-33 tipusu irógépen keresztül tartja a kapcsolatot.

A programpróbák megkönnyitését szolgálja a rendszer részét képző nyomozó program.

A szimulációs rendszer igy a következő modulokból épül fel:

- beolvasó program
- szimulátor /ennek része a nyomozó ill. cimkövető program/
- vezérlőprogram
- az aritmetikai egységet szimuláló szubrutinok

3.1 A beolvasó program

Lyukszalagról vagy irógépről beolvassa a programot, szintaktikus és szemantikus ellenőrzést végez; az utasitásokat előre adott formában tárolja a fiktiv gép adott cimü "rekeszében".
Megjegyezzük, hogy az egységes kezelés miatt az egy-, két- ill. háromcimes gépek esetén is ugyanazt az utasitás reprezentációs formát alkalmaztuk /az utasitás müveleti kódja egyértelmüen meghatározza a hozzátartozó cimek számát/.

A fiktiv gép egy memóriarekeszét a MINSZK-22 két egymás utáni szavára képeztük le. Az 1K-s fiktiv memória igy 3777_8 szót foglal el.
A beolvasó program egy programsor karaktereit egy rekeszenként 4 byte-os pufferbe gyüjti. Minden szintaktikus kategória felismerésekor információt rak le egy stack következő szabad helyére. Ez az információ tartalmazza a felismert kategória sorszámát, valamint első puffer-

beli karakterének cimét /rekeszcim-bytecim/. Sor vége hatására a stack elemeit generálja és a sorszám ismeretében elvégzi a hozzárendelt tevékenységet /például: cimkéhez rendelt sorszám esetén veszi a stack következő elemét/.

Ha a stack minden elemét generálta - a kapott utasitást vagy konstanst elhelyezte a megadott fiktiv rekeszbe - eldönti, hogy van-e listázási kérelem. Ha nem olvassa a következő programsort, ha igen a stack elemeit ujból generálva a szélesnyomtatón előre meghatározott formában reprodukálja a beolvasott információt, majd rátér az olvasásra. A program végét egy speciális /END/ direktiva jelöli.

3.2 Szimulátor

A szimulátor megtervezésénél egy számitógép mikro-szintjén lejátszódó folyamatokat programoztuk.
A cimkövetés ill. nyomozás - mely a szimulációs rendszer opcionális szolgáltatása - tulajdonképpen nem más mint a fiktiv gép bizonyos regisztertartalmainak outputra adása.
A vezérlőprogram a MINSZK-22-höz kapcsolt irógépről, megszakitásos üzemben képes parancsszavakat átvenni. A jelsorozatok felismerésére a Dömölky-féle algoritmus egy általunk módositott változatát használtuk.

4. Befejezés

A szimulációs rendszer életrekelése után a matematikus hallgatók gépi gyakorlatát Kalmár professzor kérésére már e rendszerre támaszkodva vezették.
Két éves üzemeltetési tapasztalataink alapján a rendszert hasznos oktatási segédeszköznek mondhatjuk.

HAJLÉKONY PROGRAMCSOMAG SZIMULÁTOR ÉPITÉSÉHEZ A CDC 3300 GÉPRE

Hoffmann Péter - Mandler György
MTA AKI INFELOR

BEVEZETÉS

A gyakorlatban gyakran szükség van számitógépes konfigurációk más számitógépen történő szimulációjára. Egy szimulátor kifejlesztése viszonylag sok munkát és főleg időt emészt fel.

Munkánk során felmerült az az igény, hogy több számitógép konfiguráció szimulátorát készitsük el. Ez a követelmény tette szükségessé, hogy megvizsgáljuk, melyek a szimulált géptől független részei egy ilyen rendszernek. Ugyanis, ha ezt a szimulált géptől független közös részt a szimulátor programon belül jól elkülönitjük, és a csatlakozó felületeket megfelelően specifikáljuk, akkor igen hajlékony eszközhöz jutunk, amely uj szimulátor program elkészitését nagymértékben megkönnyiti.

A cikk a programcsomag szolgáltatásaira koncentrálva az uj szimulátor készitője szemszögéből ismerteti a rendszer felépitését.

A rendszer felépitése, szolgáltatásai

A vezérlő program modulja

A vezérlő program az egész rendszer irányitását végzi. A felhasználó vezérkártyák segitségével irhatja le programja kipróbálásához az egyes lépéseket.
A szimulátor inditásakor először a vezérlő indul el. Különböző müveletek elvégzése után /input-output beállitás bináris program betöltése, javitás, stb./ egy megfe-

lelő vezérkártyával elindítható a szimulátor. Ekkor történik a szimulált gépen lévő program végrehajtása. Ha a szimulálandó utasitás HALT, ha lejárt egy előre felhuzott számláló, vagy ha valamilyen hiba detektálódik, akkor a vezérlő program lép ismét működésbe. Mielőtt a vezérkártyákat ismertetnénk megjegyezzük, hogy a vezérlő modul a vezérkártyák felismerését elvégzi, a paramétereiket feldolgozza és részben a végrehajtásukat is megteszi, részben pedig külső - az aktuális szimulátor irója által irandó - szubrutinokat hiv le pontosan specifikált paraméter szerkezettel. Az általunk megadott vezérlő utasitások halmaza természetesen bővithető a vezérlő modul felismerő és ugró táblájának kiegészitése révén.
A vezérlő utasitásokat tipusuk szerint hat csoportba oszthatjuk.
/Az utasitások pontos értelmezésétől, és a vezérkártyán szereplő paraméterek megadásának ismertetésétől eltekintünk, csak az utasitások nevét soroljuk fel./

A szimulált gép kezelőpultjának működtetésére szolgáló utasitások

CLEAR	- regiszterek törlése
BOOTSTRAP	- rendszer betöltő
READ	- olvasás a szimulált tárba
WRITE	- a szimulált tár kiiratása
RUN	- a szimulátor elinditása

Bináris formátumu programok betöltésére és dumpolására szolgáló utasitások

LOADER
DUMP

Input-output beállitás, adathalmaz mozgatás

INPUT

OUTPUT

TRANSFER

CLOSE

A szimuláció alatti nyomkövetés és a szimuláció megszakitására való utasitások

DEBUG	- nyomozás utasitásra
DEBRW	- nyomozás irás-olvasásra
INTERRUPT	- megszakitás kérés

Konzol üzenet kiiratása

TYPE

Visszatérés a MASTER-be

FINIS

Input-output modul

Rutinjai a perifériás müveletek perifériától független, logikai szinten való kezelését teszik lehetővé.

Négy fontosabb rutin tartozik a modulba:

IDEVICE	- input eszköz beállitás
INCHAR	- egy karakter szolgáltatása a beállitott input eszközről
ODEVICE	- output eszköz beállitás
OUCHAR	- egy karakter kiadása a beállitott output eszközre.

A szimulátor felhasználója az INPUT és OUTPUT vezérkártyán beállithatja az aktuális input és output eszközöket.

Az input eszközök lehetnek:
lyukszalag olvasó,
kártyaolvasó BCD-ben /automatikus BCD-ISO konverzió/,
kártyaolvasó binárisan,
diszk file,
mágnesszalag file

Az output eszközök lehetnek:
lyukszalag lyukasztó,
kártya lyukasztó BCD-ben /automatikus ISO-BCD konverzió/
kártya lyukasztó binárisan,
diszk file,
mágnesszalag file,
sornyomtató.

A BOOTSTRAP, LOADER, TRANSFER, valamint a szimulált gép lyukszalagos bemenete a beállitott input eszközről veszi az inputját.
A DUMP, TRANSFER és a szimulált gép lyukszalagos outputja a beállitott output eszközre adja az outputot.

Megszakitás és nyomkövetés modul
Egy szubrutint tartalmaz, mely a szimulált utasitás végrehajtása után kerül behivásra. A szubrutin megvizsgálja, hogy egy korábbi DEBUG, DEBRW vagy INTERRUPT vezérkártyán adott feltétel teljesül-e. Ha igen, akkor behiv egy rutint, melyet az aktuális szimulátor készitője irt, vagy, interrupt teljesülése esetén, visszatér a vezérlőbe.

Tapasztalataink szerint a szimulált kisgép input-output és megszakitási rendszerének az elkészitése az egész munka igen jelentős részét teszi ki.

Sok munkát takarithatunk meg, ha ezen részek szimulációja helyett a gép operációs rendszerének input-output és megszakitásokkal kapcsolatos makróit szimuláljuk. Ebben az esetben a szimulátor működését is meggyorsitjuk. Természetesen ez csak akkor valósitható meg, ha az elképzelés a szimulálandó számitógép operációs rendszerének ezen részeiről már kikristályosodott.

Tapasztalatok és megjegyzések a programcsomag használatával kapcsolatban

Nagy segitséget nyujt, hogy a programcsomag a szimulátor rendszertervét tartalmazza, tehát az aktuális szimulátor készitőjének előre pontosan specifikált szubrutinokat kell csak megvalósitania. Pl: szubrutin, mely egy byteot olvas ki a szimulált tárból, stb.
A vezérlő modul szolgáltatásai egy viszonylag teljes rendszert alkotnak. A programcsomag bővitése révén nyert szimulátorból nem maradnak ki fontos szolgáltatások tervezési hiba folytán.
Lehetőség van a lyukszalagos müveletek kiküszöbölésére, vagy minimálisra csökkentésére. Ez ugyan nem elvi követelmény, de a gyakorlatban igen lényeges szempont.
A programcsomag használatával uj szimulátor készitésénél az összes munka kb. 50%-át megtakarithatjuk az átfutási időt pedig harmad részre csökkenthetjük.

CANDYS I. SZIMULÁCIÓS PROGRAMRENDSZER

Jávor András - Görög Péter
Központi Fizikai Kutató Intézet

Bevezetés

Bonyolult rendszerek kialakitásánál a megvalósitás magas költségeire való tekintettel nagy jelentőséggel bir a bemérés időszakában, illetve a telepitést megelőző információgyüjtés a megvalósitandó rendszer várható üzemi paramétereiről.

Ezen rendszerek egy részénél a feladatok zárt matematikai alakban történő megoldása igen nehézkes, sőt sok esetben nem is lehetséges |1|, |2|. Ez általában két okra, illetve azok kombinációjára vezethető vissza.

- A rendszer rendkivül bonyolult és nehezen kezelhető algoritmussal irható le.
- Egyes paraméterek csak sztochasztikusan adhatók meg. Az eloszlás függvények analitikus formában gyakran nem állithatók elő, csak empirikus formában kerülhetnek meghatározásra.

Ezen kategóriába tartozik a számitógéprendszerek, valamint a távadatfeldolgozó hálózatok információforgalmi problémáinak jelentős része is.

Ilyen kérdések vizsgálatára került kidolgozásra az MTA Központi Fizikai Kutató Intézet Elektronikus Főosztályán a Monte Carlo módszereken alapuló CANDYS I. /Computer and Network Dynamic Simulator/ programrendszer.

Programrendszer kialakitásának elvi szempontjai

A szimulációs rendszer kialakitása során döntő jelentőségü

volt, hogy az idő- és munkaráforditás, illetve az általa elért eredmény viszonya rentábilis legyen.

A feladat jellege peremfeltételként szabta meg változó algoritmusu elemek, különböző topológiai elrendezések és eloszlásfüggvények előfordulását.

Ennek alapján döntöttünk a különböző információforgalmi rendszerek vizsgálatánál univerzálisan felhasználható szimulációs programcsomag mellett. Az esetenként irt programok u.i.nemcsak rosszabb ráfoditás/eredmény arányt adnak, hanem a gyorsan változó problémák miatt a munka esetenként való ujrakezdésével a szimulációs vizsgálat elvégzése időben nem is lenne biztositható.

Fontos kérdés volt az alkalmazott programnyelv megválasztása. Három választási lehetőségünk volt:

- Assembler tipusu nyelv
- Magasszintü általános programozási nyelv
- Szimulációs célnyelv

Az első lehetőséget elvetettük, u.i. a programok elkészitése rendkivül munkaigényes, a programok csak az adott assembler forditóprogramjával rendelkező gépeknél lettek volna felhasználhatók. Ezen hátrányok nem álltak arányban azzal az előnnyel, amit a futás során a helyben és időben való megtakaritás nyujt.

A szimulációs célnyelvek /GPSS, CSL, SIMSCRIPT, stb/ alkalmazásának lehetőségét megvizsgálva arra a következtetésre jutottunk, hogy ezek egy-egy speciális feladat megoldására igen jól alkalmazhatók, de több olyan megkötést tartalmaznak /pl. időinkrementálás/, amelyek miatt az adott feladatkörben flexibilisen változtatható programrendszer kialakitásánál gyakran az optimálistól igen távoli megoldásra vezethetnek. Hátrányként emlitendő, hogy ezen nyelvek forditóprogramjával csak bizonyos tipusu

számitógépek rendelkeznek, és igy megnőtt volna - az első megoldáshoz hasonlóan - a számitógép-típushoz való kötöttség.

Ezen szempontok alapján döntöttünk a második lehetőség mellett és választottuk a FORTRAN nyelvet, melynek foditóprogramjával szinte valamennyi számitógép rendelkezik. Figyelemre méltó, hogy az általános szimulációs nyelvek közül pl. a CSL is közbenső compilerrel müködik, azaz első menetben FORTRAN-ra fordítódik.

A programrendszer szerkezeti felépitése

A CANDYS I. segitségével információs hálózatok üzemi paramétereinek vizsgálata végezhető el, /átviteli hatásfok, puffermemória kihasználás, késleltetési-, várakozási- és válaszidő. stb./

A forgalmi viszonyok jellemezhetők egyrészt az információforrások által kibocsátott üzenetcsomagok hosszának, illetve ismétlődésének eloszlásfüggvényeivel, másrészt - adatátviteli vonalaknál - a hibaeloszlások és a hibavédelmi algoritmus /CCITT decision feedback/ segitségével.
A hálózat elemei a kivánt elrendezés szerint modulárisan épithetők be a programba.

Az elemek, a topológia, az információforgalmi paraméterek /üzenethosszuság, ismétlődési idő, stb./, valamint a kezdeti értékek és a szimulálandó idő megadása után egy keretprogram felépiti a felhasználó által kivánt konfigurációt.

A futás után a program táblázatos formában közli az eredményeket. Opcióként a program rajzgépen szolgáltatja a hisztogramokat, lineáris vagy logaritmikus léptékben.

A rendszer bemenő paraméterei és a vele kapcsolatos kérdések adatszalag segitségével egyszerü módon megoldhatók.

Az eddigiek során többszintü adatátviteli hálózatok forgalmi viszonyait |3|, |4| és időosztásos rendszerek puffermező kihasználási algoritmusait vizsgáltuk különböző peremfeltételek mellett.

A programrendszer rövid specifikációja a függelékben található.

Végezetül megemlitjük, hogy a CANDYS I. moduláris felépitése lehetővé teszi az elemválaszték egyszerü bővitését ujabb szubrutinok irásával, és ez a munka jelenleg is folyik.

FÜGGELÉK

A jelenlegi kiépitésben az alábbi elemekből képezhetők rendszerek:

Terminál tipusu elemek

- SINK — információ adó
- SOURCE — információ vevő
- TERMINAL — Terminal funkcióit modelezi /program bevitel, párbeszéd, random bejelentkezés, adott eloszlásnak megfelelő üzenetek, stb./

Információ átviteli csatorna

- ARQ — hibavédett adatátviteli csatorna modelje

Csomóponti elemek
/különböző vezérlési algoritmussal és tárolóval/

- NODE 1 — adatátviteli hálózathoz
- NODE 2 — időosztásos rendszerhez

A hálózat max. 99 elemet tartalmazhat és az elemek funkcióinak értelemszerü figyelembevételével tetszőleges topológia alakitható ki.

Bemenő paraméterek

- TS — szimulálandó idő
- N — az elem topológiai sorszáma
- P — az elemhez tartozó topológiai csomópontok
- H — adatátviteli vonal blokkhiba valószinüsége
- S — az adatátviteli vonal jelölt információt visz-e át?
- TMAX — tároló mérete

- TACT tároló kezdeti tartalma
- NT terminálok száma
- MLD terminálok által generált üzenetek hosszának eloszlásfüggvényei
- RT üzenetek gyakorisága
- M NODE 2-ban lévő puffermezők száma. M≤NT egy puffermező kapacitása
- PRI terminálok prioritáskezelő algoritmusa

Irodalomjegyzék

|1| Tocher, K.D.: The Art of Simulation /D. Van Nostrand/ 1963.p.120

|2| J.H. Mize, J.G. Cox: Essentials of Simulation Prentice Hall Inc. Englewood, Cliffs N.J.1968

|3| A.Csákány-A.Jávor: Investigation of Throughput Efficiency of Multilevel Data Networks by Means of Simulation XIX. Rassegna Internazionale Elettronica, Roma 27-30, Marzo 1972. pp. 251-258

|4| A.Csákány-A.Jávor: Some Achievements in the Investigation of Data Transmission Networks using a Flexible Simulation System XVIII. Rassegna Internazionale Elettronica, Roma Marzo 1971. pp. 331-337

SZIMULÁCIÓS NYELVEK ALKALMAZÁSI LEHETŐSÉGEI DIGITÁLIS RENDSZEREK AUTOMATIZÁLT TERVEZÉSÉBEN

BOHUS MIKLÓS

Budapesti Müszaki Egyetem Hiradástechnikai Elektronika Intézet

1. Bevezetés

Digitális számitógépek, adatfeldolgozó rendszerek strukturális és algoritmikus szimulációjára számos szimulációs nyelv használható. A Budapesti Müszaki Egyetem Hiradástechnikai Elektronika Intézete a Számitástechnikai Koordinációs Intézet megbizásából szovjet-magyar kétoldalu együttmüködés keretében foglalkozik az OSzSz-2 hivatkozási nyelv transzlátorának elkészitésével. A nyelv felépitésének és főbb jellemzőinek ismertetése - más nyelvekkel együtt - [3]-ban megtalálható. A munka előzményeként 1970-ben elkészült a "SUBSET szimulációs nyelv" transzlátora és leirása [5], [2], mely nyelv az OSzSz-2 nyelvvel alapkoncepcióban megegyezik, de csak közbülső helyet foglal el az alacsony és magas szintü szimulációs nyelvek között. Alkalmazási kérdésekkel több publikációban foglalkoztunk [1], [2], [4].

Ebben a dolgozatban - a korlátozott terjedelem lehetőségein belül - [1] - [6]. felhasználásával digitális rendszerek automatizált tervezési lehetőségeivel foglalkozunk.

2. Digitális rendszerek leirása

A leirás során deklarálni kell a rendszer felépitését /automatákat, ezek állapotait, funkcionális egységeket, időzitést biztositó hálózatokat, stb./; az operandusok között megvalósitandó müveleteket, valamint a vezérlések /mikroprogramozott, vagy huzalozott/ strukturáját.

Korlátozások:

a/ Az operandusok dimenziójának vagy rendjének megváltoztatására szolgáló operátorok alkalmazása a logikai struktura meghatározása során kerülendő.

b/ Az architektura leirásában redundáns részek nem engedhetők meg. A részletes logikai terv kialakitása során egy szubrutin megvizsgálja az ismétléseket eredményező hálózatokat, a többször előforduló változók kiküszöbölésével egyszerüsiti a leirást.

c/ A tervezői programrendszer bemenete formális nyelvü leirás, kimenetét az automatákat meghatározó Boole egyenletek képezik.

d/ Előzetesen definiálandók a tárolók tipusától függő vezérlési utasitások /ez szimuláció során nem szükséges/, valamint makro funkcióként az összetett logikai operációk /dekódolók, számláncok, aritmetikai elemek, stb/. A leirás során a makro funkciókat több mikroprogram használhatja.

3. Tervezői program rendszer felépitése

A formális leirásból egymáshoz kapcsolódó lépésekkel alakitható ki a leirást realizáló logikai terv.

a/ A vektorokból /regiszterek, sinrendszerek, stb./ és mátrixokból álló kifejezésekhez egy olyan struktura rendelhető, amely az adatáramlást figyelembe véve önálló /részleteiben később megvalósitandó/ részekre bontja a formális leirást. Ennek során közbülső változók keletkeznek, ezeket az eredeti leirás nem tartalmazta. A formális nyelvü leirás ilymódon olyan - jellegében változatlan - részekre esik szét, melyek definiált méretü önálló be-, ill. kimenettel rendelkeznek.

b/ Az operandusok /vektor, mátrix/ a leirás során indexelt mennyiségek. Az index konstans vagy változó lehet. Változó index esetén az index helyettesitendő egy szelekciót végző logikai blokkal, mely az index változó felhasználásával egy dekódolót vezérel és ez egy szelekciós vektort állit elő. A szelekciós vektor bitjeihez rendelt kapuk vezérlik az adatáramlást. A dekódolót makro operátor tartalmazza.

c/ A makro funkcióhoz /a be-, ill. kimenetek méreteinek ismeretében/ hozzárendelhetők az előzőleg definiált logikai hálózatok. A makro generátorok felhasználásával a formális leirás egy része már megvalósult. Az algoritmusok többi része itt már mikroesemények diszkrét sorozatára van felbontva.

d/ A mikroprogram végrehajtásához szükséges vezérlési rendszer kidolgozása során figyelembe kell venni a feltétel nélküli és feltételes vezérlésátadásokat, párhuzamos műveletvégzéseket.

e/ A logikai tervet alkotó tárolók egy részét a formális leirás deklarálja. Keletkezhetnek tárolók a közbenső változókból is, ha azok értéküket a vezérlés után is megtartják. A felhasználandó tároló tipusától függően meghatározhatók a vezérlési egyenletek. Ilymódon egy-egy utasitás a tároló méretétől és vezérlendő bemeneteinek számától függő kombinációs hálózatot eredményez.

f/ Az automaták vezérlése az állapotregiszter tartalma alapján történik. Az állapotregiszter dekódolásával nyert

vezérlőjelek egy-egy mikroesemény aktivizálására használhatók. Az állapotregiszter megfelelő vezérlésével hajtható végre a mikroprogram segitségével történhet vezérlésátadás, kapuzással várakozás.

g/ Az eddig előállitott logikai egyenletek azonos bemenetek mellett azonos müveletekkel különböző változókat hozhatnak létre. Ezt megvizsgálva a logikai hálózat egyszerüsithető.
Azonos tárolók különböző jelforrásból érkező vezérlése esetén a tárolók vezérlése összevonható.

h/ Az eddig tömbökként kezelt változók elemekre történő szétbontásával minden komplemenshez hozzárendelhető egy-egy önálló logikai hálózat. Ezzel a részletes logikai hálózatot sikerült előállitani.

A tervezés során a változók konkrét realizálása megtörtént, a konstansok feldolgozásra kerültek. A konstansokhoz nem tartozik önálló hardware, ezek az utasitások kezelését befolyásolják.

A leirt tervezési eljárás alapján egy konkrét logikai hálózat lépésről-lépésre történő kialakitását az előadáson mutatjuk be.

Irodalom

[1] Bohus M.: Szimbólikus nyelvek felhasználása digitális rendszerek funkcionális és parametrikus szimulációjára. Hiradástechnika. XXIII. No.6. /1972./

[2] Bohus M., Németh G., Trón T., Varró L.: SUBSET szimulációs nyelv digitális rendszerek funkcionális vizsgálatára. Hiradástechnika XIII. No.6. /1972/.

[3] Drasny J., Bohus M., Diósi I., Flesch I., stb.: Elektronikus számitógépek automatikus tervezése. 16-803/5-Et.sz. OMFB tanulmány. 1971.

[4] Bohus M.: Szimbólikus nyelvek alkalmazása számitógépek funkcionális és strukturális szimulációjára. "Számitógép' 71 Konferencia" Esztergom. 1971. II. kötet.

[5] Bohus M., Németh G., Trón T., Varró L.,: OSzSz-2 SUBSET. kisérleti szimulációs nyelv. Számitástechnikai Koordinációs Intézet. 1970.

[6] Strukturális algoritmusok és kapcsolási ábrák leirására szolgáló nyelv: OSzSz-2 /szabványtervezet/. Elektronikus Számitástechnikai Tudományos Kutató Központ. Moszkva 1970.

[7] K.E.Iverson: A Programming Language, John Wiley and Sons, New-York, 1962.

[8] J.R.Dumey - D.L.Dietmeyer: A digital system design language /DDL/ IEEE Trans. Computers vol C-17, pp. 850--861, September 1968.

[9] F.R.A.Hopgood: Compiling techniques. Macdonald. London. 1970.

DIGITÁLIS BERENDEZÉSEK SZÁMITÓGÉPES SZIMULÁCIÓJÁNAK ÉS ON-LINE BEMÉRÉSÉNEK EGY MÓDSZERE

Arató Péter, Kalmár Péter, Kondorosi Károly, Lantos Béla

Budapesti Müszaki Egyetem, Folyamatszabályozási Tanszék

1. Bevezetés

Digitális készülékek fejlesztése, gyártása során a bemérés, ellenőrzés az egyik leghosszadalmasabb és legfáradságosabb feladat, mely ritkán oldható meg célberendezések alkalmazása nélkül. Az összépitett készülék, vagy részegységei az ellenőrzés szempontjából bizonyos számu bemenettel és kimenettel rendelkező sorrendi vagy kombinációs logikai hálózatnak tekinthetők.

Ezen hálózat bemeneteire kerülő jelkombinációt a továbbiakban vezérlésnek, a kimenetein megjelenő jelkombinációt pedig válasznak nevezzük.

Az ellenőrzést két módon hajthatjuk végre:

a/ Előidézünk minden lehetséges vezérlés sorozatot, melyet a környezet /pl. a csatlakozó berendezések/ előállithat, azaz szimuláljuk a környezetet. Megvizsgáljuk a válaszok alapján, hogy ezek a sorozatok az előirt müködést eredményezik-e.

b/ Az ellenőrzendő berendezés felépitésének ismeretében meghatározunk egy olyan vezérlés-sorozatot, mely minden szerkezeti egységet ellenőriz anélkül, hogy valamennyi lehetséges sorozatot előállitanánk. Ezzel az ellenőrzés gyorsitható, de a felépités ismerete szükséges.

Az ellenőrizhetőség feltétele, hogy minden vezérlés hatására a rendelkezésre álló eszközökkel megfigyelhető változások történjenek, vagy ha gyors aszinkron folyamatok játszódnak le, akkor ezek lefutásának helyessége utólag megállapitható legyen.

Egy speciális periférikus készüléket és megfelelő programrendszert alkalmazva az ellenőrzés számitógép segitségével is elvégezhető.

A következőkben bemutatunk egy olyan szimulációs ellenőrző rendszert, mely lehetővé teszi a VIDEOTON 1010B számitógép alkalmazását tetszőleges digitális készülék környezetének /működtető berendezésének/ szimulációjára és a készülék bemérésére.

2. A szimulációs ellenőrző rendszer

A rendszer két részből áll:

- egy speciális periférikus készülékből, /továbbiakban szimulátor/ mely lehetővé teszi a vezérlés programozott beállitását és a válaszok programozott ellenőrzését,
- egy programrendszerből, mely a vezérlések beállitását, a válaszok ellenőrzését és a hibák kijelzését végzi.

2.1. A szimulátor

A számitógép programozott csatornájára, a prioritási láncban tetszőleges helyre kapcsolható. Jelenlegi kiépitésében 64 kimenete van, melyek a vezérlés beállitására, és 64 bemenete, melyek a válaszok érzékelésére szolgálnak. Mind a kimenetek, mind a bemene-

tek 8-as csoportonként programmal választhatók ki. Az információátvitel egy-egy ki- vagy beviteli utasitás hatására a kiválasztott vezetékcsoport és egy memóriarekesz között történik.

2.2. A programrendszer

A programrendszer kialakitásának fő szempontja az volt, hogy az ellenőrzés folyamatábrájának ismeretében az ellenőrző program könnyen, áttekinthetően, a számitógép részletes ismerete nélkül elkészithető legyen, és legyen meg a lehetőség a működtető berendezés bizonyos funkcióinak szimulációjára is.

Ez a cél egy szimbolikus nyelv kidolgozásával valósitható meg a legteljesebben, azonban az ASTROL assemblerben meglévő szubrutin-láncolási lehetőség, az u.n. "ASTROL interpretativ" nyelv használata, e nélkül is lehetővé tette a probléma csaknem ilyen szintü megoldását.

A programrendszer két részből épül fel:

a/ Az ellenőrzési folyamatábrákon előforduló müveleteknek megfelelő szubrutinrendszer. Ezen szubrutinok az ellenőrzendő berendezéstől függetlenek.

b/ A szubrutinrendszer elemeiből felépitett szimulációs ellenőrző program, mely az ellenőrzés folyamatábrájának alapján könnyen, áttekinthetően felépithető.

A szubrutinrendszer a következő tipusu müveleteket tartalmazza:

a/ Vezérlő és válaszjelek kezelése.

Például: vezérlőjel beállitás, elágazás válaszjel értéke szerint, rendszer ellenőrzés, 8 bites csoport beállitása adott kombinációra, stb.

b/ Belső feltételek kezelése, és belső müveletek.
Például: belső tároló beállitása, elágazás belső tároló értékétől függően, inkrementálás, byte összehasonlitás, stb.

c/ Külső beavatkozási lehetőség és a kezelő tájékoztatása.
Például: szöveg kiiratás, bitenkénti kiiratás, elágazás a vezérlőpult kapcsolóállásától függően, stb.

3. ESZR csatorna szimuláció

A szimulációs ellenőrző rendszert sikeresen alkalmazták az ESZR csatornára illeszkedő periféria-vezérlő készülékek bemérése és approbációja során.
Az ellenőrzés folyamatábrája az ESZR csatorna folyamatábrája alapján készült. A csatorna müködési módját, és az átviteli folyamat jellemző adatait a program inditáskor konzol irógépről bekéri, majd ezen paramétereknek megfelelően lebonyolitja a teljes adatátvitelt /kezdeti kiválasztástól a végállapot beadásáig/.
A bekért paraméterek a következők:

CU CIME :
PARANCS :
HALT I/O :
BYTE-SZAM :
SELECTOR MOD :
PARANCSLANCOLAT :
STATUS TAROLAS :
STATUS ELFOJTAS :

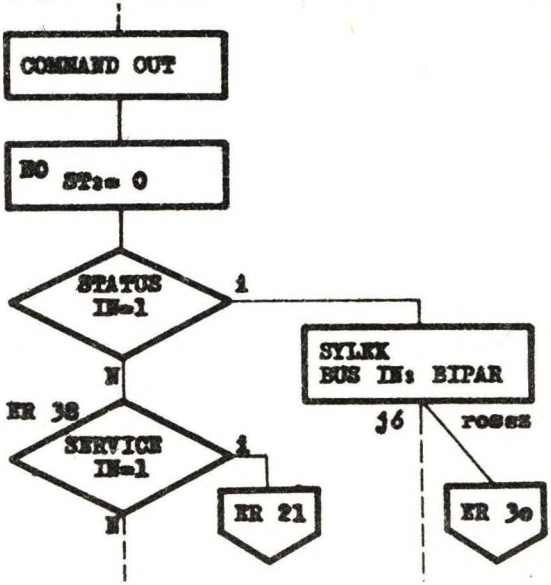
ADATELFOJTAS :

Válaszképpen a CU CIME, a PARANCS és a BYTE-SZAM kiírásoknál hexadecimális számot, a többi esetben I vagy N karaktert kell beütni.

A program a vizsgált periféria-vezérlő egységgel folytatott dialógus során annak minden válaszjelét ellenőrzi, hibás müködés esetén konzolüzenet formájában a hiba pontos megnevezésével tájékoztatja a kezelőt.

Az átvitel során valamennyi adatbyte bitenkénti és karakter formájában kiiratható. A kiiratások a kezelőpult kapcsolóinak megfelelő állásánál kiiktathatók, ekkor a hibás jelsorozatot ciklikusan ismétli a program, lehetővé téve a müszeres ellenőrzést.

Az ellenőrzés folyamatábrájának egy részlete, és az annak megfelelő programrészlet az ábrán látható.



```
        .
        .
        .
        ADS   CUTAL
        *     CCL.O
        *     KUL
        ADS   CCNI
        *     ?T
        *     FUL
        ADS   INTER
        *     STIN
        *     LGY
        *     IR 38
        ADS   SYNK
        CH2   0200
        *     BUSIN
        *     BIPAR
        *     IR 30
              .
              .
              .
IR 38   ADS   INTER
        *     STIN
        *     LGY
        *     IR 87
        .
        .
        .
```

Az ellenőrzés után a periféria-vezérlő készülék összekapcsolva egy MSZR szaritógéppel kifogástalanul működött.

4. Alkalmazási területek, a továbbfejlesztés irányai

A szimulációs ellenőrző rendszer statikus, illetve a számítógép működési sebességénél lényegesen lassabb folyamatok esetén dinamikus ellenőrzésre alkalmas. A jelenlegi programrendszer folyamatábrával leirt berendezések ellenőrzésekor használható a legkényelmesebben, ez pedig abban az esetben célszerü, ha az ellenőrzendő készülék és a működtető berendezés vezérlési kapcsolata dialógus jellegü.

Az ismertetetthez hasonló rendszert kidolgoztunk az ODRA 1013 számitógépre is. Ennél kettéválasztottuk a programrendszert a kombinációs és sorrendi hálózatokat ellenőrző programrendszerré, sorrendi hálózatokhoz kidolgoztunk egy szimbolikus nyelvet, melyet HTL /Hardware Testing Language/ nyelvnek neveztük el. Ezt az ellenőrző rendszert sikeresen alkalmaztuk a VIDEOTON 1010B számitógép programozott, ill. multiplex csatornájára illeszkedő periféria-vezérlő egységek kifejlesztése során a logikai kártyák és az összeépitett berendezések bemérésére.

A továbbfejlesztés lehetőségeit a következőkben látjuk: Az ellenőrző programrendszer része lehet egy számitógépes tervező-ellenőrző rendszernek. Elsősorban logikai kártyák beméréséhez célszerü lenne biztositani olyan programrendszert, mely az elemek és az összekapcsolás leirása után automatikusan generálná a vizsgáló vezérlés-sorozatot, és a hibás alkatrész helyéről is tájékoztatást adna.

IRODALOM:

1. ASTROL 8 manuel d' operateur. - NT 2595/FR-CII 197o.

2. 10010 computer vol. 2. programming system
NT 2242/En-CII 1968.

3. ASTROL 8 assembler manuel d' utilisation
NT 2446-1/FR-CII 1968.

4. Interface 10010 manuel d' utilisation
NT 2243-1/FR-CII 1968.

LINEÁRIS SZABÁLYOZÁSI RENDSZEREK SZÁMITÓGÉPES TERVEZÉSE

Nagy Judit, dr Somló János

MTA Automatizálási Kutató Intézet

1. Bevezetés

A modern szabályozáselmélet egyik legfontosabb iránya rendszer analizisre és szintézisre szolgáló számitógépes programcsomagok kifejlesztése. Ezek egyszerü input rendszer mellett lehetőséget teremtenek szabályozási rendszerek sokoldalu vizsgálatára. A témakörben ismeretes számos munka közül megemlitjük a P.Atkinson [1], valamint az F.Fallside [2] által javasolt programrendszereket, amelyek a rendszeranalizis klasszikus módszereit /frekvencia, gyökhelygörbe módszerek/ implementálva a rendszerről átfogó képet nyujtanak. Az optimális és az adaptiv irányitás eredményeit használja fel E.J. Davison [3] a rendszer paramétereinek megfelelő kiválasztásához. A terület további nagyszámu munkájának felsorolásától e helyen eltekintünk

Számitógépes rendszertervezéskor a tervezés hatékonysága nem csak a kiválasztott módszer hatékonyságától, hanem az ember-gép kapcsolat lehetőségeitől is függ. Munkánkban egy olyan számitógépes módszert javaslunk, amely állandó együtthatós, lineáris szabályozási rendszerek paramétereinek interaktiv módon történő kiválasztásához a klasszikus módszereket használja fel. A módszer a paraméter szintézis javasolt útja mellett sokoldalu rendszervizsgálatot tesz lehetővé a felhasználó által kiválasztott sorrendben.

2. A módszer ismertetése

Állandó együtthatós lineáris szabályozási rendszerek leirásának legáltalánosabb alakja, matrix egyenlet formájában a következő

$$D(p)\bar{x}(t) = E(p)\bar{g}(t) \qquad /1/$$

ahol $\bar{x}(t)$ az állapotváltozók n vektora, $\bar{g}(t)$ a külső beavatkozójelek m vektora, $D(p)$ n * n, $E(p)$ n * m polinomiális matrixok, és a matrixok elemei:

$$D_{ij}(p) = d^{o}_{ij} + d^{1}_{ij}\,p + \ldots + d^{\ell_{ij}}_{ij}\,p^{\ell_{ij}} \qquad \begin{matrix} i=1\ldots n \\ j=1\ldots n \end{matrix}$$

ill.
$$E_{ij}(p) = e^{o}_{ij} + e^{1}_{ij}\,p + \ldots + e^{k_{ij}}_{ij}\,p^{k_{ij}} \qquad \begin{matrix} i=1\ldots n \\ j=1\ldots m \end{matrix}$$

differenciál operátorok, amelyeknek együtthatói között ismeretlen értékü paraméterek is előfordulnak. Az (1) rendszer formálisan megoldható a Cramer szabály alkalmazásával [4]:

$$\Delta(p)\; x_i(t) = \sum_{j=1}^{m} \Delta_{ij}(p)\, g_j(t) \qquad i=1\ldots n \quad /2/$$

ahol $\Delta_{ij}(p)$ és $\Delta(p)$ differenciál operátorok, melyeket polinomiális matrixok determinánsainak kifejtésével kapunk meg. A determinánsok kifejtése elvégezhető mind szimbolikus, mind numerikus alakban [5], mindkét esetben eredményképpen a

$$\Delta_{ij}(p) = \sum_{\ell=1}^{k} K_\ell\, Q_\ell^{ij}(p) + Q_o^{ij}(p) \qquad \begin{matrix} i=1\ldots n \\ j=1\ldots m \end{matrix}$$

illetve a

$$\Delta(p) = \sum_{\ell=1}^{k} K_\ell\, Q_\ell(p) + Q_o(p) \qquad /3/$$

összeget kapjuk, ahol $K_1 \ldots K_\ell$ a rendszer paraméterei, vagy a rendszer paramétereit tartalmazó kifejezés, $Q_\ell^{ij}(p)$ ill $Q_\ell(p)$ pedig polinomok. Valamely állapotváltozó és külső jel közötti eredő átviteli függvény a következő:

$$\frac{X_i(p)}{G_j(p)} = \frac{\Delta_{ij}(p)}{\Delta(p)} = \frac{\sum_{\ell=1}^{k} K_\ell \, Q_\ell^{ij}(p) + Q_0^{ij}(p)}{\sum_{\ell=1}^{k} K_\ell \, Q_\ell(p) + Q_0(p)}$$

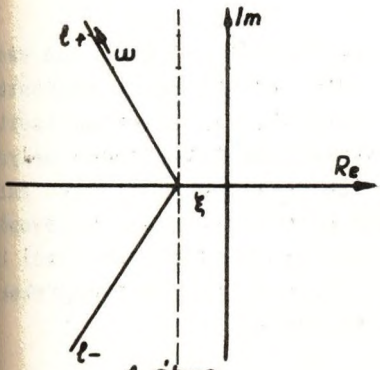
ahol $X_i(p) = L\left\{x_i(t)\right\}$; $G_j(p) = L\left\{g_j(t)\right\}$ Laplace transzformáltak.

A továbbiakban az átviteli függvény zérus-pólus konfigurációi alapján, a D-szétválasztás módszereinek felhasználásával törekszünk ugy megválasztani a paramétereket, hogy a rendszer müködése kedvező legyen. A javasolt módszer szerint a tervezés célja olyan paraméter értékek kiválasztása, amelyek mellett a rendszer viselkedését alapvetően egy kedvező elhelyezkedésü komplex-konjugált gyökpár határozza meg. /domináns gyökpár/

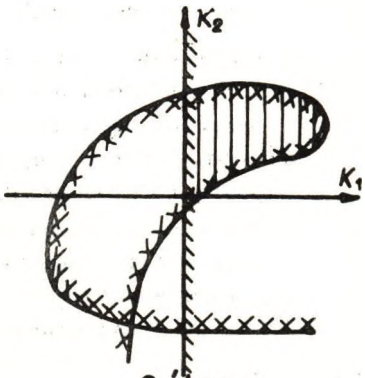
Tételezzük fel, hogy a /3/ karakterisztikus polinomban mindössze két paraméter, K_1 és K_2 szerepel. Ekkor a karakterisztikus egyenlet :

$$Q_0(p) + K_1 Q_1(p) + K_2 Q_2(p) = 0 \qquad /4/$$

alaku. A p változó helyett a $p = \xi + (\sigma + j)\omega$ $(\xi \leq 0,64)$ kifejezést a /4/ egyenletbe helyettesitve, a valós és képzetes részt szétválasztva és a kapott egyenletrendszert $K_1 = K_1(\omega)$ és $K_2 = K_2(\omega)$ -ra megoldva ω értékének 0 és ∞ közötti változtatásával a komplex sik ℓ_+, ℓ_- egyenespárját /1. ábra/ leképezzük a K_1 K_2 paramétersik $K_2 = f(K_1)$ görbéjébe /2. ábra/. A D-szétválasztás törvényei alapján $\xi = 0$ és $\sigma = 0$ értékekkel megkapjuk a stabilizálási tartomány hosszát, illetve $\xi \neq 0$ $\sigma = 0$ értékekkel az adott stabilitási tartaléku tartomány határát. Igy meg tudjuk határozni az ℓ_+ ℓ_- egyenespárok leképzésével kapott görbe tervezés szempontjából kedvező részét. Ezekből a tartományokból választva a K_1 K_2 paraméterpárokat, az átviteli függvény zérusait és pólusait meghatározzuk, majd az idő és frekvencia függvényeket kihasználva lehetőség nyilik a legmegfelelőbb paraméterpár ki-



1. ábra



2.ábra

választására. A kiválasztott K_1K_2 párhoz tartozó gyökpárt rögzitve, a karakterisztikus polinom fokszámát, és a paraméterek számát kettővel csökkentve a tervezés a többi paraméterrel hasonló módon tovább folytatható.

3. A számitógép program ismertetése

A számitógép program legfontosabb szubrutinjai :

1. Polinomiális determinánsok kifejtése
2. D-szétválasztás
3. Pólusok-zérusok kiszámitása

A fenti szubrutinok meghatároznak egy olyan paraméter tartományt, mely a tervezés szempontjából lényeges. A következő szubrutinok a kiválasztott paraméterekkel további rendszeranalizist végeznek :

4. Állandósult hiba értékelés
5. Időfüggvény meghatározás
6. Frekvencia függvény meghatározás
7. Kettőnél több paraméter esetén a paraméterek számának csökkentése és további tervezés.

Ezenkivül a további speciális vizsgálatokra van még lehetőség :

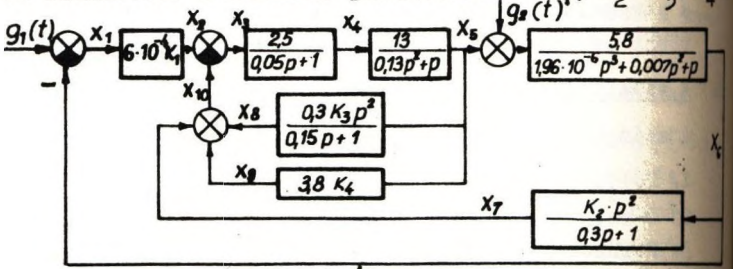
8. Belső hurok elemzés
9. Popov görbe számitás /esetleges nemlinearitások hatá-

sának elemzésére/.

Az eddigiekből kitünik, hogy a módszer hatásosan csak interaktiv módon alkalmazható. Az ember-gép párbeszéd kulcsszavak megadásával történik, igy kivánság szerint a program bármikor megszakitható, és tetszőleges helyről folytatható. Az adatok ki és bevitelét, valamint kulcsszavak szerint a feladatok elrendezését egy szervezőprogram végzi. A szervezőprogram gondoskodik arról is, hogy kivánság szerint az idő, vagy frekvencia-görbék a plotteren kirajzolásra kerüljenek.

4. Példa

Vizsgáljuk a 3. ábrán bemutatott többhurkos szabályozási rendszert. A rendszer paramétereit K_1, K_2, K_3, K_4-el



3. ábra

jelöltük. A rendszert leiró egyenletrendszer a következő

$$x_1 + x_6 = g_1(t) \qquad 2{,}5\,x_3 - (0{,}05p+1)\,x_4 = 0$$

$$6\cdot10^{-4}K_1 x_1 - x_2 = 0 \qquad 13\,x_4 - (0{,}13p^2+p)\,x_5 = 0$$

$$x_2 - x_3 - x_{10} = 0 \qquad 0{,}3\,K_3 p^2 x_5 - (0{,}15p^2+1)\,x_8 = 0$$

$$3{,}8\,K_4 x_5 - x_9 = 0 \qquad K_2 p^2 x_6 - (0{,}3p+1)\,x_7 = 0$$

$$x_7 + x_8 + x_9 - x_{10} = 0 \qquad 5{,}8\,x_5 - (1{,}96\cdot10^{-6}p^3 + 0{,}007p^2 + p)\,x_6 = -5{,}8\,g_2(t)$$

A feladat megoldása több lépésben történik.

I. A karakterisztikus polinom kiszámitása /bemenő adatként a polinomiális mátrix elemei, vagy eredeti formá-

jukban a fenti egyenletek szerepelnek/ :

INPUT MATRIX ON CARDS

OUTPUT SIGNAL

II. A D-szétválasztás elvégzése a megfelelő paraméter tartomány kiválasztásához :

FIXED PARAMETERS :

K3 = 1.0, K4 = 1.0 ,

COMPLEX LINE MAPPING SIGMA = 0.

PLOTTER (SIGMA = 0)

COMPLEX LINE MAPPING SIGMA = -1.

PLOTTER (SIGMA = -1)

III. Pólusok kiszámitása megadott paramétereknél./A paramétereket meg lehet adni az eredményül kapott lista sorszáma szerint is /

POLES FOR NUMBERS :

25, 26, 27, 35, 36, 40

IV. A $\Delta_{ij}(p)$ polinom kiszámitása zérusok meghatározására, frekvencia és időfüggvény számitás :

OUTPUT SIGNAL I = 6, J = 2

ZEROS FOR 2 PARAMETERS :

K1=3.16, K2=1.13, K1=5.02, K2=1.85

FREQUENCY RESPONSES FOR PARAMETERS :

K1=5.023, K2=1.851

PLOTTER / FREQUENCY DIAGRAM /

TIME FUNCTION

V. A karakterisztikus egyenlet egy gyökpárját rögzitve a tervezés a K_3, K_4 paraméterekkel folytatható.

REDUCED SYSTEM SIGMA=0., KSI=-0.95

FIXED ROOT PAIR: -0.315 $\pm$ 0.315 * i

PLOTTER /REDUCED SYSTEM/

VI. A redukált rendszerre is elvégezhető a fenti müveletek, hasonló módon.

5. Összefoglalás

A programcsomag a CDC-3300 gépen FORTRAN nyelven került kifejlesztésre. A jelenlegi lehetőségek mellett az interaktivitás alacsony foku, de az eszközök fejlődésével ez könnyen módositható. A dolgozatban ismertetett programcsomag felhasználó orientált és az alkalmazott módszerek elvi ismerete mellett felhasználása igen egyszerü.

Irodalom

[1] P. Atkinson: Computer Aided Design of Closed Loop Control Systems. Computer Aided Design 1972 Nº 4.

[2] F. Fallside : Interactive Graphics Technique for the Design of Single-input Feedback Systems. Proc. IEE. Vol 119 Nº 2. 1972.

[3] E.J. Davison : The Systematic Design of Control Systems. V. IFAC congress 1972.

[4] Nagy J.-Nagy Zs.: Eredő átviteli függvények számitógépes meghatározása. Mérés és Automatika 1972 Nº 4.

[5] Gantmacher : Teorija matric Nauka, Moszkva, 1966.

TRANZ - TRAN 2
NEMLINEÁRIS ÁRAMKÖRANALIZIS RENDSZER

Dr. Tarnay Kálmán
Dr. Székely Vladimir

Budapesti Müszaki Egyetem
Elektroncsövek és Félvezetők T.

Az elektronikai tervezésben egyre nagyobb tért hóditanak az univerzálisan alkalmazható, felhasználó-orientált áramköranalizis programrendszerek. A Budapesti Müszaki Egyetem Elektroncsövek és Félvezetők Tanszékén dolgoztuk ki a TRANZ - TRAN nemlineáris áramköranalizis programrendszert. Ez az alábbi feladatok elvégzésére alkalmas:

1. nemlineáris egyenáramu áramköranalizis
2. nemlineáris egyenáramu érzékenység-vizsgálat
3. nemlineáris tranziens analizis
4. lineáris kisjelü áramköranalizis
5. zajanalizis.

A 2 - 5. szegmensek az első szegmens által meghatározott egyenáramu munkapontra vonatkozóan, ill. ebből a munkapontból kiindulva végzik az analizist.

A programrendszer tartalmazza az elektronikában gyakrabban előforduló áramköri elemek (tranzisztor, logikai áramkörök, stb.) modelljeit, valamint lehetővé teszi többször előforduló áramkör-részletek olyan jellegü tárolását, mellyel ezek későbbi áramkörökhöz irt programokban rövid utasitás segitségével ismételten behivhatók. Lehetőség van az áramköri elemek értékének módositására. Az analizis során egyes áramköri elemek (elsősorban félvezető-eszközök) hőmérséklet-függése is figyelembevehető.

Előadásunkban nem tárgyaljuk részletesen a programrendszerrel kapcsolatos modellezési, hálózatelméleti és matematikai jellegü kérdéseket; ezekről korábbi közleményeinkben számoltunk be. Foglalkozni kivánunk viszont azokkal a problémákkal, melyek a programrendszer széleskörü felhasználásra alkalmas jellegéből fakadnak.

A programrendszer csomóponti analizist végez. Az analizis alapösszefüggése:

$$0 = \overline{UCS} + \overline{\overline{Z}} \cdot \overline{IG}\,(\overline{UCS}, dQ/dt)$$

ahol UCS a csomóponti feszültségek vektora

Z a csomóponti impedanciamátrix

IG az ágak forrásáram-vektora, mely az aktiv ágak forrásáramát, valamint a nemlineáris elemek áramát tartalmazza (tranziens analizis esetén a dQ/dt kapacitiv áram-összetevőket is).

A program módositott Newton-Raphson módszerrel oldja meg a nemlineáris (differenciál) egyenlet-rendszert. A csomóponti impedanciamátrix előállitását és az egyenletmegoldás elvét a függelékben adjuk meg.

A programrendszer használatához felhasználó-orientált bemeneti nyelvet fejlesztettünk ki. Ez a bemeneti nyelv alkalmas az áramkörnek az elektronikában megszokott, alkatrészlista jellegü leirására, valamint a módositások és az analizisvezérlő, dokumentáló utasitások szöveges megadására.

A programrendszer első szegmense értelmezi az áramkörleirásban megadott szöveges, ill. numerikus információkat, és több vonatkozásban diagnosztikai vizsgálatot végez. Ellenőrzi pl., hogy a bemeneti adatok megfelelnek-e a bemeneti nyelv szintaktikájának, hogy zárt áramkörről van-e szó, stb. A módositó ill. analizisvezérlő utasitások beolvasásakor szintén megtörténik azok diagnosztikai vizsgálata, részben szintaktikai szempontból, részben pedig arra vonatkozóan, hogy az adott módositás végrehajtható-e, hogy az adott helyen az analizisvezérlő utasitások alkalmazhatók-e. A bemeneti nyelv számos, különböző dokumentálási lehetőségeket biztositó utasitással is rendelkezik. Ezek beolvasásánál is megtörténnek a megfelelő diagnosztikai vizsgálatok.

A bemeneti szegmens az áramkör-leirás beolvasása és értelmezése után az áramkör nevével és egy azonositószámmal ellátva szalagra irja az áramkör jellemző adatait -- és ez a nevével azonositható adatcsoport szolgál a további analizisek alapjául. A további szegmensek az előbb emlitett adatcsoport beolvasása után

sornyomtatón kiirják az áramkör leirását, majd ezt követően beolvassák az analizisvezérlő, dokumentáló, stb. utasitásokat. Ezek ellenőrzése után indul meg a tulajdonképpeni analizis. A program ilyen felépitése biztositja azt, hogy hibás leirásu analizis-feladat ne kerülhessen futtatásra.

A felhasználó rendelkezhet arról is a programjában, hogy a futás során egyes ellenőrző adatok kinyomtatásra kerüljenek (pl. nemlineáris feladatok megoldásánál az egyes iterációk jellemző adatai). Deklarálható maximális gépidővel adtunk módot a felhasználónak a számitás időbeni korlátozására.

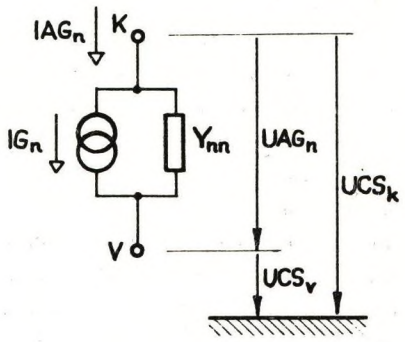
A programrendszer számos, az áramkör müködése ill. az analizis lefutása szempontjából többé-kevésbé általános adatot (pl. környezeti hőmérséklet, megengedett gépidő, iteráció-hibakorlát, stb.) automatikusan beállit szokásos értékekre — de az analizisvezérlő utasitások között alkalmazhatunk olyan utasitást is, amely ezeket az értékeket az adott analizis-feladat tartamára a programozó által kivánt értékre módositja.

A bemeneti nyelv szöveges információit a program bemeneti szegmense 8 karakterre kiterjedő, bitenkénti ekvivalencia-képzéssel értelmezi. A numerikus információknál általában megtörténik annak az ellenőrzése, hogy egész- vagy valós tipusu számadat szükséges az adott helyen. Amennyiben a szöveges információ nem értelmezhető, a programrendszer hibajelzést ad (minősitve a hibát), és a bemeneti adatok következő soránál folytatja az értelmezést. (Természetesen, egy vagy több ilyen hiba a bemeneti adatok beolvasása után is hibajelzést, megállást eredményez.) Amennyiben a következő sorra térve sem képes a bemenő adatokat értelmezni, a lyukszalag-olvasón lévő bemeneti szalagot a programrendszer szabványos végjeléig továbbitja, és rátér a következő áramkörre.

A programrendszert ALGOL nyelven irtuk, RAZDAN-3 gépi reprezentánsban, egyes helyeken a müködés gyorsaságát biztositó gépi kódos szegmensekkel. Jelenlegi formájában a teljes programrendszer több mint 40000 utasitásból áll.

Függelék

Az áramköri ág általános felépitése az alábbi ábrán látható:



A hálózat topológiáját definiálja az A incidenciamátrix, mely kapcsolatot teremt az UAG ágfeszültségek és az UCS csomóponti feszültségek között:

$$UAG_n = A_{nm}\, UCS_m$$

Az ágak árama az ág forrásáramból és az ágadmittancián átfolyó áramból tevődik össze:

$$IAG_n = UAG_k\, Y_{nk} + IG_n(UCS_m)$$

— ahol az IG forrásáramok bármelyik csomóponti feszültség nemlineáris függvényei lehetnek.

A felirt alapösszefüggések a

$$0 = UCS_q + Z_{qn}\, IG_n(UCS_m)$$

nemlineáris egyenletrendszerre vezetnek, ahol a csomóponti impedanciamátrix

$$Z_{qn} = (A_{kq}\, Y_{kj}\, A_{jm})^{-1}\, A_{nm}$$

Irodalom

1 Dr. Tarnay K. - Székely V.: Dióda és tranzisztormodellek számitógépes áramkörtervezés céljára, Mérés és Automatika, V.17, No.4, pp. 130-135 (1969)

2 Dr. Tarnay K. - Dr. Székely V.: A TRANZ-TRAN nemlineáris áramköranalizis program DC szegmensének használata (1971)

3 Dr. Tarnay K. - Dr. Székely V.: TRANZ - TRAN nemlineáris áramköranalizis program, a III. Országos Elektronikus Müszer és Méréstechnikai Konf. Kiadványa, pp. 11-18 (1972)

4 Dr. Tarnay K. - Dr. Székely V.: Áramkör - analizis programok alkalmazása az oktatásban, Szerkezeti Konstrukció az Elektronikában konferencia kiadványa, pp. 299-306 (1972)

A GEORGE 3 OPERÁCIÓS RENDSZER ÁLTAL NYUJTOTT LEHETŐSÉGEK ALKALMAZÁSA PROGRAMCSOPORTOK FEJLESZTÉSÉBEN

Bogdánfy Géza - Laufer Tamás
NIM IGÜSZI Számológközpont
Software-fejlesztési Osztály

1. Bevezetés

Az operációs rendszerek elsődleges feladata, hogy hatékony segítséget nyujtsanak a számológépeken folyó munkák megszervezésében, lebonyolításában és adminisztrálásában. Érthető, hogy a velük kapcsolatos vizsgálódások tárgyát az előbbi feladatkörökhöz való illeszkedésük képezi. Jelen beszámolónk azonban a fentiektől eltérő szempontból tekinti az ICL 1900-as sorozatu gépek GEORGE 3 operációs rendszerét. Célunk egyrészt az, hogy rámutassunk a GEORGE 3 néhány olyan értékes tulajdonságára, amelyek jól hasznosithatók a sok programból álló összetett strukturáju programrendszerek fejlesztésében, másrészt az, hogy az elmondottak megvilágitására példaként bemutassuk egy binárisan rendelkezésre álló, az ICL cég által készitett programcsomag továbbfejlesztését.

2. A GEORGE 3-ról

2.1 A job-leirás nyelv néhány jellegzetessége

A GEORGE 3 által végrehajtandó feladatokat a job-leirásban kell megfogalmazni. Ennek nyelvezete a job-leirás nyelv, melynek szintaxisa és szemantikája a magasabb szintü programozási nyelveket tekintve különösen az ALGOL 60-al mutat részleges analógiát. Felismerhetjük a blokknak, a változónak és az eljárásnak megfelelő elemeket.

Mind a job-leirások, mind az un. makrók /ez utóbbiak az ALGOL eljárások megfelelői/ önálló blokknak tekinthetők. Ezekben

automatikusan 24 érték szerint hivott formális paraméter specifikálódik. Nem kötelező azonban híváskor valamennyit aktuális paraméterrel helyettesiteni. A blokkok paraméterei, mivel érték szerint hivottak, az ALGOL-hoz hasonlóan lokális változóknak minősülnek. Ezek a változók nemcsak a hivás által kaphatnak értéket, hanem a blokk belsejéből is. Bár ezen változók rendre string tipusuak, ahol ez nem vezet ellentmondásra, ott megfelelő automatikus konverzió révén aritmetikai-, logikai-, reláció- és értékadási müveletek is elvégezhetők velük az értelmezett szövegmüveletek mellett.

A fenti értelemben vett blokkok tetszőleges mélységben egymásba skatulyázhatók. Jelentősnek mondható, hogy lehetőség van rekurziv makrók definiálására is, amely a feladatok tömörebb megfogalmazására nyujt módot. A rekurziv makrók különösen olyan esetekben lehetnek előnyösek, amikor több program változó feltételek melletti ciklikus egymásutánban való futtatásainak száma előre nem meghatározható, és a közbenső eredmények függvénye.

2.2 Extrém off-line I/O lehetőségek

A GEORGE 3 vezérlése alatt futó programok inputjait és outputjait on-line és off-line módon egyaránt képes kezelni. Mi most csupán az off-line lehetőségekkel foglalkozunk, amelyekkel kapcsolatban két jelentősebb tulajdonságra hivjuk fel a figyelmet:

a/ Konfiguráció- és eszközfüggetlenség

A GEORGE 3 alatt futó programok párhuzamosan gyakorlatilag tetszőleges számu szimulált alap- és háttérperifériát kezelhetnek, függetlenül a tényleges konfigurációtól. Igy pl. egy program számára leköthetünk 16 kártyaolvasót és ugyanennyi sornyomtatót is. Ez lehetővé teszi többek között az input adatok és eredmények megfelelő csoportositását és szétválasztását

olymódon, mintha valóban egy nagy konfigurációju gépen dolgoznánk. A háttértárolók kapacitásának felosztása, a file-ok fizikai elhelyezése a GEORGE 3 hatáskörébe tartozik. A felhasználónak nem kell feltétlenül tudnia arról, hogy adott időpontban file-jai mely fizikai eszközön és hol helyezkednek el. A file-okat csupán "nevük"-kel azonositjuk.

b/ Az alapperifériás file-ok egyenértékűsége

Bizonyos korlátozásoktól eltekintve - amely esetekben konverziót kell alkalmaznunk - az off-line I/O tárolása azonos módon történik. Ez azt eredményezi, hogy valamely program off-line outputja, esetleges automatikus rekordkonverzió mellett egy másik program off-line inputjául szolgálhat. Igy egy program TP, CP vagy akár LP outputját egy másik program CR vagy TR inputként fogadhatja. Ugyanaz az alapperifériás file egyszerre több egyidejüleg vagy egymásután futó programhoz is hozzárendelhető, akár inputra, akár outputra. Az ilymódon egymás mellett futó programok szinkronizációjáról - ha szükséges - a rendszer automatikusan gondoskodik.

2.3 Job-leirások és makrók generálása

A rendszer nemcsak az I/O kezelésében mutat nagyfoku rugalmasságot, hanem a job-leirások és makrók tárolásában és végrehajtásában is.

A job-leirások, illetve makrók tárolási módja semmiben sem különbözik az off-line I/O tárolástól. Igy alapperifériás file-okban job-leirásokat és makrókat is tárolhatunk. Tárolt utasitások a file nevével történő hivatkozással job-ként és makróként egyaránt végrehajthatók. Eltekintve az előbbi kétféle hivás formai különbségeitől, a végrehajtás abban különbözik, hogy mig a job-leirásban egy makró meghivását majd végrehajtását követően a

GEORGE a vezérlést a makrót hivó job-ba adja vissza, addig ugyanezen file-ra job-ként hivatkozva egy önállóan kezelt és a hivást kezdeményezővel párhuzamosan futó job-ot kapunk. A job-leirásokat és makrókat tároló file-oknak az alapperiférias file-okkal való egyenértékuségéből és a 2.2 b/ pontban elmondottakból következik, hogy lehetőség van arra, hogy a job-leirásokat és makrókat programokkal generáljuk. Ennek érdekében nem kell egyebet tennünk, mint egy GEORGE 3 utasitásokat előállitó program outputját offline kezelve egy file-ba irányitjuk; erre a file-ra a későbbiekben mint tárolt job-ra vagy makróra hivatkozhatunk.

A generált makrók használata lehetővé teszi egyrészt, hogy egy programrendszert futtató job-leirást az input adatok vagy közbenső részeredmények függvényévé tegyünk, másrészt, hogy összetettebb funkcióju job-okat egyszerübb logikával szervezzünk meg. E módszer különösen akkor lehet előnyös, amikor kész vagy eleve csak binárisan rendelkezésre álló programokból, illetve programcsomagokból kell adott célra egy uj programrendszert szerveznünk.

3. Tapasztalatok egy ICL által készitett programcsomag továbbfejlesztésében

Gyakori probléma, hogy a számológépekhez szállitott alkalmazási programcsomagok nem minden tekintetben elégitik ki a hazai felhasználók igényeit. Ez vonatkozhat az input vagy output formátumára, szerkezetére és a programcsomag funkcióira is. Ilyen természetü igény merült fel a NIM IGÜSZI Számológközpontjában az ICL cég villamosenergia-rendszerek üzemviszonyait számitó programcsomagjával kapcsolatban is. A feladatmegoldást egyszerüsitette, hogy felhasználtuk a GEORGE 3 2. pontban vázolt lehetőségeit-

3.1 A továbbfejlesztéssel szemben támasztott igények

A vonatkozó programcsomaggal kapcsolatban a hazai felhasználók

az alábbi többletigényeket támasztották:

a/ alkalmas legyen járulékos direktivák bevezetésével háttéren tárolt információk "könyvtárának" létrehozására és kezelésére;

b/ a programcsomag által szolgáltatott eredményeken tulmenően olyan járulékos eredményeket is kaphassunk, amelyek az input és output összevetése alapján előállithatók;

c/ a felhasználó adatelőkészitési munkájának megkönnyitése érdekében a programrendszer által egyszerre fogadható adatsorozat volumenek az eredetinek többszöröse legyen.

3.2 A megoldás néhány jellemzője

A feladat megoldásához több kiegészitő program megirására volt szükség, amelyek egyrészt a 3.1 a/ ill. b/-nek megfelelő háttérkezelési és számolási funkciókat látják el, másrészt szövegkezelési és "szervezés"-jellegü tevékenységeket végeznek.

Nyilvánvaló, hogy szükség volt olyan programra, amely az uj rendszer inputját elemzi, az abban elhelyezett vezérlő direktivákat értelmezi, valamint az eredeti programcsomag input szintaxisának megfelelő formára transzformálja. /Pl. kiegészitő direktivák kiszürése, a nagyobb méretü adatsorozat feldarabolása stb./

Az inputot elemző program egy másik lényeges funkciója, hogy az általa nyert információk alapján az egész programrendszert vezérli olymódon, hogy a 2.3-ban mondottakat kihasználva makrót generál. Ezen makró olyan állandó makrók hivásait tartalmazza, amelyek ujabb makrógenerátor futtatásán keresztül, az input elemző által nyert információkat tovább feldolgozva, az eredeti progra-

csomag és a kiegészitő programok futtatásait végző makrókat generálnak. Igy egy olyan többszintü vezérléshez jutottunk, amely összetett módon függ az input szerkezetétől és volumenétől.

A továbbfejlesztett programrendszert a NIM IGÜSZI Számolóközpont ICL 1903 A gépén kisérletileg üzemelő GEORGE 3 Mark 5 alatt próbáltuk ki. A futtatások során szerzett tapasztalatok kedvezőnek mondhatók.

GEOLÓGIAI KUTATÓFURÁSOK ADATAINAK GÉPI FELDOLGOZÁSA

Virágh Károly-Révész Bendegúz-Eszterhás Sándor
MÉV NIM IGÜSZI NIM IGÜSZI

Hazánkban a földtani kutatásokra előirányzott anyagi eszközök 75-85 %-át kutatófurásokra forditják.

Egy közepes mélységü kutatófurás dokumentációja több ezer földtani megfigyelést tartalmazhat és az ásványi nyersanyag mennyiségi és minőségi felmérésében egy mintának tekinthető.

Az ismertetendő rendszer feladata a kutatófurások adatainak gépi tárolása és visszakeresése, valamint az ércesedés lokalizációs törvényszerüségeinek, mint többváltozós valószinüségi függvényeknek a vizsgálata, a matematikai statisztika módszereivel.

1. A földtani folyamatok néhány sajátossága

A földtani folyamatokban több, kölcsönösen egymásraható tényező érvényesül, melyek hatása térben és időben változik.

A tanulmányozott Mecseki Uránérclelőhelyre vonatkozóan megállapitást nyert [1,2,3] hogy az ércképződés folyamatában számos földtani, ősföldrajzi, geokémiai, tektonikai és biológiai körülmény játszott szerepet. A lelőhely földtani fejlődéstörténete minden egyes szakaszának meg volt a jelentősége a jelenlegi érceloszlás kialakitásában. Jellemző, hogy az egymást követő szakaszok eredményei gyakran elfedik, esetenként megsemmisitik az előző szakaszok bélyegeit. Tehát a megfigyelhető jelenségek sok változó tényező egymásrahatása következtében alakultak ki és valószinüségi jellegüek.

A földtani dokumentációk tartalmazzák az ásványi nyersanyag kifejlődésre jellemző és attól független adatokat is. Ezért a matematikai statisztika módszereivel szét kell választani az ércesedéssel ténylegesen kapcsolatban lévő alapvető jelenségeket a háttérjelenségektől.

2. Gépi feldolgozás rendszere

Jelen dolgozatunkban csak a földtani jelenségekkel foglalkozunk; mellőzzük a szintén fontos szerepet játszó geofizikai adatok lehetséges bevonását.

2.1 Feldolgozásra kerülő földtani jelenségek kiválasztása és kódolása

Az érckutató furásokra rendszeresitett dokumentációkból számitógépes feldolgozásra kerültek a kőzetek: rétegvastagsága, szemcsenagysága, osztályozottsága, kötőanyaga, szine, rétegzettsége, kimosási-felületek, földtani kifejlődés /fácies/, kavicsok nagysága, összetétele és az ércre vonatkozó közvetlen adatok, mivel ezek meghatározása közelitőleg objektiven elvégezhető.

A földtani jelenségek diszkrét megfigyelések. A kódszámrendelésnél csak akkor jelentkezett nehézség, amikor az adott jelenségnek nincs közvetlen fizikai mértékegysége /pl. a szürkét is magában foglaló szinváltozatoknál, avagy a szemcsehalmaz egynemüségét kifejező osztályozottságnál stb./.

A kódolás adatlapokon kézi uton történt. Feldolgozásra került 180 kutatófurás ércesedés szempontjából fontosnak itélt mélységköze. Egy-egy kutatófurás 1500-4000 adatot tartalmaz.

2.2 A földtani megfigyelések feldolgozása

Az input formátum kialakitásánál figyelembe vettük a geológiai dokumentálás szokásait. A lyukkártyák beolvasása során a megfigyelések kódjait értelmezhetőség szempontjából a program /BRO7/ ellenőrzi.

Az adatok rendezését könyvtári program végzi /XSMC/. Az ICL 1900 sorozatu FORTRAN filek a szabványos file felépitésen belül további megkötéseket igényelnek, igy a rendezett file-t alkalmassá kell tenni további feldolgozásra [4] .

Az adatok értékelését a matematikai statisztika módszereivel végezzük /BRO1/. Az eredményeket /várható értékek, szórások, momentumok, sürüség - és eloszlásfüggvények, t, F és χ^2 próbák, trendek/ mágnesszalagos filen tároljuk, a táblázatok elemzését lyukszalagon, illetve lyukkártyán megadható feltételek mellett program végzi /BE05/, amely ugyanakkor a könyvtári statisztikai program [5] számára input filet állit elő.

2.3 Többváltozós regressziószámitás

Az előzőekben ismertetett problémánál a rendelkezésre álló könyvtári statisztikai rendszerből a többszörös regressziószámitás alkalmazása volt célszerü.

Az alapfeladat a következő: Keressük valamely valószinüségi változó M /η / x/ feltételes várható értéket. Feltesszük, hogy a feltételes várható-érték valamely y=f/x,a/ függvénynyel közelithető, majd az a értékét meghatározzuk olymódon, hogy a hibát valamilyen értelemben minimalizáljuk. A fenti program lineáris függvénykapcsolatot tételez fel, ez azonban nem jelent megszoritást, mert FORTRAN nyelvü transzformációkkal a linearitás többnyire megvalósitható.

Mivel a megfelelő valószinüségi változók együttes eloszlása nem áll rendelkezésre, ezért a paramétereket a mintából becsüljük.

A regressziós modell a következő:

$$\underline{y} = \underline{a}\,\underline{X} + \underline{u}$$

ahol: $\underline{y}$ - a függő változóra kapott minta;
$\underline{X}$ - a független változóra kapott minta;
$\underline{u}$ - valószinüségi hibavektor.

Jelentse $\hat{e}_i$ az i-edik megfigyelt és az i-edik $\underline{a}$-val becsült függő változó közötti különbséget

$$\hat{e}_i = y_i - /x_{io}, x_{i1}, \ldots\ldots, x_{in}/\ \underline{a}$$

ahol n - a független változók száma.

Az a vektort, amely a regressziós egyenlet együtthatóit tartalmazza úgy határozzuk meg, hogy

$$\hat{e}^T \cdot \hat{e} = \min$$

teljesüljön. Az említett könyvtári programot /XDS3/ sikerült szervesen beépíteni a rendszerünkbe. Rendkívül rugalmasan alkalmazható, mivel a rendelkezésre álló számítógép konfiguráció mellett egyszerre kb. 110 változót tud kezelni. A változók közül - megadott szignifikancia szint mellett - a Student kritérium alapján kiválasztja azokat, melyek a függő változóval kapcsolatban állnak. A számítás során több mint 200 függvényt használtunk a linearizáláshoz fel, míg az optimálisnak kapott regressziós egyenletet megkaptuk.

3. Következtetés /Tapasztalatok/

A rendszer algoritmusai geológiában és számítástechnikában jártas szakemberek együttműködésével alakultak ki. Sikerült a geológiai alapon sikerült létrehozni a tanulmányozott földtani jelenségek stochasztikus modelljét, melyet valós adatok felhasználásával állítottunk fel. Jelenleg modellünk gyakorlati ellenőrzés alatt áll.

Eddigi tapasztalatok szerint megvalósítható a kutatófurások ajánlásának és ezzel a véletlenszerű adatok helyettesítése valószínűbb értékekkel. A kiértékelés megbízhatósági szintjének változtatása megkönnyíti a gazdasági döntést az ásványi nyersanyag művelhetősége tekintetében.

A számítógép alkalmazása lehetőséget nyújt a földtani jelenségek összetettebb vizsgálatára, mivel a regressziós egyenletbe bekerülő tényezők egyben genetikai kapcsolatot is jelentenek. Helytálló érvényességének kimutatása csak egzakt genetikai értelmezésen keresztül lehetséges.

Az irodalmi közlések szerint jól alkalmazhatók geológiai feladatok megoldására más módszerek is pl. faktoranalízis [6] fourier-analízis [7] lineáris programozás, Monte-Carlo módszer stb. A rendszerünk továbbfejlesztésére a faktorana-

lizis látszik legkedvezőbbnek.

Végezetül meg kell említenünk, hogy Máriai Pál tudományos csoportvezető és Dravecz József okl. matematikus szintén jelentős munkát végeztek a feladat megoldásában. A földtani jelenségek kódolását Kerekes Jenőné geol.technikus végezte.

Irodalom

1. **А Барфаш, К Вираг /1966/ Механизм образования осадочных урановых руд на примере Мечекского месторождения. Лит. л.и. пол. иск. № 2.**

2. Virágh Károly-Vincze János /1967/: A Mecseki Uránérclelő hely képződésének sajátosságai. Földt.Közl. 97.

3. Virágh Károly-Szolnoki János /1970/: Baktériumok szerepe a mecseki uránérc keletkezésében és későbbi áthalmozódásában. Földt.Közl. 100.

4. Ambrus Zoltánné-Révész Ferdegúz /1972/: Mágnesszalagos FORTRAN file-k rendezése NIM [illegible]SZI /kézirat/

5. Révész Bendegúz-Szterhás Sándor /1972/: ICL statisztikai programrendszer: Regressziószámítás NIM [illegible]SZI /kézirat/

6. **В.Я. Воробьев - В. Жукова /1968/: Выбор информативных показателей в геологических задачах классификации, Геохимия № 5.**

7. Merrian D.F. /1967/: Comporter aids exploration geologists The Oil and Gas Journal jan.

8. Harman H.H. /1960/: Modern factor analysis University of Chicago Press, Chicago

EGY KÜLKERESKEDELMI VÁLLALAT ADATFELDOLGOZÓ PROGRAM-RENDSZERE

Kostyán Ákos

KGM ISZSZI

Ezt a program-rendszert három programozó kollégámmal 1968-tól 197o-ig készitettük el. Ez idő alatt állandóan kapcsolatban voltunk egy rendszerszervező kollégánkkal, aki feladattá fogalmazta számunkra a megrendelő igényeit. Összesen 54 programot irtunk, amiből jelenleg 48-at tartalmaz a rendszer.

Ügyfelünk legfontosabb kivánsága az volt, hogy input adatainak feldolgozása naponta történjék. Emiatt legelőször egy működtető rendszert készitettem /erről egy másik dolgozatban számolok be/, mert a gépünk nem rendelkezett ilyennel. Az elkészült működtető rendszer mágnes-lemezen tárolt, és könyvtár szervezése 38 felhasználói program kezelését teszi lehetővé.

A rendszer-lemezeket ugy lehetett kialakitani, hogy az egyiken az egyszeres- és a naponta ismétlődő, a másikon pedig a dekád- és a havi menetek programjai vannak tárolva. Néhány programunkban - főleg amelyeket utóljára irtunk - gondoskodunk a következő program behivásáról, de ez általában még a gépkezelő feladata.

Jelenleg kényelmetlenséget okoz, hogy a géphez járó szellemi termékből sajnos /megszokásból/ felhasználtuk az adatállomány-meghatározó kártyák futási időben való beolvastatását, és emiatt a gépkezelő csak akkor automatizálhatja a folyamatokat, hogyha a programok nevét tartalmazó vezérlő-kártyák közé ezeket is -- kötött sorrendben! -- elhelyezte. Természetesen, ha módunk lesz egy-egy állomány számára mindig ugyanazt a lemezt /szalagot/ biztositani, a vezérlő- és meghatározó-kártyák csomagjait állandósitva, a kényelmetlenséget kiküszöbölhetjük.

A napi feldolgozásban javitólistát, export számlákat, export számla összesitőket, belföldi számlákat, belföldi számla összesitőket, szállitók összesitőjét, vevők összesitőjét és dekád-állományt ellenőrző listát készitünk.

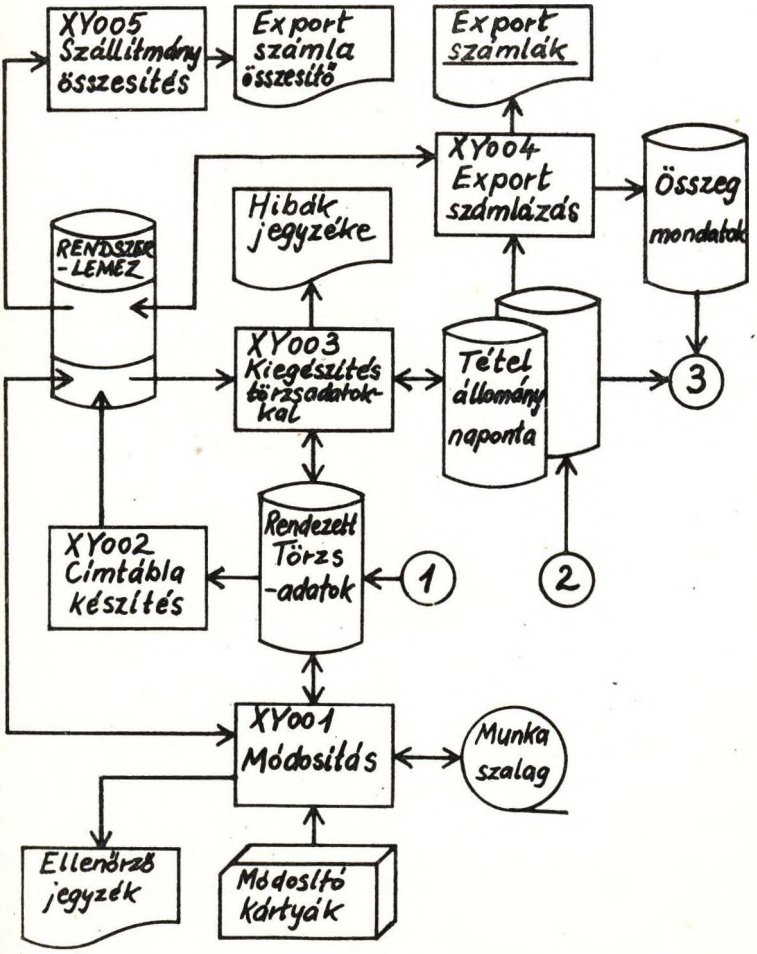
A dekád feldolgozásban készül raktárforgalmi kimutatás, bizományosi elszámolás, kötésenkénti bonyolitás listája, készlet--lista, nem bonyolitott kötések listája, elszámolás a csomagolóanyagokról és ellenőrző lista. Havonta a jutalékok és a csomagolóanyagok elszámolását készitjük.

Azokat a programokat választottam ki beszámolóm tárgyául, amelyek legkifejezőbben mutatják meg a feldolgozás ujszerüségeit a korábbi munkáinkhoz képest. A feldolgozás legfontosabb része egyébként is az exportált áruk sürgős számlázása a jó devizagazdálkodás érdekében. A bizonylati adatokból naponta kell a számlákat elkésziteni; ezt a tevékenységet pedig összekötjük a teljes feldolgozás adatainak előkészitésével, kialakitásával.

A bizonylatok adatait mágnes-szalagról egy 2 mágnes-lemezes tétel-állománnyá irja át a rendszer. A tételeket igen sok állandó adattal kell kiegésziteni, ezeket egy törzs-lemez tartalmazza. A törzsállományt alkotó mondatok /record/ lo csoportot képeznek: ezekből a legkülönfélébb módon épülnek be az adatok a tételekbe, ahogyan azt a tételek tipusa és a bennük lévő kereső-adatok előirják. Egy tételhez a lo csoportból 7-nek a mondatait kell meghivni. A törzsmondatok hossza egységesen 8o jel, a lemez 1 szektorában 1 mondat és 2o üres jel van; a terület igény legfeljebb lo ezer szektor. A tételek egyenként 4 szektort igényelnek a soros hozzáférésü iker-lemezeken.

A törzsállomány a tételek kereső-adatainak megfelelő 7 jelü kulcsok szerint rendezett. A 7 jelből 2 a lo csoportba sorol, a többi 5 a kereséshez kell. Mivel naponta módositani is kell a törzsadatokat, nagyon előnyös a rendezett fekvés.

A napi feldolgozás egy részletét az alábbi ábrán láthatjuk. Az 1-es pont azt a folyamatot jelenti, amelyben lemezre kerültek a törzskártyák, és be is rendeztük a lemez tartalmát. A 2-es pont a bizonylati input adatokat előkészitő programsort jelenti, a 3-as pedig a további meneteket és a tárolást dekádra.



Az XYool program a módositó kártyákat kötetlen sorrendben a rendszer-lemez elejétől felirja, majd sorrendben feldolgozza, összevetve a törzsadatokkal. Az eredményt szalagra kiirja. Visszatekercsel és a szalagról az uj lemez-tartalmat kialakitja. A szalag mindig alkalmas reprodukálásra, ha a törzs-lemezt baj érné. Automatikusan behivja az XYoo2 programot.

Az XYoo2 tartalomjegyzéket és cimtáblát készit a rendszer-lemez elejétől. Ezek között és az aktuális törzsállomány között döntő kapcsolat van: az XYool éppen azért teszi tönkre kártya-képekkel a tartalomjegyzéket /cimtáblát/, hogy az uj törzsállományról uj cimtábla készitése el ne maradhasson.

Az XYoo3 a tételeket beolvassa, kiegésziti és visszairja. Ha egy tételt nem tud kiegésziteni, kinyomtatja a hibás kereső-adat/ok/ aláhuzásával, majd az összetartozó tételek közül az elsőt ¤oSSSSSo SKIP alaku jelsorozattal törli, amelyben oSSSSS a hibás tételcsoportot követő szektorcim. A program csak akkor működik, ha a rendszer-lemezen megtalálja a tartalomjegyzéket, amit a magtárolóban helyez el és segitségével keres a cimtáblában. Mivel a cimtábla rendezett, nem olvas be, csak összehasonlit. A megtalált cimtábla-szektorból veszi a kulcshoz tartozó cimet, amivel beolvas a törzs-lemezről. A kiegészités befejeztével számlaszámokat képez, ezekkel a törzsállomány ötödik csoportját mindig naprakészen is tartja.

Az XYoo4 készitì az export számlákat. Az előző programtól mindent megkap, ami ehhez kell. Nyomtatványra ir 4 egyforma példányt; a rendszer-lemez munka-területén alakitja ki a lapot, hogy egymás mellett és után 2-szer kiirhassa. A terület használata nem ujrakezdő, hanem folyamatos, hogy az XYoo5 onnan megkaphassa a már elkészült "kimutatást", mert ebből a legkönnyebb elkésziteni a hozzá tartozó összesitést.

Kissé bővebben minderről az előadásomban szeretnék szólani.

A KERINFORG GAZDASÁGMATEMATIKAI RENDSZERE BEVEZETŐ ELŐADÁS

Skrabski Árpád
KERINFORG

A gépi adatfeldolgozásba bevont vállalatoknál egyre fokozódó igény jelentkezik arra, hogy a feldolgozott adatokból további elemzésekkel a vezetők számára könnyen áttekinthető, döntéseiket előkészitő információkat nyerjünk. A gazdaságmatematikai programrendszer fejlesztésénél a vállalati igényeken tul figyelembe vettük a minisztériumi döntéselőkészitő elemzések, a közgazdaság és társadalomtudomány kutatása és a piackutatás területén támasztott követelményeket is.

Az ismertetésre kerülő programrendszerek mindegyike programmodulokból áll.

Az adatkezelő rendszer input-modulja mágneses tárolóra viszi fel az alapadattömeget. Az adatbázis kezelését a programmodulok összefüzését és az összes matematikai feldolgozást ezek után egy "report language" /vezérlőnyelv/ és paraméterrekordok segitségével az adatkezelő rendszer automatikusan végzi el.

A programrendszerek között az adatok közlése az egységesitett un. bázis-formátumon történik. A feldolgozásra előkészitett alapadattömeg rekordjai egységes szerkezetüek, a rekord elemei alfabetikus szöveggel is megnevezhetők. Az eredménytáblákon a változónevek alfabetikusan feltüntethetők.

Az adatkezelő rendszer közvetlenül hivja az egyes programrendszereket. A rendszerek közül ismertetésre kerül a lineáris modell, lineáris és nonlineáris regresszió, trend és prognosztikai modellek és a készletgazdálkodási rendszer. Nem kerül ismertetésre a matematikailag egyszerübb STATO statisztikai program, Felsővályi Ákos munkája, amely az

adatkezelő rendszer segitségével többszempontos gyakoriságtáblák felállitására, a százalékos gyakoriságok kiszámitására és a szempontok között feltételezett összefüggés matematikai statisztikai bizonyitására vagy elvetésére szolgál. A rendszer 64 szempontból tetszőleges számu kétszempontos tábla felállitására képes, szempontonként maximálisan nyolc /összesen max. 64/ paraméterekkel megadható csoportba számlálva a megfelelő rekordokat.

Az együtt működtetett SAMPO adatkezelő rendszer és a STATO statisztikai rendszer nagytömegü adat pl. piackutató kérdőiv, összefüggéseinek rutinszerü értékelésére alkalmas.

A teljes rendszert eredményesen alkalmaztuk vállalati adatok elemzésére. A feldolgozásnál az adattömeget az Intézetünknél végzett gépi adatfeldolgozásból havonként idősorok alakjában nyertük. Rutinszerü vállalati feldolgozásoknál a rendszer vezérlését szintén mágneses tárolón rögzitjük. Ebben az esetben a rendszer működtetése az operátorok számára is igen egyszerü feladattá válik.

Az alkalmazások más területén kérdőivekről vagy a statisztikákkal nyerhető adattömeget visszük fel adatbázisba kutatási célból. Ebben az esetben a feldolgozások menete a részeredmények alapján módositható. Rugalmas eszközt kaptunk arra, hogy gazdasági hipotéziseink igazolását vagy cáfolását gépi feldolgozás segitségével rövid idő alatt elvégezhessük. A programrendszert a közgazdasági elemzéseken kivül eredményesen alkalmaztuk orvosi epidemiológiai kutatásban is.

A programrendszerek nagy részéhez a matematikai előkészitő munkát Éltető Ödön és Vita Zoltán végezte, szeretném igen értékes közreműködésüket ezuton is megköszönni.

LINEÁRIS MODELLEK PROGRAMRENDSZERE

Nagy Endre
KERINFORG

1. Bevezetés

Az ökonometriai elemzések egyik legjellemzőbb feladata annak meghatározása, hogy valamely véletlen hatások által befolyásolt mennyiség viselkedését adott magyarázó változó-rendszer ismeretében leirja. Az alapfeltevés szerint az eredményváltozó és a magyarázó változók között determinisztikus jellegü kapcsolat van.
A lineáris modellek determinisztikus jellegüknél fogva a gazdasági döntések előkészitésében kiemelkedő fontosságuak. Feltételezve, hogy a modell magyarázó változóinak legalább egy része befolyásolható /előirható/, a modellt célfüggvényként tekintve lineáris programozási eljárások alkalmazására van lehetőség.

2. Az alkalmazott matematikai apparátus

A modell paramétereinek becslése a legkisebb négyzetek módszerének alkalmazásával történik. A paraméterek szórásának, adott megbizhatósági szintre vonatkozó konfidencia-intervallumának meghatározására a matematikai statisztika ismert összefüggései szolgáltak. A modell jóságát a reziduális változó szórásával, illetve az illeszkedési együtthatóval határoztuk meg. Két tipusu hipotézisvizsgálatot alkalmaztunk: vizsgálhatjuk, hogy a pontrendszer leirható-e egy előre meghatározott együtthatóvektorral, illetve azt, hogy az egyes változók elhagyása javitja-e a modell jóságát. A modell ismeretében előrejelzés végezhető, s számithatók az adott megbizhatósági szintü előrejelzési intervallumok is.
A témakör kidolgozott volta lehetővé teszi, hogy azokat

az eseteket is megvizsgáljuk, ahol a reziduális változó viselkedése folytán a lineáris modell torzulást szenved. A programrendszerben a tiszta autokorrelált és a tiszta heteroszkedasztikus modellt vizsgáltuk. Az elsőnél különböző pontokban fellépő véletlen hatások között sztochasztikus kapcsolat van, míg a másiknál a reziduális változó szórása nem független attól, hogy a rész-megfigyelések melyik halmazára számítjuk.
Előírt feltételek teljesülése esetén a modell paramétereinek torzítatlan és konzisztens becsléseit az említett két esetben a programrendszer meghatározza. Mindkét speciális esetben meghatározhatók a szórások és konfidencia-intervallumok, a hipotézisvizsgálatok szintén elvégezhetők.

3. A programrendszer programjai

A rendszer három programból és egy segédprogramból áll.

3.1. LIMOD-program

Független, állandó szórású reziduális változó esetén alkalmazható, outputját a 2. pontban felsoroltak képezik.

3.2. LAMOD-program

Autokorrelált reziduumokat tartalmazó lineáris modell becslésére alkalmazható. A becslés iterációs eljárással történik, a kezdeti értékek a LIMOD által meghatározott paraméterek.

3.3. LHMOD-program

Heteroszkedasztikus jellegű lineáris modellek becslésére alkalmazható. Az outputra és a módszerre vonatkozóan hasonló a LAMOD programhoz.

3.4. LIMUP-segédprogram

Mivel a reziduális változó viselkedése ritkán adható meg előre, szükségessé vált egy speciális vizsgáló program elkészitése. Ez feltételezi, hogy a LIMOD program már előzőleg lefutott, vagy a modell egyenlete korábbról már ismert. A modell alapján meghatározható reziduumok autokorrelációs együtthatóit, ill. a részhalmazokra vonatkozó reziduális szórásokat számitja.

4. A programok specifikációja

A programok futásához szükséges minimális konfiguráció: 96 K központi memória, 2 lemezegység, 1 szalagegység, egy-egy kártyaolvasó és sornyomtató. A futtatás feltétele az általunk alkalmazott szabványositott adatkezelő rendszer használata és a küszöbértékeket tartalmazó file lemezreirása. A modell 31 magyarázó változót tartalmazhat. Az adatok száma maximálisan 600 lehet.

5. A programok felépitése

A szubrutinos felépités lehetővé tette, hogy a programokon belüli láncolás nem vált szükségessé.
Más programjainkhoz hasonlóan itt sem kötöttük meg az alkalmazó kezét, a vezérlő kódszám-rendszer alkalmazása lehetővé teszi, hogy bizonyos müveleteket /pl. előrejelzés, hipotézisvizsgálat/ elhagyhasson. Ugyancsak lehetőség van az alkalmazó által irt kiirató program használatára. A flexibilis alkalmazhatóságot a programok moduláris felépitése biztositja. Egy-egy modulon belül több speciális, az illető programra jellemző szubrutin foglal helyet, ezeken belül további általános rendeltetésü /mátrixaritmetikai/ szubrutinok vannak. Egyes modulok mindhárom programrendszerben megtalálhatók.
A fentiekből következik, hogy a file-ok felépitése rögzitett, az adattömegek meghatározott helyet foglalnak

el. A három programban alkalmazott modulok következők:

- rendező-modul /a számitás kiinduló alapját képező mátrixok előállitására/;
- illeszkedés-vizsgáló modul /az illeszkedési együttható, a reziduális szórás és a relativ reziduális szórás számitására szolgáló szubrutinok/;
- megbizhatósági modul /a modell becsült eredményváltozója konfidenciaintervallumának becslésére szolgáló hierarchikus felépitésü szubrutin/;
- előrejelző modul /az előrejelzett értékek és az előrejelzési intervallumok számitására szolgáló hierarchikus rendszer/;
- hipotézis-modul /a hipotézisek ellenőrzésére/;
- kiiró modul, amely az alkalmazó által irt programmal helyettesithető.

A modell paramétereinek, valamint a modell eredményváltozóinak becslésére a LIMOD-programban egy kizárólag mátrixaritmetikai szubrutinokból álló modul szolgál. A másik két speciális programban ez a modul a fentieken kivül más jellegü szubrutinokkal is bővül /adatgeneráló, számitó és összehasonlitó szubrutinok/.

Az adatgeneráló szubrutinok alkalmazását az teszi szükségessé, hogy az algoritmus szerint olykor 600x600-as mátrixokat kellene létrehozni, ami ugyan megvalósitható, de a futási időt nagymértékben megnövelné. Ezért a program felépitése során olyan algoritmusokat dolgoztunk ki, amelyek alkalmazásával a közbenső számitási müveleteket átugorva közvetlenül a kisebb méretü eredménymátrixot generáljuk. Az algoritmusok kidolgozásánál a mátrixok esetleges szimmetriáját vagy diagonális voltát maximálisan kihasználtuk.

A LAMOD-program paraméter-becslő modulja egyetlen ite-

rációs eljárásból áll. Feltevés szerint a reziduális változó autokorrelációs együtthatója adott törvényszerüségnek engedelmeskedik. Az iterációs eljárás során a reziduális változó autokorrelációs együtthatójának értékét módositjuk mindaddig, amig két egymást követő becslés reziduális szórásainak eltérése el nem hanyagolható.

Az LHMOD-programnál a becslő modul kétszeres iterációs eljárást rejt magában.Itt nemcsak a speciális jelleget leiró együtthatót, hanem a modell paramétereit is módosithatjuk a fent emlitett határig.

A program futtatásához három file /input-, munka-, küszöbérték-file/ szükséges.

6. Alkalmazási rendszerek

A program a kódvezérlésü CALL PROG utasitások felhasználásával a SYSTØ általános statisztikai rendszerbe, valamint a SYSTI és SYST2 prognosztikai rendszerekbe kapcsolható.

További hat lehetőség van arra, hogy az alkalmazó az általa irt feldolgozási rendszer tagjaként is használhassa mindhárom programot.

S T O C K
KÉSZLETEZÉSI PROGRAMRENDSZER

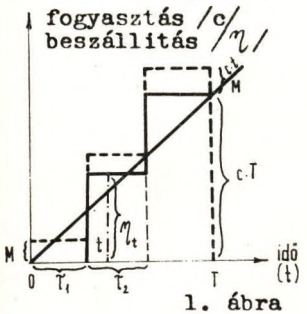
Vass István
KERINFORG

A STOCK készletezési programrendszer alkalmas arra, hogy a hazai kereskedelem sajátosságainak megfelelően a fogyasztók lehet ő legjobb ellátását szem előtt tartva optimális készletszinteket adjon meg, figyelembe véve a kereskedelmi vállalat rendelkezésére álló anyagi eszközöket. Az alkalmazott matematikai modell nem egyezik meg a nyugati készletezési package-kban általánosan használt, a rendelések számát megadó és költségoptimalizáló /beszerzési és készletezési költségekre/ modellel.

Az alkalmazott matematikai modell.

Az előzetesen végzett módszertani kisérleteink alapján kereskedelmünk sajátosságait a Véletlen nagyságu és ütemezésü részszállitmányok /Dr. Prekopa - Dr. Ziermann munkája/ cimü modell közeliti meg a legjobban.
A nevezett modell az alábbi beszerzési és készletezési politikára épül: egyenlő időközökben rendelést adunk fel, amelynek hatására a szállitó cég véletlen időközönként és véletlen nagyságu részszállitmányokban juttatja el az árut. A fogyasztás intenzitását egyenletesnek feltételezve a fogyasztás ill. a beszállitott árumennyiség változását az 1. ábrán követhetjük figyelemmel.
Az ábráról leolvashatjuk, hogy az első tétel beérkezéséig a fogyasztói igényeket nem lehet kielégiteni, mert eddig az időpontig a beszállitott mennyiség $\eta_t = 0$ /ha $t<\tau_1$/.
A probléma ugy oldható meg, ha a raktárban már a 0 időpontban egy bizonyos nagyságu készlettel rendelkezünk. Ezt a készletet nevezzük biztonsági készletnek /M/. Mivel

a szállitmányok beérkezése mind időben, mind nagyságban véletlenek csak egy bizonyos valószinüséggel tudjuk biztositani a fogyasztók ellátását.



1. ábra

T : két megrendelés közti idő

c : a fogyasztás intenzitása

M : biztonsági készlet

η_t: a "t" időpontig beszállitott árumennyiség

Ha a kivánt valószinüséget %-osan az /1-ε/.100 kifejezés adja meg, akkor célunk matematikailag a következő valószinüségi egyenlőséggel fogalmazható meg:

$$P\left[\min\ /M+\eta_t - c.t/ > 0\right] = 1 - \varepsilon \qquad [1]$$

E valószinüségi egyenlet közelitő megoldása a levezetést mellőzve:

$$M = c.T\sqrt{\frac{1}{2n}\log\frac{1}{\varepsilon}}$$

amely a következő feltételek mellett teljesül:

- a szállitott részszállitmányok egyenlő nagyságuak és
- egy megrendelés hatására minimum 8 részszállitás történik,
- a fogyasztás intenzitása állandó.

A valóságban rendszerint mind a részszállitmányok nagysága, mind a fogyasztás intenzitása változó. Ezeket a tényezőket is figyelembe véve a biztonsági készlet:

$$M = c.T\sqrt{\frac{1}{2}\left[\frac{1+/1-\lambda/^2}{n} + \frac{1+/1-\mu/^2}{m}\right]\log\frac{1}{\varepsilon}} \qquad [2]$$

A fenti képletben a már ismert jelöléseken kivül:

$\lambda = \frac{n \cdot \sigma}{c \cdot T}$ a szállitott mennyiség ingadozását veszi figyelembe

$\mu = \frac{m \cdot K}{c \cdot T}$ a fogyasztás intenzitásának változását veszi figyelembe

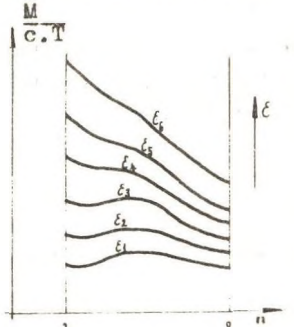
σ a legkisebb részszállitmány

m az értékesitések száma

K a legkisebb értékesitett mennyiség

A [2] képlet, tehát tartalmazza azt a korlátozást, hogy a szállitások száma minimum 8. /A közelitő megoldás csak e korlátozás mellett ad jó eredményt./

Ha a szállitások száma 8-nál kisebb, akkor az [1] egyenletet az alábbi ábra segitségével határozhatjuk meg:



2. ábra

Az ábrában az ε ellátási szinvonal paraméterként szerepel és az időszakra eső fogyasztás /c.T/ ismeretében a biztonsági készlet a szállitások számának /n/ függvényében az $\frac{M}{c.T}$ arányból számolható. A görbék pontjait megfelelő pontossággal discen tároljuk, az "n" és az "ε" fv.-ében kereshetjük meg az M/c.T arányokat.

A programrendszer feladata

A STOCK készletezési programrendszer feladata, hogy a készletezési határköltség elmélete alapján a vállalat anyagi eszközeit figyelembe vévé olyan készletszinteket állitson fel, amellyel a vásárlói igényeket megfelelően ki tudja elégiteni egy bizonyos valószinüségi szinttel. Két alternativát jelölhetünk meg a rendszer számára:

1. Előirjuk, hogy az egyes cikkekből milyen valószinüséggel álljon a vásárlók rendelkezésére áru és ennek függvényében meghatározzuk, hogy ez mekkora anyagi

eszközöket köt le.

2. Megadjuk vagy meghatározzuk a rendelkezésre álló anyagi eszközöket és kiszámitjuk, hogy ezzel mekkora készleteket tarthatunk és milyen ellátási szinvonalat biztosithatunk.

Input adatok

A rendszer előnyének tekinthető, hogy az input adatok tekintetében rugalmas. Ez azt jelenti, hogy több formában adhatók meg a kiinduló adatok és kódkártyán jelöljük meg, hogy milyen adatokból indulunk ki. Inputként a következő adatok jelenhetnek meg:

- az egy-időben feldolgozásra kerülő cikkelemek száma /max. 80/
- az összes értékesitési adatok száma
- az ellátási szinvonalak
- értékesitési idősorok
- beszállitási adatok vagy a szállitások átlagos száma és a legkisebb tételnagyság
- egységnyi árura eső készletezési költség
- cikkcsoportra eső készletköltség
- cikkelemek és cikkcsoportok neve
- ha egyidőben több cikkcsoport feldolgozása történik, akkor a cikkcsoportba tartozó cikkelemek sorszáma
- a programrendszer vezérlését végző kódszámok.

A programrendszer működése

A rendszer működését tehát kódszámokkal vezéreljük, ezek száma 7. A kódszámoktól függően a rendszer a következő feladatokat oldhatja meg:

1. Megfelelő hosszuságu értékesitési idősor esetén az INFOR szubrutin előrejelzi a fogyasztást, ellenkező esetben a várható fogyasztás nagyságát átlagolással és az átlagostól való eltéréssel vesszük figyelembe.
2. A szállitások átlagos számának és a legkisebb tételnagyságok hiányában a rendszer kiszámolja ezeket.
3. A kódszám értékétől függően a fajlagos készletköltségek

ségek alapján vagy meghatározzuk a cikkelemre eső összes készletköltséget vagy nem. Lehetőség van a teljes készletköltség meghatározására is. /cikkcsop./

4. Ha kiindulásként a cikkcsoportra eső összes készletköltség áll rendelkezésünkre, akkor azt fel kell osztani a cikkelemek között. Ez esetben ki kell számolni azt is, hogy a rendelkezésünkre álló készletköltség milyen ellátási szinvonalat biztosit.
5. A megrendelő igénye esetén egy olyan tájékoztató tábla is kinyomtatható, amely felvilágositást ad arról, hogy a különböző ellátási szinvonalak esetén milyen biztonsági készletek tartandók a szállitások átlagos számának függvényében.

A közbenső eredmények után /amelyet igény esetén szintén kinyomtathat a rendszer/ végül kétféle eredményt kaphatunk:

- vagy a következő időszakra megrendelendő mennyiséget a meglevő készlet figyelembevételével cikkelemenként,
- vagy a rendelkezésünkre álló anyagi eszközökkel milyen szinvonalon biztosithatjuk a vásárlók igényeit, azaz egy tetszőleges időpontban milyen valószinüséggel találunk árut, ugyancsak cikkelemenként.

A rendszer jellemzői

A programrendszert a Honeywell 2200-as gépre készitettük el, FORTRAN F programnyelven /irva/ és a gazdaságos memóriakihasználás érdekében a szubrutintechnikára épül. Igy kb. 32 K memóriát köt le, a futási idő a megrendelő igényétől függően 5-15 perc 80 cikkelemre. Lehetőség van arra, hogy egyidőben a feldolgozást az előadás elején emlitett két alternativa szerint végezzük el. A programrendszer módszertani próbája a jelenleg folyó adatgyüjtés befejezése után a következő hónapban kezdődik el.

PROGNOSZTIKAI PROGRAMRENDSZEREK

Nagy Endre
KERINFORG

1. Bevezetés

Tetszőleges gazdasági egység egymástól többé-kevésbé függő folyamatok összességeként fogható fel. Ezen folyamatokat leíró egyenletek rendszerét az adott egység modelljének nevezzük.

A modellek három csoportba oszthatók:

- egyszerű modellek, ahol a folyamatok nem hatnak egymásra,
- kauzális modellek, ahol a folyamatok között egyirányu /oksági/ összefüggések vannak,
- szimultán modellek, ahol a folyamatok kölcsönösen hatnak egymásra.

A SYST1, SYST2 és SYST3 programrendszer rendre a fenti modellek paramétereinek becslésére és a modellek alapján történő előrejelzés céljaira készült.

2. A modellekben alkalmazott algoritmusok

Az első két modellnél a legkisebb négyzetek klasszikus módszerének alkalmazásával jó eredményt érhetünk el. Rendre meghatározhatók az egyes egyenletek paramétereinek szórása, adott megbizhatósági szintü konfidenciaintervalluma, a reziduális változó szórása és az egyenlet illeszkedési együtthatója. Az egyetlen kikötés az előrejelzésre vonatkozik: a kauzális modellnél a modell eredményváltozóinak előrejelzése csak az oksági sorrendnek megfelelően történhet.

A szimultán modellnél a legkisebb négyzetek klasszikus

módszere, mint ismeretes, nem ad konzisztens becsléseket. A strukturális paraméterek becslését a legkisebb négyzetek többfokozatu módszereinek valamelyikével, vagy a korlátozott információn alapuló módszerrel becsülhetjük. A programrendszer felépitése valamennyi változat alkalmazását lehetővé teszi. A rendszer a paramétereken kivül a reziduális változók szórását, valamint a paraméterbecslések variancia-kovariancia mátrixát is számitja. Az előrejelzés a modell alapján elvégezhető. A modellek nem szoritkoznak statikus esetre, a magyarázó változók között az eredményváltozók tetszőleges késleltetésü értékei is szerepelhetnek.

3. Specifikációs adatok

A SYST1 és SYST2 programok futásához 96 K, a SYST3 futásához 64 K központi memórián kivül 2 lemezegység, 1 szalagegység, egy-egy sornyomtató és kártyaolvasó szükséges. Az alapadatok bevitele csak a szabványos adatkezelő rendszerrel végezhető el.
A modell változóinak száma max. 32, a modell egyenleteinek száma elvileg szintén maximálisan 32 lehet. A gyakorlatban az egyenletek identifikálhatósága miatt a modell ennél csak jóval kisebb méretü lehet. Az adatok száma maximálisan 600.

4. A programrendszerek felépitése

A rendszerek független programokból épülnek fel. A futás során a programok behivását a megfelelő vezérlőprogram végzi. A feldolgozás elvégzésére természetesen a vezérlőprogram alkalmazása nélkül is sor kerülhet. A vezérlőprogram input-adatait a programok nevei alkotják.

A rendszert alkotó programok nagy része a szükséges matematikai eljárások algoritmusait tartalmazzák. A mátrixmüveleteket minden esetben szubrutinok végzik.

A rendszerhez egy speciális, a modell egyenletei alakjának, jellemzőinek bevitelére szolgáló program és egy inputprogram tartozik. Az esetleges hibák hatásának kiküszöbölésére, ill. az adott munka célszerű szervezésére, szolgáló check-point rendszert az operációs programrendszerhez tartozó UNLOAD és LOAD programok alkalmazásával valósitottuk meg. E két program a prognosztikai rendszer tetszés szerinti, részprogram után nagyméretü modell esetén valamennyi részprogram után beirható, akár közvetlenül egymás után, akár időben szétválasztva. Alkalmazásuk során a discen tárolt részeredményeket szalagra, majd innét vissza lehet vinni.

4.1. SYST1-egyszerü modell

Az S1PØ1 program a modell egyenleteinek jellemzőit /egyenletek száma, késleltetett változók, a késleltetés mértéke, az egyes egyenletekhez tartozó magyarázóváltozók száma, stb./ rögziti. Az előrejelzés elvégzéséhez szükséges adatok, a magyarázó változók előrejelzett értékeinek bevitelére az S1PØ2 program szolgál. Ezek az adatok vagy a TREND programból nyerhetők, vagy előirások alapján adhatók meg. A strukturális paraméterek becslését és egyéb szükséges becsléseket az S1PØ3 program végzi, melyet az S1PØ4 kiirató program követ. E két programot annyiszor kell behivni, ahány egyenletből a modell áll. Az S1PØ3 program lényegében a LIMOD-program egyszerüsitett változata. Az eredményváltozók előrejelzése az S1PØ5 programmal történik.

4.2. SYST2-kauzális modell

A rendszer felépitése megegyezik a SYST1 programrendszerével. Maguk a részprogramok is azonosak az S2PØ5 előrejelző program kivételével, itt ugyanis az oksági kapcsolat némileg eltérő algoritmust eredményez.

Mindkét rendszer - szükség esetén - felépithető a SYSTØ általános statisztikai programrendszer lényegesen több adatot szolgáltató programjaiból is. A rendszerből ekkor az S1PØ3 és S1PØ4, illetőleg az S2PØ3 és S2PØ4 programok kimaradnak. Helyüket valamelyik regressziós program és a teljesen azonos felépitésü S1PØ6 ill. S2PØ6 program foglalja el. Ez utóbbiaknak az a feladata, hogy az előrejelzéshez szükséges adatokat az S1PØ5 ill. S2PØ5 előrejelzőprogramoknak megfelelő formára hozza.

A megfelelő vezérprogramok felépitése lehetővé teszi, hogy ez utóbbi esetben is automatikus vezérléssel futtathatók a programok.

4.3. SYST3-szimultán modell

Az eltérő matematikai apparátus nem tette lehetővé, hogy az előbb leirt részprogramokat itt is alkalmazzuk. Ez alól kivételt csak az előrejelzési alapadatok bevételére szolgáló S3PØ2 program képez.

A modell alaki jellemzőit beiró S3PØ1 program az egyenletek identifikálhatóságának rend-feltételét is vizsgálja. Itt történik a legkisebb négyzetes becslés fokozatszámát meghatározó kódszám bevitele is. Zérus értékü kódszám a korlátozott információn alapuló módszer alkalmazását irja elő.

A legkisebb négyzetek meghatározott fokozatu módszerének alkalmazásakor a rendszer a következő programokból épül fel:

- S3PØ3: a változók átlagértékeit és átlagtól való eltéréseit számitja, valamint felirja a modell összes külső változóiból képzett Z_k mátrixot.
- S3PØ4: a Z_k mátrixból alkotott négyzetes mátrix inverzét számitja.
- S3PØ5: a modell adott egyenletéhez tartozó alapadatmátrixokat irja fel.

- S3PØ6: a strukturális paraméterek számításához és az identifikációs vizsgálat elvégzéséhez szükséges közbenső adatokat számitja.
- S3PØ8: folytatja a strukturális paraméterek becsléséhez szükséges mátrixok számitását.
- S3PØ9: az S3PØ8 program futása alatt képződő négyzetes mátrix invertálására szolgál.
- S3PlØ: számitja az egyenlet strukturális paramétereit, az egyenlet szabad tagját, a becslések variancia-kovariancia mátrixát és a reziduális változó jellemzőit.
- S3Pll: az egyenlet identifikálását végzi el az F-eloszlás megfelelő küszöbértékének figyelembevételével. A program szükség esetén elhagyható.
- S3Pl2: kiirató program, a számitások során kapott eredményeket nyomtatja ki.

Az S3PØ5, S3PØ6, S3PØ8, S3PØ9, S3PlØ, S3Pll és S3Pl2 programok annyiszor ismétlődnek, ahány egyenletből a modell áll. Kivételt képez az az eset, amikor a modell azonosságot kifejező egyenletet tartalmaz. Ez esetben az S3Pl3 programot kell alkalmazni, amely csak a kiiratást és az előrejelzéshez szükséges együtthatómátrix megfelelő sorának generálását végzi el.

A korlátozott információn alapuló módszernél az S3PØ6 program után az S3PØ7 programot kell futtatni. Ez a program a legkisebb variancia-arány értékét számitja. Mivel ebben az esetben az identifikációs eljárás elvégzéséhez a strukturális paraméterek ismeretére nincs szükség, az egyenlet identifikálását is ez a program végzi. Ennek megfelelően az S3Pll identifikációs program futtatására nincs szükség. Az egyenlethez tartozó programsor ez esetben tehát S3PØ5-S3PØ6-S3PØ7-S3PØ8-S3PØ9-S3PlØ-S3Pl2.

Az előrejelzést a megfelelő mátrixműveletek elvégzése az esetleges készletezések figyelembevétele után az S3Pl4 előrejelző program végzi mindkét esetben.

A programokban szereplő szubrutinok a legkisebb variancia-arány számitásakor szükséges determináns-számitó szubrutinon, valamint az identifikáláshoz szükséges F-eloszlás küszöbértéket kikereső szubrutinokon kivül mátrixaritmetikai szubrutinok.

REGRESSZIÓANALIZIS PROGRAMRENDSZER

Gáti Zoltánné
KERINFORG

1. Bevezetés

A regresszióanalizis a feltételezett sztochasztikus kapcsolatban levő adatsorok közötti összefüggéseket vizsgálja és azt matematikai függvényekkel és jellemzőkkel irja le. Az elemző módszer lényege, hogy egyes változók értékeit meg lehet becsülni másik változók értékeiből egy megfigyelési sorozatban mutatkozó függvénykapcsolat alapján. Legegyszerübb esetben ez a függvénykapcsolat lineáris, bonyolultabb esetben valamilyen nem lineáris jellegü. Ezen két esetnek megfelelően két programrendszert fejlesztettünk ki.

2. Lineáris regressziós modell

2.1 A programrendszer által számitott értékek

A lineáris regressziós modell alkalmazásánál feltételezzük, hogy a változók mind valószinüségi változók. A rendszer a "legkisebb négyzetek módszere" segitségével becsüli a regressziós paramétereket, majd kiszámitja a függő változónak a regresszió alapján számitott értékeit. A residuumokat a függő változó megfigyelt és számitott értékeinek különbsége adja. Ezen értékek szórásának számitásával kapjuk meg a regressziós közelités relativ hibáját. A programrendszer számitja a változók átlagát, szórásnégyzetét, relativ szórását, kovarianciáit és a korrelációs együtthatókat. A korrelációs mátrix inverse segitségével becsli a parciális együtthatókat /korrelációs/ és a többszörös korrelációs együtthatót.
Amennyiben a változókra bizonyos feltételek teljesülnek a rendszer intervallumbecsléseket és bizonyos hipotézis-

vizsgálatokat is végez. Minden számitást megelőzve vizsgálja a függő és magyarázó változók korrelációjának szignifikánciáját. Ha egy magyarázó változó nem szignifikáns korrelációju, a modell további számitásaiból kimarad. Vizsgáltatni lehet a regressziós paraméterek és többszörös korrelációs együttható zérustól való eltérésének szignifikanciáját is. Konfidenciaintervallumok számithatók a közönséges és parciális korrelációs együtthatókra, a regressziósegyütthatókra, valamint a függő változó becsült értékeire. A rendszer számithatja az egyes magyarázóváltozók és függő változó közötti rugalmassági együtthatókat pontonkénti és átlagértékekre is.

2.2 A programrendszer felépitése

A programrendszer maximálisan 31 magyarázóváltozót tételez fel, a mintavételek száma maximálisan 2.000 lehet. A változók száma és a mintavételek száma a rendszer vezérlésében mint paraméterek szerepelnek.

A minimális futási idő lecsökkenthető, ha a modell csak a legszükségesebb paraméterek számitását végzi. A megrendelő kivánsága szerint a többi paraméter számitása kódszámmal vezérelhető.

A programrendszer láncolási és szubrutintechnika párhuzamos alkalmazásával épül fel. Egy-egy konkrét müvelet végrehajtása történik egy-egy szubrutinban.

A láncolt programrészek hivása kódszámokkal van vezérelve a szükséges kiegészitő számitásoknak megfelelően.

A programrendszer egységes output táblázatot ad, melyben soronként az egyes számitott paraméterek, mutatók szerepelnek. Mód van arra, hogy kódszámmal vezérelve külső programban meghatározott formátumu output tábla készüljön. A rendszer hardware igénye 64 K központi memória, 2 lemezegység, 1 kártyaolvasó és 1 kiiró egység. A felhasznált lemezfile közvetlen hozzáférésü.

2.3 Csatolás más rendszerekhez.

A programrendszer beépült bonyolult elemzőrendszerbe is. Ezt a csatlakozást kódszám segitségével biztosithatjuk. A regressziós elemzés elvégzése után a rendszer a kódszám értékétől függően visszaadhatja a vezérlést egy meghatározott elemzőrendszernek.

2.4 Mátrix müveletek elvégzéséhez használt szubrutinok

A mátrix müveletek gyakori előfordulása tette szükségessé külön szubrutinok kialakitását ezekhez a feladatokhoz. Ha a változók száma nagyobb 8-nál, a számitásban előforduló mátrixok nem férnek el egyszerre a memóriában. Ilyen esetben a mátrixot csak soronként olvashatjuk be és igy a számitási idő megnövekszik. Ezt az időnövekedést ellensulyozzuk, ha figyelembe vesszük, hogy az itt előforduló mátrixok szimmetrikusak, tehát a részmüveletek száma felére csökkenthető.

3. Nem lineáris regressziós modell

3.1 Input változók tipusai

A programrendszer egyszerre alkalmas olyan elemző vizsgálatra, amikor az összes változó valószinüségi változó, és olyanra, amikor csak a függő változó valószinüségi változó a többi matematikai változó. Az inputváltozók tipusa határozza meg a modell tipusát.

3.2 A programrendszer által számitott értékek

A programrendszer csak olyan nem lineáris modellekkel foglalkozik, melyek alkalmas transzformációval lineárissá alakithatók. A transzformált változókra alkalmazható a lineáris regressziós elemzés. A számitott paraméterek természetesen a transzformált változókra vonatkoznak.Az eredeti változókra vonatkozó paramétereket alkalmas visszatranszformálás után kapjuk. A transzformált változókra

vonatkozó korrelációs együtthatók értékéből nem következtethetünk egyértelműen az eredeti változók között fennálló sztochasztikus kapcsolat szorosságára, ezért helyettük a korrelációs indexet számoljuk. /Lineáris esetben megegyezik a többszörös korrelációs együtthatóval./ Ha a magyarázó változók matematikai változók, akkor ez az index nem a sztochasztikus kapcsolat szorosságának a mérőszáma, hanem a megfigyelt és becsült függő változók egyezőségének mértékét kifejező illeszkedési együttható.

A programrendszer számitja a reziduális szórást, és annak relativ értékét, amely szintén jellemzi a választott modell jóságát. A modell a regressziós paramétereket a transzformált változókra a "legkisebb négyzetek módszerével" becsli, majd a választott nemlineáris függvénynek megfelelően visszatranszformált értéket is ad.

Ugyancsak becsli a paraméterek és a függő változó becsült értékeinek szórását is. A programrendszer alkalmas a megfigyelési tartomány határain belül a függő változó értékének előrejelzésére, ezenkivül rugalmassági együtthatók számitására pontonkénti és átlagértékekre.

3.3 A programrendszer felépitése

A rendszer alkalmas 31 magyarázó változó értelmezésére. A mintavételek száma maximálisan 2.000 lehet. A változók számát, a mintavételek számát és az előrejelzési pontok számát a vezérlésben mint paramétereket megadjuk.

A modell tipusát és a nemlineáris függvény tipusának választását kódszámok segitségével adjuk meg. A rendszerben regressziós közelitésre felhasználható nemlineáris függvény hiperbolikus, parabolikus, exponenciális, multiplikativ és polinomiális lehet. A megfelelő transzformáció kiválasztása komoly vizsgálatot igényel. Két változós esetben célszerü egy pontdiagram készitése, melyet segédprogrammal végeztethetünk. A legszükségesebb paraméterek

méterek számitásán kivül kódszámmal vezéreljük a többi számitásokat, /paraméterek szórása, rugalmassági együtthatók stb./ a megrendelők kivánsága szerint ezzel a minimális futási idő lecsökkenthető.
A programrendszernek két fő része van, ahol külön-külön végzi a számitást kétváltozós és többváltozós esetre.
A kétváltozós rész még két részre oszlik, mivel egyes transzformációk többváltozós lineáris függvényt eredményeznek /polinomiális fv. stb./. A rendszer láncolási és szubrutin technika együttes alkalmazásával épül fel.
A fő részek láncolva vannak egymáshoz, mig a konkrét számitási feladatokat egy-egy szubrutin végzi.
A kódszámok értékétől függően történik a számitó szubrutinok hivása. Ez a vegyes megoldás gazdaságosnak bizonyult a helykihasználást és a futási időt vizsgálva.
A programrendszerben van egy beépitett kimeneti táblázat készitési lehetőség, azonban ezt a megrendelő kivánsága szerint le lehet tiltani. Ebben az esetben egy külső kiiró programot hiv be a rendszer, amelyet előzőleg a megrendelő kivánságainak megfelelően el kell készitení. A rendszer hardware igénye 64 K központi memória, 2 lemezegység, 1 kártyaolvasóegység és 1 kiiróegység.
A rendszer két lemez file-t használ, mindkettőt közvetlen hozzáféréssel.

3.4 Csatolás más rendszerekhez.

Ha igény van rá, hogy a transzformált adatokkal egy teljes lineáris regressziós vizsgálatot is végezzünk, akkor a rendszer megfelelő kódszám hatására az input file-ba a transzformált adatokat helyezi és automatikusan elinditja a lineáris modellt.
Elemző programrendszerekhez való csatlakozását is a kódszámok segitségével biztositjuk. A megfelelő kódszámoktól függően a rendszer az input file-ba az eredeti adatokat hagyva a nemlineáris regressziós elemzés befejezése után a vezérlést visszaadja egy meghatározott elemzőrendszernek.

IDŐSOROK ELEMZÉSÉRE SZOLGÁLÓ PROGRAMRENDSZER /TREND/

Nagy Endre
KERINFORG

1. Bevezetés

Gazdaságmatematikai számításoknál, de más területeken is gyakorta van szükség arra, hogy valamely idősor formájában adott mennyiség időbeli alakulásának törvényszerüségeit meghatározzuk és e törvényszerüségek ismeretében a szóbanforgó mennyiség jövőbeli viselkedését leirjuk.

2. A programrendszer matematikai háttere

2.1. Az idősorok komponensekre bontása

A rendszer elsődleges feladata, hogy az idősort komponenseire bontva /trend, szezonális hatás, véletlen /reziduális/ hatás/. Az idősort e komponensek összege /additiv modell/ vagy szorzata /multiplikativ modell/ adja meg. A trendfüggvény megválasztására öt lehetőség van: lineáris, polinomiális, exponenciális, hatvány- és logisztikus függvény. A polinomiális függvény max. hatodfoku lehet, az alkalmazható legmagasabb fokszámot a program automatikusan, F-próba alkalmazásával határozza meg.

A komponensekre való bontás iterációs eljárással történik. Az iteráció kezdeti lépése a mozgóátlagos trend meghatározása. A program az átlagolási ciklust a szezonális ciklussal egyező tartalmunak tekinti, igy az idősorból a szezonális hatás kiszürhető. Az átlagolás additiv modell esetén ezek logaritmusára történik. Ezt követően kezdődik a tényleges iterációs eljárás:

- a trendfüggvény paramétereinek becslése a legkisebb négyzetek módszerének alkalmazásával /megfelelő

transzformációs eljárásokkal/;
- a tényadatok szezonhatástól való megtisztitása.

Ez a három lépés mindaddig ismétlődik, amig két egymást követő felbontásnál a reziduális változó szórásai közötti különbség el nem hanyagolható.

2.2. A modell jellemzőinek meghatározása, előrejelzés

A program számitja az idősor és a trendből, valamint a szezonális hatásból álló szisztematikus komponens közötti illeszkedés szorosságát mérő illeszkedési együtthatót, a reziduális szórást, valamint ennek relativ értékét. Ez utóbbi az illeszkedési együtthatóval együtt a választott modell jóságára ad információt.

A program a fentieken kivül a trendfüggvény paramétereinek szórását is meghatározza. Az előrejelzés tetszőleges időtartamra végezhető el. Az előrejelzett értékeken kivül a tetszőleges megbizhatósági szinthez tartozó megbizhatósági tartományok is meghatározhatók.

2.3. A reziduális változó vizsgálata

A reziduális változó vizsgálata során a program számitja a különböző időpontokhoz tartozó reziduumok közötti összefüggés szorosságára jellemző szeriális korrelációkat, s adatokat szolgáltat az autoregressziós séma meghatározásához. Az autoregressziós séma meghatározását követően a program a séma paramétereinek konzisztenciáját is vizsgálhatja.

3. A programrendszer felépitése

3.1. A rendszer specifikációs adatai

A program egyidejüleg 32 idősort vizsgálhat. Az idősorok maximális hossza 600 adat, az előrejelzési pontok

száma maximálisan 3oo. A szezonális adatok számításának feltétele, hogy legalább öt szezonális ciklus rendelkezésre álljon. A reziduális változó vizsgálatához minimálisan 5o adat szükséges.
A program futtatáshoz szükséges minimális konfiguráció: 96 K központi memória, 2 lemezegység, 1 szalagegység, 1 kártyaolvasó és 1 sornyomtató.

3.2. Felépités

A minimális futási idő biztositása végett a program felépitése olyan, hogy a futás során csak az alapadatoknak megfelelő számitásokat lehessen elvégezni. A vezérlés kódszám-rendszerrel történik, a program futás közben csak a kódszámok értéke által meghatározott müveleteket végzi el. Ennek biztositása csak a program moduláris felépitésével volt biztositható. Egy-egy modul valamely számitási eljárás programját tartalmazza. A kompilációs kötöttségek miatt előfordul, hogy egy modult több szubrutinból álló rendszer alkot, általában azonban egy modulnak egy szubrutin felel meg.
A modulok két nagyobb csoportot alkotnak, elhelyezésük a Honeywell FORTRAN-compiler lehetőségeinek megfelelően chain-ekben történt. A láncolási eljárást a korlátozott memóriaméret teszi szükségessé, a futási idő minimálása végett azonban a lehető legkevesebb számu láncszemet kell alkalmazni. A láncszemek tartalmát az alkalmazott matematikai eljárások is meghatározták, az első láncszem szükségszerüen az iterációs eljárást tartalmazza, a második láncszem a fentmaradó modulokból áll. Maguk a modulok két csoportra oszthatók. Az elsőbe az általánosan alkalmazott szubrutinok tartoznak /mátrixmüveletek, Bernouilli-módszer, szórásszámitás, stb./ a másodikba csak a program által alkalmazott speciális szubrutinok.
A program a CALL PROG utasitás alkalmazásával lehetővé

teszi, hogy számitási eredmények kiiratása ne a program printermodulja szerint, hanem a felhasználó által irt és forditott kiiratő programmal történjék.

3.3. File-szerkezet

A program három file-t alkalmaz.

- az input-adatok tárolására /kapcsolódó programok esetén a további feldolgozások input-adatait is itt helyezi el a program/;
- a számitások során képződő adatmátrixok tárolására;
- a statisztikai próbák elvégzéséhez szükséges különböző eloszlások küszöbértékeinek tárolására /ennek tartalma változatlan, a file a programrendszer tartozékának tekinthető/.

A file-ok adatszerkezete kötött. Ez a kötöttség egyrészt a futási idő minimálását célzó törekvésből adódik, másrészt feltétele az alkalmazó által irt kiiratő program használatának.

4. Segédprogramok

Az alkalmazás hatékonyságának növelését két segédprogrammal biztositottuk. Ezek akkor használhatók, ha az idősort illetően nem tudunk feltevéssel élni, azaz kérdéses az alkalmazandó modell, ill. trendfüggvény tipusa.

4.1. TPROB-program

A program idődiagrammot rajzol az eredeti adatokra, ill. ezek logaritmusára. A diagramm képéből az alkalmazható modell tipusára és a trendfüggvényre következtethetni lehet.

4.2. GENPM-program

Az előbb ismertetett feladaton kivül lehetővé teszi, hogy a ponthalmazra tetszés szerinti függvényt illeszthessünk.

Az illesztés a lineáris alakra való transzformálás után a legkisebb négyzetek módszerével történik. A program a transzformált modell paramétereit és a modellre jellemző reziduális szórást számitja.

4.3. VARST-program

Ez a program olyan idősorok vizsgálatát teszi lehetővé, ahol a szezonális hatás stacionárius jellegének feltételezése nem jogos. Ezzel a programmal lehet olyan, a szezonális hatást is figyelembe vevő előrejelzést készíteni akkor is, ha a szezonális ciklusok száma ötnél kevesebb.

5. Általános statisztikai programrendszer

A TREND program más statisztikai programokkal együtt alkotja a SYSTØ elnevezésü programrendszert. A SYSTØ alkalmazásával tetszés szerinti feldolgozási rendszerek alakithatók ki.

A rendszer adatkezelő és statisztikai programokból áll. A programok inputjainak és outputjainak szabványositása lehetővé teszi, hogy a célnak megfelelő rendszert állithassunk össze. A rendszer futtatása tetszés szerint manuális, automatikus vagy kevert tipusu vezérléssel végezhető el. Az automatikus vezérlést a SYSTØ vezérlőprogram biztositja. E program input-adatai a programok nevei. A programneveket tartalmazó kártyák megfelelő sorrendü összeállitásával a kivánt rendszer egyértelmüen meghatározható. A vezérlő program müködését az egyes programok végére beépitett CALL PROG utasitások teszik lehetővé, ezek hivják vissza a vezérlő programot.
A CALL PROG utasitás argumentumában lévő programnevet egy kódszám értéke határozza meg, ezáltal biztositható, hogy a program a SYSTØ-on kivül valamely speciális vezérlőprogramot hivjon vissza.

EGY TÁBLÁZO PROGRAM

Máté Eörs
Kibernetikai Laboratórium

Bevezetés

Az adatfeldolgozási feladatok jelentős része a következő részfeladatokra bontható:

beolvasás,
file szerkesztés, rendezés,
táblázás.

Az utóbbi részfeladat programozásához kíván segitséget nyujtani a MINSZK-32-re irott TABUL táblázó program.

A TABUL nevü táblázó program egy adatfeldolgozási szempontok szerint kialakitott egyszerü célnyelv - a továbbiakban TPN /táblázó programnyelv/ három menetes forditóprogramja. Rendezett karaktersorozat file-ok táblázására szolgál. A TPN adatleirásokra, müveletvégzésre kontrolmegszakitások figyelésére alkalmas utasitásokat tartalmaz. A táblázó program a laprabontásról, fejlécek kiirásáról automatikusan gondokodik, lehetőséget nyujt cimlap és hátsó boritólap nyomtatására.

1. A táblázó programnyelv leirása

A TPN programot kártyatömbként kell perforálni.

1.1. A TPN programnak a ↑FILE utasitással kell kezdődni.

Ez az utasitás a feldolgozás alapját képző file adatainak leirására szolgál.

Az utasitás formája:

↑FILE: <a file-t tartalmazó mágnesszalag neve> :
<a file tömbazonositója> : <a file egy rekordjának hossza szavakban> ;

1.2. Adatleirásra szolgál a ↑FORMAT utasitás.

Az utasitás alakja:

↑FORMAT: <formanév> : <rekord hossz szavakban> :

P_1;P_2; ••• ; P_k;

ahol p_i A:n,m,r — A mezőnév
vagy A:n,m — n kezdő karakter pozíciója
vagy A:n, ' idézet — m a mező hossza karakterekben
r a tizedes jegyek száma

Az első esetben a mező szám, a továbbiakban szöveg jelentésü.

A rekordon belül a pozíciók számozása 0-val kezdődik.

A rekord hosszát csak az operációs rendszer korlátozza.

A tizedes jegyek száma legfeljebb 7 lehet.

Idézetben a ↑ és a ; karakterek nem fordulhatnak elő.

Minden TPN programban ↑FORMAT utasitással definiálni kell az INPUT formanevet. Az INPUT forma mindig a táblázás alapját képző file rekordjainak leirását tartalmazza, és meg kell feleljen a ↑FILE utasitásnak.

A ↑FORMAT utasitással definiálhatunk különböző output, gyüjtő és munkarekordokat is.

1.3. A file rekordjaiban bizonyos pozíciók változásának figyelésére, és a bekövetkezett változásnak /kontrol-megszakitásnak/ megfelelő tevékenység inditására szolgál a ↑CONTROL utasitás.

Az utasitás alakja:

↑CONTROL:S_1;S_2; •••; S_k;

ahol S_i p:n,m P program vagy formanév,
n a kontrolmező kezdő pozíciója,
m a kontrolmező hossza.

1.4. A tevékenységek leírására szolgál a ↑PROGRAM utasítás.
Az utasítás alakja:

↑PROGRAM:P;U_1;U_2; ... ;U_k;

ahol P programnév,
U_i A = B C A, B és C mezőnevek,
vagy A = B
vagy A = /az A mező törlése/
vagy F /az F forma nyomtatása/
vagy Q /a Q program futtatása/

1.5. Speciális jelentésü programnevek:

OPEN A file megnyitása után elsőként kerül végrehajtásra. Alkalmas pl a feldolgozás megnevezésének megadására.

BASE A file minden egyes rekordjának beolvasása és a kontrolmegszakitások ellenőrzése után végrehajtásra kerül. A BASE programban kell megadni azokat a tevékenységeket, amelyek minden rekorddal elvégzendők.

PAGE A táblázó program egy lapra legfeljebb 66 sort ir ki. Ha egy forma kiirása nem fér el az aktuális lapon, akkor még a kiirás megkezdése előtt automatikusan bekövetkezik az uj lapra állás, utána végrehajtásra kerül a PAGE program. Természetesen más programból is aktivizálható a PAGE program. Alkalmas az output anyag tördelésére, fejlécek kiirására.

CLOSE Az utolsó rekord feldolgozása után kerül végrehajtásra. Alkalmas a gyüjtött sorok ismételt kinyomtatására, az output anyag lezárására.

1.6. Megjegyzések:

a/ Az első rekord nem okoz kontrolmegszakitást, a file vége minden szempont szerint kontrolmegszakitást okoz. Ha egyszerre több kontrolmegszakitás következik be, akkor először az elsőként felsorolthoz tartozó tevékenységek kerülnek végrehajtásra.

b/ A PAGE utasitás hatástalan, ha a megkezdett lapra még nem történt nyomtatás.

2. A fordító program működése

Az első menet elvégzi a programkártyák beolvasását, ellenőrzi a kártyaazonositókat, a kártyák sorrendjét, szintaktikusan ellenőrzi és compiláló technikával egy közbülső nyelvre forditja a TPN programot. A forditóprogram egy része is ezen a közbülső nyelven van leirva. A második menetben megvizsgálja, hogy nincs-e definiálatlan azonositó a programban, az egyes utasitásokban szereplő azonositók azonos tipusuak-e, és a tipusuk az utasitásnak megfelelő-e. Az igy keletkezett és ellenőrzött programot a harmadik menet interpretáló technikával végrehajtja.

NAGYKERESKEDELMI VÁLLALATOK ADATFELDOLGOZÁSÁRA SZOLGÁLÓ PROGRAMRENDSZER MINSZK-22 SZÁMITOGÉPEN

Matievics István
JATE Kibernetikai Laboratórium

1. Bevezetés

Szegedi nagykereskedelmi vállalatok részéről igény merült fel egy adatfeldolgozási rendszer elkészitésére. Az adatfeldolgozási rendszernek a következő igényeket kellett kielégiteni:

a/ komplett cikkfigyelés
b/ szállitók és vevők figyelése
c/ különböző statisztikák készitése PM, BKM, KSH szabvány szerint

A rendszer filemanipulációs, mátrixmanipulációs, input és output rutinokat tartalmaz. A rutinok megirása szimbolikus kódrendszerben /a gép assembly szintü nyelvén/ történt. A rendszer elkészitésénél fontos volt, hogy a célprogramok futásideje minimális legyen, mivel a felhasználók számára csak igy gazdaságos adatfeldolgozási problémákat megoldani MINSZK-22 számitógépen.

2. File kezelés

Háttérmemóriaként a MINSZK-22 számitógépen csak mágnesszalagok használhatók, ezért a rendszerben csak szekvenciális file-feldolgozás alkalmazható. A file-ok fix hosszuságu rekordokat tartalmaznak. A rekordok filejei integer, real, string skalárok és tömbök.

A filemanipulációs rutinok szervezőrutinok, a konkrét igényeket mindig egy specifikációval (szubrutinhivó sorozat, amellyel aktivizáljuk a szervezőt)

és egy alprogrammal fogalmazzuk meg. A specifikációban adjuk meg az input és output file-ok számát, azonositóját, mágnesszalagbeli helyét, rekordstrukturáját és a file-pufferek hosszát a gyorsmemóriában. Az alprogramban jelöljük ki az aktuális mezőkön végzett müveleteket, az output file-ok rekordjai kialakitásának módját, a rekordváltás szabályait és a file lezárás feltételét. A szervező rutin feladata az alprogramban kijelölt rekordmüveletek végrehajtása (értelmező technikával történik), rekordváltás, puffercsere, file lezárás és a specifikáció értelmezése. A kijelölt rekordmüveletek nem igényelnek külön rekordterületet, mivel ezek végrehajtása a file-pufferben történik. Kis rekordszámu output file-ok esetén megengedett a random elérésü output file használata is. Egy file manipulációnál maximum 4 input és 4 output file használható. File-ok hosszára kikötés nincs, file kezdődhet a mágnesszalag bármely zónáján.

Van olyan filemanipulációs rutin, amelynek outputja egy vagy több gyüjtő (n dimenziós mátrix). Ilyenkor a felhivó sorozat output paraméterei értelemszerüen a gyüjtőre az alprogramon a gyüjtési szabályokra vonatkozik. A gyüjtő mérete tetszőleges, ha nem fér el a kijelölt puffer területen, akkor a file-okból csak többszöri átfutás után alakul ki a mágnesszalagon.

3. Mátrix manipulációs rutinok

A rutinok inputadatai mágnesszalagon elhelyezett n dimenziós mátrixok (elemei vagy integer vagy real tipusuak). A rendszerben a következő mátrixmüveleteket definiáltuk:

a) elemenkénti összeadás

b/ elemenkénti kivonás

c) részsor és részoszlopösszegekből, mátrix képzése

d) részsor és részoszlopösszegek képzése és ezen összegek adjungálása a mátrixhoz

e) mátrix dimenzió redukálás

f) sorokon és oszlopokon végzett müveletek és az eredmények adjungálása a mátrixhoz.

A mátrixok méretére az a kikötés, hogy egy mágnesszalagra ráférjenek. Egy konkrét feladat végrehajtása során általában csak több filemanipuláció és mátrixmanipuláció után jutunk el a kivánt végeredményhez. A programok aktivizálása overlay technikával történik.

4. Output rutinok. Mágnesszalagon elhelyezett file-kat és mátrixokat lehet ezekkel a rutinokkal szélesnyomtatóra irni. A file-kiiró rutin paramétereivel a kiirandó rekordok fieldjeit (nem feltétlenül irunk ki minden fieldet és nem feltétlenül a mágnesszalagon elhelyezett sorrendben) és fieldek formátumát specifikáljuk. Több rekord kerülhet egy sorba és egy rekordot több sorba is lehet irni. Mátrixok nyomtatásánál a nyomtatandó papir méretétől függően meg lehet a mátrixot vizszintesen és függőlegesen bontani. A nyomtató rutin a lapok számozását automatikusan végzi.

A rendszert nem tekintjük lezártnak. Cél, hogy az alprogramok megadása egy célnyelv segitségével történjék. Ebben az esetben a célnyelven felirt programokat egy minikompiláló segitségével alakitanánk át a rutinok számára értelmezhető alakra.

MÁGNESLEMEZES DARABJEGYZÉK LEBONTÓ PROGRAMRENDSZER TERVEZÉSE a SYSTEM 4/50-es SZÁMITÓGÉPRE

dr.Kiss Dénes, Ruszthy Csaba, Gajári Gyula

Magyar Hiradástechnikai Egyesülés
Számitástechnikai és Szervezési Központja

Az előadásunk alapját képező feladat a Magyar Hiradástechnikai Egyesülés egyik tagvállalatának müszaki előkészitési programrendszere. A gyár hiradástechnikai berendezéseket gyártó szerelő jellegü vállalat 5-6000 fős létszámmal. A kezelendő adatmennyiségre jellemző, hogy egyik gyáregységének darabjegyzék állománya megközeliti a 300000 tételt. A rendelkezésre álló System 4/50-es gép 128 Kbyte ferrit, 4 db 7,2 Mbyte-os cserélhető lemeztár és 5 db 20 Mbyte-os mágnesszalag memóriákkal rendelkezik.

A gyár vezetősége fokozatosan óhajtja elérni a számitógépes termelésirányitási szintet és ennek első lépcsőjeként a müszaki előkészités statikus szükségletmegállapitását akarja számitógéppel megoldani. Ezért a távlati elképzelések miatt a mágneslemezre szervezés követelményként adódott.

1. Az adattárolás problémája

A programtervezés első problémája - bármilyen meglepőnek tünik - a gép háttérmemóriájának szük kapacitása volt. Rövid számitással meggyőződhettünk arról, hogy a darabjegyzék állomány master és struktura alapon történő tárolása csak egy gyáregységnél mindhárom adatlemezt /a negyediket System Disc-nek használva/ lefoglalja.

Megoldásként a következő lehetőségek adódtak:

a. Tisztán mágnesszalagos rendszert készitení. Soros feldolgozásnál nem probléma a nagy adatmennyiség, hiszen többkötetes file-ok használhatók, amelyeknek nem kell állandóan on-line lenni. Nagy hátránya viszont, hogy kis file aktivitás esetén is a feldolgozás rendkivül időigényes, s ezzel a gyors naprakészen tartást és a finomprogramozáshoz szükséges kisebb lebontásokat nem tudjuk gazdaságosan megoldani.

b. Felosztani az állományt több lemezes rendszer között. Cserélhető lemeztárakról lévén szó, látszólag logikusnak tünik. Az üzemeltetési nehézségek azonban mindjárt az adatok szétválasztásával kezdődnek. Honnan tudjuk, hogy az egyes gyáregységekhez tartozó alrendszereknek mely adatokra van szükségük? Nyilvánvaló, hogy egyes tételeknek több rendszerben is elő kell fordulniok.

Hogyan oldható meg ezek karbantartása? Honnan tudjuk, hogy az egyik rendszerből kitörölt alkatrész nem él-e tovább egy másik rendszerben?

2. Kombinált rendszer tervezése

Mivel határfeltételként meg akartuk tartani a lemezes rendszer gyorsaságát, ugyanakkor a változtatások egyértelmü nyilvántartását, ezért a következő hibrid rendszer kidolgozására tettünk javaslatot.

a. A gyár törzsállománya mágnesszalagon helyezkedik el. Ez lehetővé teszi az egész gyár adatainak egyszeri és biztonságos tárolását.

b. Az alap törzsfile-okból a gyáregységek gyártási programjának megfelelően időszakosan független lemezes rendszereket generálunk, amelyek csak az adott végtermékhez tartozó anyag, alkatrész és szerelvény adatokat tartalmazzák. /Manuálisan ezt szétválogatni lehetetlennek bizonyult./

c. A finomprogramozás és naprakészen tartás az alrendszerekben történik, s az alap törzsállományt az összegyüjtött változtatásokkal csak nagyobb időszakokban, vagy újabb generálások előtt tartjuk karban.

Az igy vázolt programrendszer felépitése a következőkből áll:

a. Mágnesszalagos előrendszer /alap törzsállomány tárolására/
   - adatbeolvasás és ellenőrzés /kezdő és újabb adatok/
   - komplettség ellenőrzés /hiányzó adatok felfedezése/
   - lebontás /gyártmányok szerint/

b. Mágneslemezes rendszer
   - rendszergenerálás /mágnesszalagos lebontásból/
   - naprakészen tartás /változtatás, beszúrás/
   - lebontás /gyártási programokhoz/.

3. Mágneslemezes rendszer

A lemezes rendszer kialakitásánál a szokásos master és struktura elvet alkalmaztuk. Ennek lényege, hogy minden egyes tétel /rajzszám/ szerepel a Master file-ban. A szerelvényekhez tartozó komponenskapcsolatokat pedig láncolásos technikával a Struktura file tartalmazza. A Master adataihoz a gyors hozzáférést egy index-szekvenciálisan szervezett Index file biztositja.

Az alábbiakban a teljesség igénye nélkül a programtervezés során alkalmazott egy-két érdekesebb megoldást ismertetünk.

3.1 Törzsfile-ok felállitása

A rendszer tervezésénél alapvető szempont volt kevés

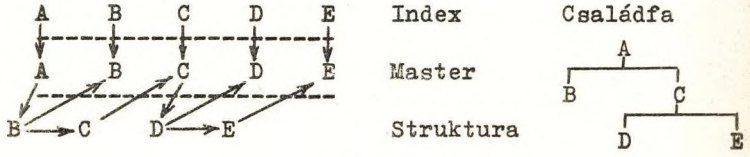
gyártmányra vonatkozó lebontások és egyes tételmódositások futási idejének csökkentése, valamint a változtatások révén bekövetkező file degenerálódások lelassitása.

A fejmozgás és a keresési idő csökkentése céljából a Struktura file feltöltését úgy végezzük, hogy egy darabjegyzék /összetartozó komponens rekordok/ egy cilinderre kerüljön. A felépitő program a következő darabjegyzéket új cilinderre kezdi irni, amig van szabad cilinder, ezután ciklikusan visszatér az első, már részben betöltött cilinderhez.

A lemez első sávján egy "rendelkezésre állási" táblázatot alakitottunk ki, amely minden cilinder első szabad helyének cimét tartalmazza. Cilindereken belül a szabad helyek egymáshoz láncoltak, s minden beszúrás vagy törlés módositja a "rendelkezésre állási" táblázatot.

3.2 Darabjegyzék lebontás szinttáblázat segitségével

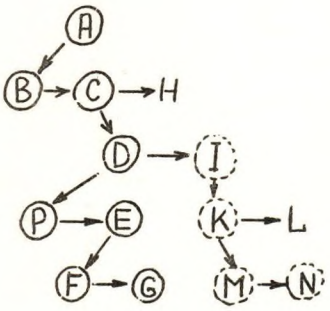
A családfa lebontás elve a következő:
Az inditó gyártmány rajzszámához tartozó rekordot megkeressük az Index file-ben. Ez a rekord közvetlen cimmel megmutatja a tétel Master-beli helyét. A Master-beli rekord /ha bomló/ rámutat a Struktura file-ben elhelyezkedő komponensek közül az elsőre. Az első hivott struktura rekord láncolásos technikával megmutatja a következő komponenst stb. Az utolsó komponenst egy végjel zárja le. /Az adatláncokat az 1. ábra mutatja./



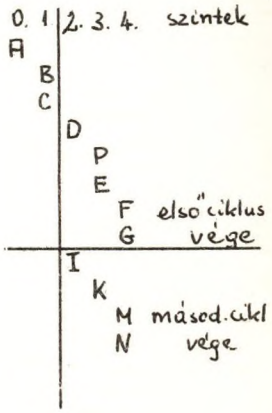
1.ábra

Az ábrából látható, hogy minden struktura rekord tartalmazza az ő master rekordjának cimét is. A master rekordból tudjuk megállapitani /jelen esetben a C szerelvénynél/, hogy további bontás szükséges-e. Több szintes családfa lebontásnál egy szint feldolgozásával mindig megállunk, ha egy tovább bomló alkatrésszel találkozunk, s a bontást az alacsonyabb szinten folytatjuk tovább. /Ez a fogazott lebontás./ A folytatáshoz szükséges cimek tárolását egy szinttáblázattal oldottuk meg. A táblázat minden szinthez egy cimmezőt rendel, amely cimmező minden megállásnál tartalmazza az aktuális szint soronkövetkező komponensének cimét. A visszatérés ezáltal egyértelmüen biztositott. A lebontás menetét és az eredményül kapott fogazott tablót

a 2. ábra szemlélteti. A folytonos karikával jelölt elemeket az első lebontási ciklusban, a szaggatott karikával jelölteket a második ciklusban nyerjük. /A ferdén lefelé mutató nyil Master-Struktura, a vizszintes nyilak a Struktura-Sturktura láncokat mutatják./



szint	cím
0.	*
1.	H
2.	I
3.	*
4.	*



2.ábra

A lebontás egy további segédeszköze egy indexregiszter, amely az éppen feldolgozandó szint szinttáblázatának címére mutat. Az első lebontási ciklus végén /4. szint/ a regiszter visszafelé lép /3. szintre/ itt érzékeli a végjelet /csillag/,vagyis, hogy a szint bontása már befejeződött. Ujabb léptetéssel eljut a 2. szintre, a szinttáblázatban megtalálja a folytatandó elemet /I/ és itt folytatja a lebontást. Az ábra jobb oldalán feltüntettük az első két lebontási út során nyomtatott fogazott tabló szokásos formáját is.

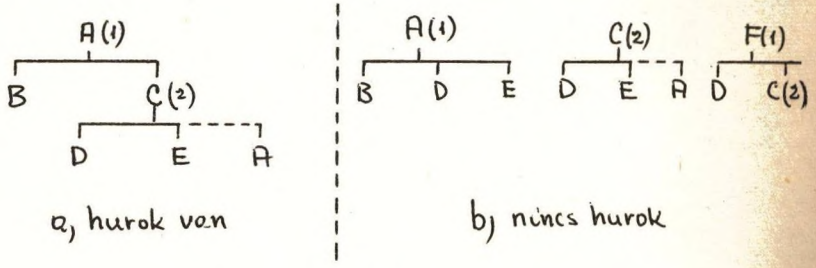
3.3 Hurok kiszürés új darabjegyzék tétel felvitelekor

A darabjegyzék lebontó rendszerek törzs file kialakitásának és karbantartásának talán legnehezebb problémája olyan rekordok kiszürése, amelyek formailag a rendszer számára elfogadhatók, de tartalmilag azt jelentik, hogy egy szerelvény közvetlenül vagy több szinttel lejjebb saját magát tartalmazná komponensként. Ilyen esetben hurok alakul ki, s a lebontás soha nem ér véget. Programrendszerünk ezt a vizsgálatot a master rekordok egyedi jellemzőjének, az un. alacsonyszintü /vagy diszpoziciós/ kódnak segitségével végzi.

Az alacsonyszintü kód az illető gyártmánynak az állományon belüli teljes termék-strukturában elfoglalt legalacsonyabb helyzetét mutatja meg. A kód értékét a master

rekordba a struktura file felállitáskor irjuk be és minden új darabjegyzék rekord felvitelekor karbantartjuk. Uj tétel beépitésekor megvizsgáljuk a rajzszámhoz tartozó master rekord alacsonyszintü kódját. Ha ez nagyobb, mint az őt hivó "apa" rajzszám alacsonyszintü kódja, akkor a felvitellel hurok nem jöhet létre, s a beépitést elvégezhetjük. Kisebb vagy egyenlő kód esetén fennáll a hurokképzés veszélye, s a program belép a hurokvizsgáló algoritmusba.

Az algoritmus a következőképpen müködik /ld. 3. ábrát/.



3.ábra

Tételezzük fel, hogy a 3.a. ábrában az A elemet be akarjuk épiteni a C-be. Az A alacsonyszintü kódja 1, a C-é pedig 2. Az új A-nak ezek szerint a 3-as értéket kell felvennie. Ekkor belép a módositó algoritmus és az A alatt lévő elemek kódját /ha szükséges/ rendre módositja. Az a. és b. ábrákban könnyen követhető, hogy ha hurok áll fenn, akkor a módositás szükségszerüen növeli a C /apa rajzszám/ alacsonyszintü kódját és igy az A tétel a módositás után sem épithető be /a C 4-es értéket kap, az A-nak ezek szerint nem 3-nak, hanem 5-nek kellene lenni./
A b. ábrán nincs hurok. A beépülő A tétel 3-as kódjával csak a B, D és E tételek alacsonyszintü kódjai változnak, s a változtatás után az A beépithető.

Irodalom

1. DIRECT ACCESS TECHNIQUES, ICL Reference Manual, 1970
2. System/360 Bill of Material Processor, IBM programmer's Manual, 1967
3. Siklaky István: Számitógépes müszaki dokumentáció szerelő jellegü iparvállalatoknál, SZÁMOK 1970.

AUTOMATIKUS SCREENING VIZSGÁLATRA ALKALMAS DIAGNOSZTIKAI PROGRAMRENDSZER

Felsővályi Ákos, Dr. Kopp Mária, Dr. Timár Miklós
ORSZÁGOS MUNKAEGÉSZSÉGÜGYI INTÉZET

Az orvos diagnosztikai döntését modellező matematikai módszerek régen ismertek, gyakorlati alkalmazásukat nagy tömegben végzett szűrővizsgálatoknál a gépi adatfeldolgozás tette lehetővé. A számítógépes diagnosztika soha nem helyettesitheti az orvost, a rutindöntések elvégzésével csupán a valójában nem orvosi feladatok alól mentesitheti.
A számítógépes diagnosztika két fő alkalmazási területe:
1. a bonyolult differenciáldiagnosztikai esetek értékelése
2. nagy tömegü vizsgálatoknál a valószinüleg beteg személyek kiválasztása.

Az első területen lényegében máig sem értek el megfelelő gyakorlati eredményeket. Screening-vizsgálatoknál azonban a számítógépes diagnosztikát mint conditio sine qua non-t már ma is sikeresen alkalmazzák. Az Egyesült Államokban 1970-ben 101 screening-rendszer működött.
Az automatikus screening-programrendszer feladata: a részletes klinikai és laboratóriumi orvosi vizsgálatokon alapuló referencia-diagnózisok felhasználásával az egyes betegségek diagnosztikai modelljének a felállitása, mellyel meghatározza, hogy az egyes betegségekre milyen tünetegyüttes milyen valószinüséggel jellemző, majd ennek alapján diagnosztizálja a vizsgált személyeket.

A diagnosztikai modell ismertetése

A bemutatandó modellt, mely elsősorban screening diagnózisra alkalmas, Neumann János közölte 1950-ben.
Előnyei:
1. Nem szükséges, hogy a tünetek függetlenek legyenek egy-

mástól /ha van kölcsönhatás az egyes tünetek között, csak a számolás lesz több, az eredmény ugyanaz/

2. Az egyik betegség felismerése nem akadályozza, hogy más betegséget is megállapitsunk ugyanannál a paciensnél.

3. Nem kell ismerni a betegség előfordulási arányát a vizsgált populációban.

A referencia-diagnózisok alapján minden lehetséges, illetve előforduló tünetkombinációra meghatározzuk az adott tünetegyüttes diagnosztikai értékét, a maximum likelihood hányados segitségével.

$$\Theta = \frac{P_D^S}{P_N^S}$$

A maximum likelihood hányados bizonyos symptoma-együttes /S/ előfordulásának valószinüsége /P_D/ a betegek között /D/, viszonyitva ugyanannak a tünetegyüttesnek előfordulási valószinüségéhez /P_N/ a nem betegek között /N/.
Ez az érték 0-tól /a tünetegyüttes csak nem betegek között fordul elő/ a végtelenig terjedhet /a tünetegyüttes csak betegek között fordul elő/.
N számu bináris tünet esetén a lehetséges tünetkombinációk száma 2^n.
Valamely megbetegedés "diagnózisprofilját" a tünetkombinációk növekvő diagnosztikai érték szerinti rendezésével nyerjük. Tömeges vizsgálatoknál a helyes diagnózis felállitása mindig bizonyos hibahatárok között lehetséges. A screening szenzitivitása a helyes diagnózis valószinüsége a betegek esetében. A szükséges szenzitivitás fokát az orvos határozza meg. A szenzitivitás hibáját Neumann János "elsőfoku" hibának /α/ nevezi. "α" a valóban betegek közül a nem felismert esetek arányának valószinüsége. Minél magasabb a szenzitivitás határértéke, annál kisebb a fel nem ismert betegek aránya, ezzel azonban nő a helytelenül diagnosztizált ese-

tek valószinüsége. /másodfoku hiba, β/. A screening specificitása a helyes diagnózis /negativ diagnózis/ valószinüsége a nem betegek esetében. A senzitivitás és a specificitás ellentétes irányban változnak, ha α értékét csökkentjük, a β értéke emelkedik. β a valójában egészségesek közül betegnek diagnosztizált esetek arányának valószinüsége.

A programrendszer felépitése

A rendszer szubrutinos szerkezetü, az előbb ismertetett feladatrészeket külön szubrutinok hajtják végre. Ez a felépités egyben a rövidebb futási időt is biztositja.
A program egyszerre több betegség diagnosztikai modelljét képes felállitani és egyszerre több betegség szerint tud diagnosztizálni. A tünetegyüttest alkotó tünetek, azok száma, paraméterezve van. A maximálisan figyelembe vehető tünetek száma betegségenként lo, igy a lehetséges tünetkombinációk száma 2^{10} = lo24. /Azért döntöttünk a lo mellett, mert Cochran szerint screening-vizsgálat esetén az optimális tünetszám 8-lo./ Az egységes bázisformátumu adatokat 1 vagy 2 mágnesszalagról tudja fogadni a program.
A betegségek diagnózisprofiljának létrehozása után azok mágnesszalagra kerülnek, igy bármikor rendelkezésünkre állnak. Ugyanezen a szalagon tároljuk a tünetegyüttes tüneteit, igy ezt a diagnosztizálás alkalmával nem kell ujra vezérelni.
Ha időközben az adatok sora bővül /pl. uj felmérés/, akkor a program a tárolt diagnosztikai modellt beolvassa és az uj adatok felhasználásával - ha kell - átalakitja a modellt.

Screening vizsgálat alkalmával a gép diagnosztizálja a vizsgált személyeket. Összehasonlitja az egyes adatokat a kivánt betegség vagy betegségek diagnózis-profiljával, ennek alapján kiválasztja a valószinüleg betegeket.

A senzitivitás fokát az orvos határozza meg, amit a gépi diagnózis alkalmával közölni kell a géppel. A diagnosztizáló szubrutin, miután megállapitotta a diagnózist, a betegségre gyanus személyeknek orvosi vizsgálatra behivó levelet nyomtat /a levél megcimzett és tartalmazza a vizsgálat időpontját is - melyet kivülről lehet vezérelni/, a kezelőorvossal pedig közli a beteg adatait és tünetegyüttesét.

Az adatok és a vezérlés helyességét a program automatikusan ellenőrzi. A program egészének a vezérlése tulajdonképpen egy kártyáról történik. Ezenkivül közölni kell bebetegségenként az egyes tünetek nevét, a gépi diagnózis alkalmával a senzitivitás fokát és a levélben szereplő vizsgálat időpontját.

A diagnosztikai modellhez szükséges adatok és a diagnosztizálandó személyek száma korlátlan lehet. A program FORTRAN F-ben van irva. Hardware igénye: 64 K központi memória, 1 disc egység, 3 mágnesszalagegység, 1-1 sornyomtató és kártyaolvasó.

A programrendszer szervesen kapcsolódik a KERINFORG Rendszertechnikai osztályán kifejlesztett matematikai-statisztikai és adatkezelő rendszerekhez.

Inputja az egységes bázisformátum, igy lehetőség nyilik arra is, hogy a SAMPO adatkezelő rendszerrel adatokat válogassunk, rendezzünk vagy transzformáljunk.

A rendszer felhasználásának lehetősége

A világon jelenleg ezzel a metodikával működő legnagyobb automatikus screening-központ a Collen és Davis által szervezett:"Automated Multiphasic Screening Project", amely 1 millió lakos rendszeres szürését végzi.

A diagnosztikai programrendszert kisérleti célból kipróbáltuk a tatabányai szénbányászok között felvett kb.500.000 adat elemzésével a krónikus nem fertőző léguti betegségek vizsgálatára.

Irodalom

COCHRAN, W.G. and HOPKINS, C.E. Biometrics 1961. 17.1o.

Neumann János First Course in Probability and Statistics New York N.Y. Henry Holt 195o. Chapter V.

COLLEN M.F., RUBIN, L. DAVIS, L. Computers in Multiphasic Screening Computers in Biomedical Research /R.W Stacy B.D Wascman, Eds/ Vol.1. Academic Press, New York 1965. 339-352.

DAVIS L.S. Collen M.F. Rubin L. Van Brunt Computer - Stored Medical Record Com.Biomed.Res. 1968. 1 452-469

JESDINSKY H.J. Diagnose - Modelle in der Medizin Math. Inform Med. 1972. 1. 48-59.

ANDERSON J.A., BOYLE J.A. Computer diagnosis statistical aspects Br. Med. Bul. 1968. 24. 3 23o-235

GLEDHILL V.X and MATHEUS J.D. Computer Aided Diagnosis: A Learning Model J. Med. 197o. 1 249-264

Kiadja: MTESZ
Neumann János Számítógéptudományi Társaság
Felelős kiadó: Jolsvai Károly
Engedélyszám: 96299/1972.
Példányszám: 350
Statisztikai Kiadó Vállalat, Nyomdaüzem - 133872

0535