

BORGULYA ISTVÁN

EGY ALGOL 60 FORDÍTÓPROGRAM

Doktori értekezés

1975

Tartalomjegyzék

Bevezetés	4
1. Fordítóprogramok	5
1.1. A fordítókról általában	5
1.2. Szintaxis-vezérelt compilerek	10
1.3. Compiler-compilerok	14
1.4. Két ALGOL 60 fordítóprogram	17
2. Az EMG-830 ALGOL fordítóprogram felépítése	20
2.1. A program első menete	22
2.1.1. Deklarációk fordítása	22
2.1.1.1. Az azonosító lista	23
2.1.1.2. Relatív címek	25
2.1.1.3. Típus és tömbdeklaráció	26
2.1.1.4. Eljárás-deklaráció	27
2.1.2. Konstansok	28
2.1.3. Címek	28
2.1.4. Kifejezések és utasítások	29
2.1.5. Új lehetőségek	31
2.1.5.1. Statement-változók és kifejezések	32
2.1.5.2. Library-deklaráció	34
2.1.5.3. Gépikódu utasítás	35
2.1.6. Input-output utasítások	36
2.2. A program második menete	37
2.2.1. Tipuskezelés	38
2.2.2. Kifejezések	39
2.2.3. Értékadó utasítás	42
2.2.4. Feltételes kifejezés és utasítás	43

2.2.5.	Ciklusutasítás	44
2.2.6.	Blokkok és deklarációk	45
2.2.7.	Eljárások aktivizálása	48
2.2.8.	Cimkék és a go to -utasítás	50
2.2.9.	Statement-változók és kifejezések	51
2.2.10.	Input-output utasítások	52
2.3.	A program harmadik menete	53
2.4.	A futás szervezése	55
2.4.1.	Blokkok	55
2.4.2.	Tömbök, indexes változók	56
2.4.3.	Eljárások aktivizálása	57
2.4.4.	Eljárások vége	59
2.4.5.	Formális és aktuális paraméterek	59
2.4.6.	Ugró utasítások	60
2.4.7.	Statement-utasítás	61
2.4.8.	Könyvtári eljárás	61
2.5.	Hibaellenőrzés	63
3.	Az EMG-830 ALGOL leírása	65
3.1.	Korlátozások	65
3.2.	Új utasítások ill. bővitések	65
3.3.	Input-output utasítások	67
3.4.	Alapjelek	70
3.5.	A forrás ill. előprogramot jellemző adatok	71
	Irodalomjegyzék	73
	Mellékletek	
1.	Az EMG-830 számítógép utasításai	75
2.	A határolókhoz rendelt prioritási számok	78
3.	A fordító blokkosmái	79
4.	Mintapéldák	
5.	A fordítóprogram listája	

Bevezetés

Jelen munkában egy, az EMG-830 típusu számítógépre irt fordítóprogramot ismertetek. A téma megválasztásában jelentős szerepe volt annak, hogy szakdolgozatomban egy az ALGOL 60 nyelvhez hasonló autokód nyelv fordítóprogramját készítettem el a MINSZK-22 számítógépen. E munka során közelebbről megismertem az ALGOL 60 lehetőségeit, s magát a fordítóprogramírás részleteit, amit érdekes témakörnek találtam. Később folytatva a munkát a MINSZK-22 számítógépen az autokód fordítóprogramnak egy teljes ALGOL 60 fordítóprogrammá váló kiegészítésén dolgoztam. Amikor oktatói munkám során kapcsolatba kerültem az EMG-830-as számítógéppel - nem lévén magasabbszintű programozási nyelvé-, elkezdtem a MINSZK gépen készülő ALGOL fordítót átültetni e gépre.

Mivel a munka során több fordítási technikával megismerkedtem, gyakorlottabban igazodtam el részletkérdésekben, ennek következtében az átírás során egyszerűsítéseket ill. bizonyos bővítéseket tudtam végrehajtani.

Munkám első része áttekintést nyújt az általában elterjedt fordítási technikákról, és két ALGOL fordítóprogramot részletesebben is ismertet.

A második rész az EMG ALGOL 60 fordítóprogram működését és sajátosságait írja le. A leírás alapja a fordítás sorrendje, ami jelen esetben három menetben valósul meg.

A harmadik rész az EMG számítógépen alkalmazható ALGOL 60 nyelvet írja le, amely a bevezetett új lehetőségek következtében bővebb a szabványos ALGOL nyelvnél.

A mellékletben megtaláljuk a fordítóprogram blokksémáit, valamint néhány lefuttatott mintapéldát a fordító lehetőségei és használhatósága szemléltetésére.

1. Fordítóprogramok

1.1. A fordítókról általában

Fordítóprogramok írása a programozáson belül már régóta igen érdekes ágazatnak számít és széleskörű hagyományokkal rendelkezik. Sokan és sokféle fordítót készítettek már, hiszen általában mindenegyres programozási nyelv számára mindenegyres géptípuson külön-külön kell elkészíteni őket.

A fordítóprogramoknak az a feladatuk, hogy a programozási nyelven felírt programokat / forrásprogramokat/ átalakítsák, vagy másképp mondva lefordítsák egy másik programozási nyelvre. Az eredményként kapott programot célprogramnak nevezik.

Mielőtt a fordítók működését részleteznénk, nézzük meg a programozási nyelveket a matematikai nyelvészet szemszögéből.

Egy L programozási nyelv egy véges A halmaz véges szimbólum láncai A^* halmazának valamely részhalmaza. A nyelvben szereplő láncok / a tulajdonképpeni programok/ milyenségét szintaktikai szabályok rendszerével írhatjuk le. E szabályrendszer, vagy másképpen gramatika, vagy olyan algoritmust ír le, amely generálja az L láncait /generatív nyelv modell/, vagy pedig olyan algoritmust, amely el tudja dönteni, hogy egy lánc az L -be tartozik-e /analitikus nyelv-modell/.

A programozási nyelveket általában generatív nyelv-moddal írják le, így a generatív grammatika definícióját nézzük: ($[S], [S]$)

$$G = \langle A, K, R, S \rangle, \text{ ahol}$$

- 1./ A egy véges halmaz - elemei a terminális jelek,
- 2./ K egy véges halmaz - elemei a kategóriák.
/ K és A diszjunktok./
- 3./ R a szabályok véges halmaza.. Bármely szabály

formája a következő:

$$U \rightarrow u, \text{ ahol } U \in K \text{ és } u \in ((K \cup A)^* - \emptyset)$$

4./ $S \in K$, egy megkülönböztetett kategória.

Ahhoz, hogy meg tudjuk mutatni, hogy a grammatika hogyan határoz meg egy nyelvet, további definíciókra van szükség.

Azt mondjuk, hogy valamely $v, w \in (A \cup K)^*$ láncok esetén a v közvetlen deriváltja a w -nek /jelölés: $w \Rightarrow v$ /, ha léteznek olyan - esetleg üres - $x, y \in (A \cup K)^*$ láncok, hogy

$$w = x U y \text{ és } v = x u y,$$

valamint

$$U \rightarrow u.$$

Ha léteznek olyan $w_0, w_1, \dots, w_i$ / $i \geq 0$ / / $A \cup K$ -beli láncok, hogy $w = w_0, w_0 \Rightarrow w_1, \dots, w_{i-1} \Rightarrow w_i$ és $w_i = v$, akkor v -t a w deriváltjának nevezzük, és $w \Rightarrow^* v$ -vel jelöljük.

E definíciók segítségével a G grammatika által meghatározott nyelvet, a következőképp adhatjuk meg:

$$L / G / = \{ X \mid S \Rightarrow^* X, \text{ ahol } X \in A^* \}.$$

E megadás szerint a nyelv mondatait - a programokat - találomra generálhatjuk az S deriválásával. A helyes programoknak pedig az u.n. szemantika megadásával lehet jelentést tulajdonítani.

Ha a programozási nyelvek és a fordítók kapcsolatát vizsgáljuk, két féle fordítót szokás megkülönböztetni: assembler, ill. compiler fordítót. Az assemblerek assembly nyelvről fordítanak gépi kódra, a compilerek pedig magasabb szintű nyelvről assembly nyelvre vagy gépi kódra. A dolgozatban leírt fordító típusából következően csak a compilereket vizsgáljuk, ahol a compiler, a fordító ill. a fordítóprogram fogalmakat szinonimákként használom.

A compilerek - a fordítás folyamatát is alapul véve - további négy csoportra oszthatók: direkt, metanyelvet használó, szintaxis-vezérelt és compiler - compilerek.

A direkt compilerek egy adott L_1 nyelvről közvetlenül egy másik adott L_2 nyelvre fordítanak. Inputjuk egy L_1 nyelvű forrásprogram, outputjuk egy L_2 nyelvű célprogram.

A metanyelvet használó compilerek szintén adott nyelveket használnak, akárcsak a direkt compilerek, de a fordítás folyamatába beiktatnak egy közbülső nyelvet is - egy meta nyelvet -, két részre bontva a fordítás folyamatát. E compiler inputja és outputja azonos a direkt compilerével, de a metanyelv megléte lehetővé teszi, hogy bármely metanyelvre lefordított programot - esetleg különböző nyelvekről - tovább fordítson egyazon nyelvre.

A szintaxis - vezérelt compilerek sokoldalubbak mint az előző típusok. Míg az előzőek egy nyelvről egy nyelvre fordítanak, addig a szintaxis-vezérelt compiler egy nyelvosztály bármelyik nyelvet le tudja fordítani egy másik nyelvosztály bármely nyelvére. Ehhez természetesen meg kell adni a konkrét nyelveket. Így a fordító inputja a forrásprogram mellett a forrásnyelv szintaxisa valamint a célnyelv szintaxisát és a szemantikát tartalmazó transzformációk. E transzformációk megmutatják, hogy a forrásnyelv szintaktikai egységeit mivé kell alakítani, hogy a forrás ill. célprogram szemantikailag azonos legyen. A compiler outputja szintén egy célnyelvű program.

A fordítók legáltalánosabb csoportját a compiler-compiler-ek alkotják. Bizonyos mértékig hasonlítanak a szintaxis-vezérelt compilerekre, de a fordítás eredménye nem célnyelvű program, hanem egy direkt compiler. A compiler-compilernek egy nyelvosztály bármely elemét mint forrásnyelvet és egy nyelvosztály bármely elemét mint célnyelvet kell inputként megadni, s az eredményül kapott direkt com-

píler a forrásnyelvről tud a célnyelvre fordítani. A célnyelv megadása természetesen itt is transzformációk segítségével történik.

A fordítók csoportosításához alapul vehetjük a fordításnak és a végrehajtásnak a kapcsolatát is. Itt azt kell néznünk, hogy a két fázis - a fordítás és a végrehajtás fázisa - szorosan egybefonódva valósul-e meg vagy elkülönül egymástól. /Olyan fordítók is működnek, amelyek e két módszert vegyésen alkalmazzák./

A fázisok egybefonódásán azt értjük, hogy gyakorlatilag nincs külön fordítási és végrehajtási fázis, azaz amint lefordítódik a forrásnyelv egy részlete a célnyelvre, rögtön végre is hajtódik anélkül, hogy megvárná a teljes célprogram kialakítását. Az ilyen felépítésű fordítókat interpretereknek nevezzük.

A másik csoportnál, a compilátoroknál viszont élesen elhatárolódik egymástól a két fázis. Végrehajtásra csak a fordítás teljes befejezése után kerül sor és többször is lehet ismételni a végrehajtást anélkül, hogy a fordítási fázishoz viszonyulhánk.

A két típus közti különbséget Lee ([14]) a következőkben fogalmazta meg: "Amíg egy interpretert úgy definiálhatunk, mint egy közbeiktatott fordítót és végrehajtót, addig a compilátort úgy lehet tekinteni, mint egy igazi fordítót, amelyben a forrásnyelv géptől független, a célnyelv pedig géptől függő".

A fordítókat vizsgálva mindegyiknél felmerülhet a kérdés, hogy a célprogram kialakításához hány menet szükséges. Így beszélünk egy- ill. többmenetes fordítókról. Egymenetesnek nevezzük a fordítót, ha a forrásprogram egyszeri átvizsgálásával kialakítja a célprogramot, többmenetesnek pedig, ha a forrásprogramot többször kell végignézni, vagy a célprogram kialakítását különböző programok kialakítása előzi meg.

További fontos vizsgálati szempont lehet, hogy a fordító a forrásprogram strukturáját milyen módszer alapján elemzi. Mint a nyelv definíciójánál láttuk, az S kategória deriválásával találomra generálhatunk programokat. A fordítóprogramoknál azonban épp fordított a helyzet: egy X program esetén meg kell találni azt a derivációt, amellyel az S-ből megkapjuk az X programot! Ezt a folyamatot nevezzük elemzésnek.

Két elemzési módszer lehetséges: felülről lefelé / top-down / ill. alulról felfelé történő / bottom-up /, e két módszer azonban egyes fordítóknál összekapcsolódik.

A felülről lefelé haladó elemzők célirányosak. A fő cél az $S \xRightarrow{*} X$ deriváció megtalálása, amelyet alárendelt célok kitűzésével keres az elemző. Első lépésként egy $S \rightarrow U_1 U_2 \dots U_n$ szabály alkalmazhatóságát vizsgálja, azaz, hogy az X redukálható-e a jobboldali $U_1 U_2 \dots U_n$ sorozatra.

Ehhez meg kell nézni, hogy az X programlánc megfelelő részei balról jobbra haladva helyettesíthetők-e rendre $U_1, U_2, \dots, U_n$ -nel. Ha valamely U_i -nek nincs megfelelő rész, akkor egy másik $S \rightarrow U'_1 U'_2 \dots U'_n$ szabály alkalmazását kell vizsgálni.

Az U_i alárendelt célok vizsgálata hasonló az előzőekhez: $U_i \rightarrow U_{i1} U_{i2} \dots U_{in}$ alaku szabályok alkalmazhatóságát kell vizsgálni, amelyek ismét alárendelt célok kitűzését váltják ki. Ha valamely alárendelt célnak nincs megfelelő rész a programláncban, egy másik változatot próbál ki az elemző.

Helyes program esetén az elemzés eredményeként közvetlen derivációk sorozatát kapjuk, amelyek segítségével előállítható a célprogram.

Az alulról felfelé haladó elemzőnek nincsenek hosszabb kiterjedésű céljai /kivéve az implicit S célt/. Az elemző a program láncát vizsgálva al-láncokat keres, amelyek

egy szabály jobboldalát képezhetik. Ha ilyet talál, a megfelelő baloldalt helyettesíti a láncba, s a folyamat kezdődik előlről. Egy-egy ilyen lépés során az elemző közvetlen derivációt határoz meg, - így végeredményben ugyanazt a közvetlen deriváció sorozatot eredményezi mint felülről lefelé haladó elemző, csak fordított sorrendben kapja. Mindkét elemző módszer olyan láncokhoz vezethet a működés során, amelyekről kiderülhet, hogy nem felelnek meg a program strukturájának. Ezt a problémát az elemzők különböző képpen kezelhetik. Vagy visszamennek egy olyan pontig, ahol egy másik alternatívát lehet kipróbálni, vagy párhuzamosan vizsgálva az egyes alternatívákat a hibásnak bizonyult változatokat automatikusan kiejtik.

A továbbiakban nézzünk néhány példát a szintaxis-vezérelt és a compiler-compiler fordítókra. A direkt és a metanyelvet használó fordítókra példát az 1. 4. pontban találunk.

1.2. Szintaxis-vezérelt compilerek

Minden fordító leglényegesebb része a strukturáját meghatározó elemző, amely működése alapján sokféle lehet. Ha az elemzési módszereket alapul véve vizsgálnánk a fordítókat, látnánk, hogy a különböző csoportokba tartozó fordítók sokszor ugyanazt az elemzőt használják.

A szintaxis-vezérelt fordítóknak olyan elemzőket kell tartalmazniuk, amelyek az inputként kapott nyelv alapján módosíthatók. Ez egy automatikusan megszerkeszthető elemző létét teszi szükségessé, amelynek előállításához a fordítónak egy algoritmussal kell rendelkeznie. Az itt bemutatott fordítóknál ezen algoritmus - mielőtt kialakítaná az elemzőt - ellenőrzi az inputként kapott szintaktikai szabályokat az egyértelműség szempontjából, majd a szabályokat ill. a lehetséges helyettesítéseket vizsgálva megállapítja, milyen terminális jelek és kate-

góriák kerülhetnek egymás mellé egy szintaktikailag helyesen felírt program elemzése során.

E párok, amiket határoló pároknak nevezhetünk, fogják meghatározni az elemzés folyamatát és segítségünkkel dönti el az elemző, hogy egy szintaktikai szabály alkalmazható-e már, további vizsgálat szükséges-e, vagy hibát kell jelezni.

Az ismertetett fordítókra jellemző, hogy elemzőjük alulról felfelé irányultságú és balról jobbra haladó elemzést alkalmaz a forrásprogram vizsgálatában. Az elemzők egységesen LIFO-vermet használnak az információ sorozatok kezelésére, melyre az jellemző, hogy az utoljára bevitt információ kerülhet ki először a veremből. Az elemzők további jellemzője, hogy az elemzés során mátrixot használnak a párok vizsgálatához. Ezt a mátrixot a nyelvelemző algoritmus állítja elő; a párokról az információkat a mátrix elemek tartalmazzák, a párok mindenegyed baloldali eleméhez egy sort, a jobb-oldalihoz pedig egy oszlopot rendelve.

Az információk megadásában két módszert nézünk meg. Az egyik a párok elemei között relációkat vezet be. E relációk a matematikai relációkkal analógok

/ $a <, >, =$ -nek rendre $\odot, \otimes, \ominus$ felel meg/.

A program elemzése során e relációk alapján keresi meg az elemző az egyes kategóriákat és alkalmazza a fordító a megfelelő transzformációkat a célprogram kialakításához.

A másik módszer lényege, hogy mindenegyed párhoz szubrutint rendelünk, s e szubrutin hajtja végre a párok által meghatározott tevékenységeket. A relációkhoz viszonyítva a szubrutinok használata nagyobb lehetőséget biztosít, mivel az elemzés bármely lépésében beiktathatjuk a transzformációkat is, nem csak akkor, amikor egy kategóriát megtalálunk. A két megoldási lehetőségnek megfelelően a mátrix elemei rendre vagy relációkat vagy szubrutin neveket tartalmaznak. Ha relációkat tartalmaznak, precedencia mátrixnak, különben pedig átmenet mátrixnak szokás nevezni őket.

A relációk módszerét használja Floyd egy speciális, az u.n. operátor precedence nyelvre, melyet az operátor grammatika ad meg [6]. / Az operátor grammatika olyan grammatika, amelyben egy szabály sem veheti fel az $U \rightarrow x U_1 U_2 y$ formát tetszőleges x és y láncok, valamint U, U_1, U_2 kategóriák esetén. / Az ilyen grammatikában a precedencia mátrix meghatározásához elegendő csak a terminális szimbólumok használata, vagyis a határoló párok csak terminális elemekből állnak.

Bármely T_1 és T_2 -vel jelölt terminális szimbólum esetén a következő három reláció határozható meg:

- 1./ $T_1 \circledR T_2$, ha létezik egy $U \rightarrow x T_1 T_2 y$, vagy pedig egy $U \rightarrow x T_1 U_1 T_2 y$ szabály, ahol az U_1 kategória szimbólum.
- 2./ $T_1 \circledR T_2$, ha van egy $U \rightarrow x U_1 T_2 y$ szabály, valamint vagy egy $U_1 \xrightarrow{*} T_1$, vagy pedig egy $U_1 \xrightarrow{*} T_1 U_2$ deriváció valamilyen z és U_2 esetén.
- 3./ $T_1 \circledL T_2$, ha létezik egy $U \rightarrow x T_1 U_1 y$ szabály, vagy egy $U_1 \xrightarrow{*} T_2 z$ vagy $U_1 \xrightarrow{*} U_2 T_2 z$ deriváció valamilyen z és U_2 esetén.

E relációk segítségével könnyen felismerhető egy részlánchról, hogy helyettesíthető-e kategóriával. Ugyanis, ha egy $T_0 X T$ lánc X részláncában a terminális szimbólumok sorrendben $T_1, T_2, \dots, T_n$ és a $T_0 \circledL T_1 \circledR T_2 \circledR \dots \circledR T_n \circledR T$ relációk érvényesek, akkor az X helyettesíthető valamilyen kategóriával.

Egy program elemzése során terminális szimbólumokat viszünk a verembe mindaddig, amíg a $\circledR$ reláció meg nem valósul a ve-

rem tetején lévő és a következő terminális szimbólum között. Ha a program szintaktikailag helyes, akkor egy T_0X alakú lánc van a veremben és megtalálhatjuk X kiinduló pontját. Ekkor, X helyébe a verembe egy kategóriát helyettesítve, megismételhető a folyamat úgy, hogy a T_0 -t vetjük össze a következő terminális szimbólummal.

Floyd módszerének egy továbbfejlesztett változata Wirth és Weber fordítási technikája [24]. Floyd technikája és ez utóbbi között az a legfőbb különbség, hogy az elemzés során bármilyen két szimbólum között fennállhat reláció - nem csupán a terminális szimbólumok között.

Ha egy programozási nyelv fordításánál kívánjuk alkalmazni e technikát, a relációk megállapítása gyakran nem egyértelmű. Ezt vagy a grammatika módosításával vagy pedig a kritikus esetekben kontextustól függően kezelve a relációkat el lehet kerülni.

McKeeman 'extended precedence' módszere [17] több kontextust von be a vizsgálatba. Alapjában véve Wirth módszere kiterjesztésének tekinthető, mert ahol csak lehetséges alkalmazza azt. Ha azonban olyan két elemhez ér, amelyek relációját meghatározni nem lehet egyértelműen, a verem tetején lévő szimbólum mellett, a verem utolsó előtti szimbólumát is bevonja a vizsgálatba.

A precedencia mátrixok kialakítása elég nagy helyet foglal le a memóriában, ezért e módszerek végrehajtásánál néhány esetben a relációkat nem az egyes szimbólumok között értelmezik, hanem a szimbólumokon értelmezett két függvény értékei között. Itt az egyik függvény a határoló pár baloldali eleméhez, a másik függvény jobboldali eleméhez rendel értékeket. Az elemek között értelmezett relációkat ezek után a függvényértékek között fennálló relációk váltják fel.

McKeeman, Horning és Wortmann [18] -ben a módszer újabb változatát adja, amikor az egyes határoló pár elemek között nem relációkat értelmez, hanem logikai értékeket ren-

del az egyes párokhoz. Egy program elemzése során a verembe mindaddig bekerülnek a szimbólumok, amíg a verem tetején lévő, valamint a program következő szimbólumához tartozó logikai érték igaz; hamis esetén pedig kikeresi a megfelelő kategóriát, ami behelyettesíthető a verembe.

Az elemzők leggyakrabban az átmeneti mátrixot alkalmazzák. Ezt az elemzési módszert először Samelson és Bauer vezette be [22].

Az eddigi elemzőkkel szemben az első szembetűnő különbség az, hogy két vermet használ - egy operátor és egy operandusz vermet.

Az elemzés során az operátor verem tetején lévő elem valamint a belépő szimbólum meghatározza az átmenet mátrix egy - egy elemét, amely egy végrehajtandó szubrutin lesz.

Ez a szubrutin hajtja végre azután a szükséges helyettesítést vagy helyezi el a szimbólumot az operátor vagy operandusz verembe. Az elemző az extra operandusz vermet csak az azonosítók könnyebb elkülönítésére használja, az elemzés lényegében véve az operátor precedence módszerre hasonlít.

1. 3. Compiler - compilerek

A fordítóprogramok automatikus készítésének igénye hozta létre a compiler - compilereket, amelyek egyuttal számos fordítóprogram író nyelv készítésére ösztönözték a szakembereket.

E nyelvek fő törekvése, hogy pontos leírást adjanak mind a forrásnyelvről, mind a célnyelvre történő transzformációkról úgy, hogy automatikusan lehessen segítségükkel fordítót generálni, s emellett géptől független nyelvek legyenek.

Egy ilyen nyelvet és compiler - compilert ismertet Feldman a [4] -ben. A fordító számára az input adatokat két

nyelv segítségével kell leírni. A forrásnyelvet egy analitikus nyelvi modellel kell megadni, a célnyelvre történő transzformációk leírását pedig egy formális szemantikai nyelvnek /FSL/ nevezett metanyelven.

Az analitikus nyelvi modell - Floyd fogalmával élve^[25] - produkciónak sorozatából áll. E produkciónak olyan elemző működését adják meg, amely LIFO- vermet használ, s a veremben lévő szimbólumok vizsgálatánál a produkciónak alkalmazza. A produkciónak helyettesítéseket írnak le, megmutatva, hogy a verem legfelső szimbólumait milyen új szimbólumokkal kell helyettesíteni, valamint megadják, hogy melyik produkció alkalmazhatóságát kell következő lépésként vizsgálni. / Ha nem kell helyettesíteni, a produkciónak természetesen hiányzanak az új szimbólumok/. Abban az esetben, ha egy produkció alkalmazását új szimbólum beolvasásának kell követnie, a produkció egy*jelet tartalmaz.

A leírtak alapján egy produkció például:

$$L1 : S_3 S_2 S_1 \mid \rightarrow S'_2 S'_1 \mid * L2$$

Itt L1 a produkció, L2 pedig a következő alkalmazandó produkció címkéje, $S_3 S_2 S_1$ a verem megfelelő szimbólumai, $S'_2 S'_1$ pedig az új helyettesíthető szimbólumok.

A produkció sorozat és az FSL-ben adott transzformációk közötti kapcsolatot "EXEC n" alaku utasítások közlik a fordítóval". Egy ilyen utasítás, amely az n-edik FSL -ben írt eljárás aktivizálását írja elő, hozzákapcsolható bármelyik produkcióhoz.

Maga az FSL egy metanyelv, amelyben nagy számú makroutasítás segítségével felépíthetjük a kívánt transzformációkat. A nyelv használatát megkönnyíti, hogy a mutatókat, rekeszeket, vermeket lehet definiálni, ill. használni.

A létrehozott compiler olyan fordító lesz, amelynek elemző-

je a produkció sorozatra épül és az egyes tevékenységeit a produkciók valamint az EXEC utasítások határozzák meg.

A fordítóprogramok számos változatánál makro-jegyeket próbálnak magasabb szintű nyelvekre alkalmazni. Az egyik változat a compilert speciális assemblernek fogja fel, amelyben pl. a nyelv egyes határolói rendre assembly nyelven írt makro-utasításokat idéznek, míg más változatok a compilert akarják makro-jegyekkel bővíteni.

Compiler-compiler-eknél is találunk ilyen megoldásokat. A legsikerültebb megoldásoknak talán azok tekinthetők, amelyek a szintaktikai leírásba próbálják bevenni a makro-jegyeket. Egy ilyen megoldás pl. a Koster által ismertett CDL ([12], [13]).

A CDL egy fordítóprogram író nyelv, a hozzátartozó fordító compiler - compiler típusu. A CDL - ben elkészített program alapja a forrásprogram szintaktikai leírása. E leírás bővíve van a célprogram kialakításához szükséges transzformációkkal. A bővítések makro-utasítások és a szükséges adatok átadására szolgáló paraméterek, u.n. ragok beépítésével történnek. Makrokat kapcsolhatunk minden kategória után és mind a kategóriákat, mind a makrokat ragokkal láthatjuk el. A teljes CDL programban a szintaktikai leírás mellett definiálni kell a felhasznált makro-utasításokat, az előforduló mutatókat, vektorokat, logikai változókat.

A makrok mellett lehetőség van további tevékenységek definiálására is, amelyeket szabadon elhelyezhetünk a szintaktikai leírásban.

A CDL program alapján a compiler-compiler olyan direkt fordítót készít, amelyben a forrásnyelv mindenegybes kategóriájának egy logikai függvényeljárás felel meg, a makronak ill. a tevékenységeknek pedig eljárások. A ragok az eljárások paramétereiként fognak szerepelni.

A létrehozott elemző egy felülről lefelé haladó elemző, amely

a szintaktikai leírásnak megfelelően, sorra hívja a kategóriákhoz, makrokhoz, tevékenységekhez rendelt eljárásokat.

Egy feltételezett kategória következését a hozzátartozó függvényeljárás dönti el. Az elemzés hatékonyságát az eljárások rekurzív hívása nagy mértékben elősegíti.

1. 4. Két ALGOL 60 fordítóprogram

Mivel a dolgozat célja egy ALGOL 60 fordítóprogram ismertetése, nézzük meg általánosságban az ALGOL fordítóprogramok felépítését, két klasszikus példán keresztül bemutatva.

A Randell és Russell ([20]) által ismertetett ALGOL fordítóprogram egy metanyelvet használó fordító. Elemzője az u.n. prioritási szint technikát alkalmazza, amely speciális változata a precedencia mátrixot függvényekkel helyettesítő elemzőknek. /Az elemző kialakítása nem tartozik a fordító feladatai közé./ Ezen elemző, amennyiben a veremből előkerült szimbólumok sorozatát vizsgáljuk, újra rendezik a forrásprogram kifejezéseit úgy, hogy mindenegyes műveletet megelőzzenek az operandusai.

Például az $a * b * / c + d /$ kifejezésnek a következő felel meg: $a b c d + * + .$

A kifejezéseknek ez a zárójel nélküli jelölése egy változata J. Lukasiqicz lengyel logikatanár jelölésének és Reverse Polish jelölésnek is nevezik.

A fordító által előállított program egy metanyelvi célprogram lesz, amely lényegében véve a Reverse Polish jelölés egy bővített változata.

A fordító a program elemzése, ill. a struktúra vizsgálata során az egyes blokkokban deklarált mennyiségeket egy azonosító listába gyűjti. A listában mindenegyes blokknak külön terület felel meg, amelyet a blokk végén töröl. Mivel az ALGOL-ban lehetőség van arra, hogy egy mennyiség dekla-

rációját a használata megelőzze, a listába a deklaráció előtt használt mennyiségek is bekerülnek. Az előbb használt és később deklarált mennyiségek, valamint a deklarációk közti kapcsolatot a fordító egy címlánc kialakításával oldja meg.

A kialakított célprogram operációk sorozata, amelyben az operációkhoz egy - egy rutin tartozik. A fordítóprogram futatórésze - amely egy interpreter - futás alatt ezeket a rutinokat hajtja végre. A fordító az egyes deklarációknak, a számítás eredményeinek, ill. az egyes paramétereknek futás alatt egy LIFO vermet szervez. Ebben a futás alatt rendre helyet foglal a blokkok egyszerű változóinak és tömbjeinek, tárolja az aktuális paraméterek értékét és jelzőszámokat helyez el az egyes blokkok és eljárások felismerésére. A kifejezések részeredményeit /és a változók értékeit is/ a veremben tárolja. Hogy az interpreter a típusokat figyelembe tudja venni, az egyes értékek mellett a típusokat is tárolja. Hogy a célprogramban előforduló változók ill. paraméterek aktuális címét előkeresse, a fordító egy statikus és egy dinamikus láncot alakít ki, melynek elemeit szintén a veremben helyez el. A statikus lánc az érvényben lévő blokkokhoz és eljárásokhoz tartozó verem kezdőcímeket tartalmazza, a dinamikus lánc pedig sorban a még be nem fejezett blokkokhoz ill. eljárásokhoz tartozókat. / A láncok elemeinek aktuálizálása természetesen egy blokk, vagy eljárás kezdetén és végén történik./

Egy változó aktuális címe így a változóhoz tartozó blokk, deklarációjának sorrendje és a statikus lánc ismeretében megadható. A blokkokhoz és eljárásokhoz rendelt verem a futás alatt nagyban megkönnyíti a dinamikus memória kezelését és a rekurzív eljárások fordítását.

Grau, Hill, Longmaack könyvében leírt ALGOL fordítási technika egy másfajta megoldást mutat be. ([7]). A fordító egy direkt compiler, és a célprogram kialakításához három menet szükséges. Elemzője az átmeneti mátrix technikáját alkalmazza, és a létrehozott célprogram egy fiktív,

egycimes gép assembly nyelvét használja.

A fordító az első menetben a forrásprogram szimbólumait, alapszavait /begin, for, stb./ átkodolja, idézőjelek közé teszi az azonosítókat, és közben feldolgozza a deklarációkat.

A deklarációkból - amelyeket kihagy a módosított programból - azonosító listát épít fel. A lista követi a program strukturáját, megtartva minden deklarációt a további menetre.

A második menet, amely a fordításból kihagyható, a for-utasítások szempontjából nézi át a módosított programot. Ahol csak lehetséges a for-utasítások fordításához ugyanis optimalizációs eljárásokat használ. A harmadik menet a tulajdonképpeni fordítás. Itt egy átmeneti mátrix segítségével elemzi a programot, ellenőrzi a szintaktikai hibákat és létrehozza a célprogramot. A célprogramban az egyes változóknak a neve szerepel, a változók és deklarációk közti kapcsolatot a futás alatt kell megteremteni. Ezen kívül egyes tevékenységek végrehajtására szubrutinhívásokat alkalmaz a célprogram. Ezek a szubrutinok külön állnak a célprogram mellett és a futás alatt dolgoznak. Főbb feladataik közé tartozik a típuskezelés, a memória felosztása, az eljárások formális és aktuális paramétereinek közti kapcsolat megteremtése.

A fő problémát - nevezetesen a rekurzív eljárások használatát - ellentétben a veremmemóriás megoldással, indexregiszterek alkalmazásával oldja meg. Mindenegyres eljáráshoz indexregiszterek segítségével kapcsolja hozzá a megfelelő memóriaterületeket. Természetesen ez a megoldás külön adminisztrációt igényel, de alkalmazása elegendő indexregiszter esetén igen hatásos.

2. Az EMG-830 ALGOL fordítóprogram felépítése

Az előző fejezetben megkíséreltem nagyvonalakban áttekinteni a fordítási technikákat. Az áttekintés nem készült - s az e területen megjelenő munkák mind nagyobb száma miatt - nem is készülhetett a teljesség igényével. Célja az általános ismertetés mellett az is volt, hogy összevetési lehetőség - séget nyújtson az általában alkalmazott és az EMG-830 ALGOL-ban használt technikák között.

A fordítóprogram kialakításánál irányadó volt az a követelmény, hogy minél gyorsabb legyen a fordító működése, s hogy alkalmazkodni tudjon a viszonylag kis tároló kapacitású géphez. Az így kialakított fordító egy három menetes compilátor, amely metanyelvet használ és elemzője pedig a prioritási szint technikát alkalmazza.

A compilátor és interpreter közti választást időtakarékosági szempontok döntötték el. A compilátor mellett szól, hogy az interpreternél a program futását lényegesen lassítja a ciklusutasítások linearizálása, másrészt a gyakorlatban sokszor kívánatos, hogy ugyanazt a programot többször tudjuk használni. Az interpreter fordító esetén mindenegyes használatnál újból kell fordítani a programot, míg egy compilátornál a lefordított programot többször felhasználhatjuk. Ez a tény szintén időmegtakarítást jelenthet.

A fordító készítése szempontjából kényelmetlen tényező, hogy a gép központi tárolója elég kicsi. Egy egymenetes fordító esetén elkerülhetetlen lenne, hogy a fordítóprogram egyes részeit újból és újból a külső memóriából kellene előkeresni; ami nehézkessé tenné a fordító munkáját és a fordítási időt is növelné. Ez az idővesztés kiküszöbölhető, ha olyan önálló részekre tudjuk a programot bontani, amelyek külön - külön, egymást követően hajthatók végre.

A fordítóprogram részekre tagolása a metanyelvet használó compilerknél a legkézenfekvőbb, így erre esett a választásom. Ha szem előtt tartjuk, hogy a rendelkezésre álló tárolóban néhány - a fordítás alatt kialakítandó - táblázatnak is helyet kell biztosítani, az ismertetett elemzési módszerek közül a prioritási szinteket használó kínálkozik alkalmasabbnak; ugyanis a mátrixot használók az előbbieken túlmenően még kb. egy 50 x 60-as mátrix kialakítását teszik szükségessé.

Az e szempontok alapján kialakított fordítóprogram az egyes menetekben a következőket hajtja végre:

Az első menetben a fordító a prioritási szint technikájának felhasználásával átrendezi, ill. átalakítja a forrásprogramot, s egyúttal szintaktikailag is ellenőrzi. A kapott program egy metanyelven írt programnak tekinthető.

A második menetben a metanyelvi programból relatív címes gépkódu programot készít, s a harmadik menetben a relatív címes gépkódu programot abszolút címes gépkódu programmá alakítja.

Ha hasonlóságot keresünk a Randell-Russell féle fordító és az EMC-830 ALGOL fordítója között, azt találjuk, hogy az előbbi és fordítónk azonos elemzési technikát használ, míg azonban az egymenetes Randell-Russell fordító egy operációk sorozatából álló célprogramot alakít ki egy interpreter számára, addig fordítónk az első menetben hasonló alaku sorozattá rendezi át a forrásprogramot /operációk bevezetése nélkül/, és ebből abszolút címes gépkódu programot készít a további menetekben. Ezen abszolút címes gépkódu program a Randell-Russell által alkalmazott futás alatti verem helyett, indexregiszterek segítségével rendel i az egyes memória területeket az eljárásokhoz, hasonlóan a Grau-Hill-Langmaack által ismertetett módszerhez.

Az ismertetett, általában használt technikáktól a fordító főleg részletmegoldásokban tér el. Az eljárások formális

és aktuális paramétereinek kezelésében megvalósított elképzelések révén, valamint újabb utasítások bevezetésével /statement-utasítások, könyvtári eljárások, gépi kódú utasítások/nőtt a fordító hatékonysága.

2.1. A program első menete

A fordítóprogram az első menetben szintaktikailag elemzi ill. ellenőrzi a forrásprogramot és közben egy metanyelvű programná fordítja le. E metanyelvű program tükrözi a forrásprogram strukturáját, az egyes utasítások felépítését, és a kifejezéseket átrendezett formában tartalmazza. E program elvileg megtarthatná a forrásprogramban használt jelöléseket /pl. begin, for +, stb/, de a könnyebb kezelhetőség érdekében ezeket rendre természetes számok jelölik /az azonosítókat cimek/. Így a metanyelvű program formailag egy természetes számokból álló sorozat lesz, s az egyszerűség kedvéért nevezzük el S sorozatnak.

A S sorozat kialakításához az elemző a prioritási szint technikát alkalmazza, melynek használatánál a határolókhoz rendelt prioritási számok - a bővítésektől eltekintve - megegyeznek a [20] -ban ismertetett prioritási számokkal. / Részletesen a 2. számú mellékletben/.

A továbbiakban nézzük a deklarációk és utasítások fordítását külön - külön.

2.1.1. Deklarációk fordítása

A deklarációk fordítása két változatban készült el. Az első változat megengedi, hogy az ALGOL 60 szabályainak megfelelően egyes mennyiségek használata megelőzze deklarációjukat/ilyen eset eljárásörzsek, vagy switch-deklarációk fordításánál fordulhat elő, a másik változat ezt a lehetőséget kizárja, amivel a használhatóságot alig korlátozza.

Mindkét változat egy azonosító listába gyűjti a deklarált

változókat, melynek használata eltérő az egyes változóknál.

2.1.1.1. Az azonosító lista

Az első változatnál a fordító megőrzi a lista tartalmát a második menet végéig. A lista felépítése követi a program strukturáját, és az egyes blokk-kezdeteknek megfelelő belépési pontokat jelző számok mutatják az S sorozatban.

A fordító a deklarált mennyiségeket - a betűk kódját figyelembe véve - a belépési ponttól indulva egy fa-strukturába helyezi el. Minden mennyiség négy szót foglal el a listában. Két szó / hat karakter / az azonosítót, egy az azonosító típusát és egy hozzárendelt címet, egy pedig a fa innen induló ágainak címét tárolja. /Ha a fa valamelyik irányba nem folytatódik, a megfelelő cím nulla. /Pl. az ABCDEF nevű változó elhelyezése a listában a következő:

A	B	C
D	E	F
tipus	cim	
nagyobb	kisebb	
cime	cime	

A típus felvilágosítást nyújt arról, hogy mi az egyes mennyiségek deklarációja. Mindenegyik deklarációnak, specifikációnak külön típus felel meg, és használatnál ebből állapítja meg a fordító, hogy az azonosító, switch, procedure, real, real array, stb. mennyiséget jelöl. A hozzárendelt cím egy relatív cím, amely a harmadik menetben kap végleges értéket.

A lista felhasználása a második menetben történik. Az első

menetben megjelölt azonosítókat a második menetben rendre kikeresi a fordító listából, és típusukkal és relatív címekkel helyettesíti őket a további vizsgálatban.

A második változat leginkább a lista használatában különbözik az elsőtől. Az első menetben ugyanis a fordító minden egyes azonosítót visszakeres a listából s a megfelelő típussal ill. címmel helyettesíti be az S sorozatba. Ez lehetővé teszi, hogy ha egy blokk befejeződik, a blokkban deklarált mennyiségeket törölje a listából. Ez nagy mértékben csökkenti a lista helyfoglalását. Tovább csökkenti az is, ha az első változatnál használt fastruktúra helyett a másodikban csak a deklarációk sorrendjében tároljuk az azonosítókat. Az ebből adódó hosszabb keresési idők nem számottevőek / A fa-struktúra előnye csak nagyobb számú deklaráció alkalmazásánál tűnne ki, mint amekkorát a futtatható programok lehetővé tesznek/.

Az így módosított azonosító listában az egyes mennyiségek három szót foglalnak el. Ezek felépítésben is és funkcióban is megegyeznek az első változatbelivel, csupán a fa-strukturából eredő címek hiányoznak.

Mindkét változat egy vektort használ a blokkokhoz tartozó deklaráció nyilvántartására. A vektor egyes elemei a különböző blokkszintű, még érvényes blokkokhoz tartozó belépési pontokat tartalmazzák. / A blokkstruktúrát a fordító egy blokkszintszámmal jellemez. Ez nulláról indul, és mindenegyes begin-nél eggyel nő, end-nél pedig eggyel csökken az értéke/. A vektor egyes elemeinek beállítása egy blokk kezdetén és végén történik.

A fordító egy azonosítónak mind a deklarálásánál, mind a visszakeresésénél felhasználja a vektort. Deklaráció esetén a blokkszintnek megfelelő belépési ponttól vizsgálja a lista elemeit. Ha már szerepel az azonosító a listában - vagyis újradeklarálás történt -, hibát jelez, ellenkező esetben a legelső szabad helyre helyezi el.

Egy azonosító visszakeresése azonban hosszabb folyamat lehet. Először, akárcsak a deklaráció esetén a blokkszintnek megfelelő belépési ponttól keresi az azonosítót. Ha megtalálta, kiolvassa az adatait és vége a keresésnek. Ha nem, addig csökkenti a blokkszintet, és olvassa végig a megfelelő listarészeket, amíg meg nem találja az azonosítót. Ha a legkülső blokkban sem találja meg, hibát jelez.

A fordítóban jelenleg az azonosító lista második változata van beépítve, mivel ez kevésbé helyigényes és a gyakorlatban épp olyan jól használható, mint az első változat.

2.11.2. Relatív címek

A fordítóprogram [7] -höz hasonlóan a deklarált változókat két felé választja. Az első csoportba azok tartoznak, amelyek helyigénye a futás előtt rögzíthető, a másodikba pedig azok, amelyeké csak a futás alatt derül ki. E csoportosítás szerint az elsőbe tartozik minden egyszerű változó, a másodikba pedig a tömbök /Helyigényük általában csak a futás alatt derül ki/. Ami közös minden tömbnél, s a helyigény szempontjából figyelembe vehető, az információs vektorok hossza. Ez az információs vektor a futás alatt szükséges és a tömbelem-címek kiszámításához szükséges információkat tárolja. A vektor hossza a deklarációból megállapítható, így szintén az első csoportba sorolhatjuk. Ez a csoportosítás mind a főprogramban /nem eljárástörzsben/, mind az egyes eljárásokban deklarált változókra érvényes. A fordító azonban, mivel gépikódu célprogramot állít elő, az első csoportbeli változókat kitüntetetten kezeli, ugyanis a változók címeit, meghatározott helyigényükből adódóan, futás előtt lehet rögzíteni. A rögzíthető helyigény lehetővé teszi a fordítónak, hogy a dinamikus memóriát még futás előtt felossza.

Egy rögzített nagyságu részt biztosít az első csoport számára, a memória többi része pedig marad a második csoport-

nak. E fennmaradó részben foglal helyet az eljárások számára is. Az esetleges rekurziók miatt - nehogy felül írjon még szükséges adatokat - minden eljárás csak aktivizálása során és minden aktivizáláskor újból kap konkrét helyet a dinamikus memóriában. Így az első csoportbeli változók címét csak relative lehet meghatározni, abszolút címeket indexregiszterek segítségével fogja futás alatt kialakítani a fordító. Mivel az eljárások egymásba skatulyázhatók, a deklaráció során mindenegybes eljáráshoz egy eljárás-szintszámot rendel a blokkszintszámhoz hasonlóan. E szintszám az indexregiszterek kijelöléséhez szükséges. Hogy az eljárások változóihoz az indexregisztereket a fordító hozzá tudja rendelni, mindenegybes változó mellé az eljárásszintszám is bekerül az S sorozatba. Ennek megfelelően különböző relativ címeket kapnak a főprogramban és az eljárásokban deklarált első csoportbeli változók. A relativ címek hozzárendelésénél a fordító természetesen figyelembe veszi a blokkstruktúrát is, amennyiben a parallel blokkok ugyanazon relativ címeket használhatják.

Az azonosító listában még kétféle relativ cím található. Az egyik relativ cím a címkék, a switch és procedure nevekhez tartozik. Eltérő kezelésük a többi deklarált mennyiségétől abból adódik, hogy címük nem értéket, hanem egy újabb címet fog jelölni.

A másik relativ cím az own-változókhoz tartozik. Formailag még az első csoportba sorolhatjuk az own-típusu deklarációkat is, mivel azonban ezek értékét a blokkból kilépve meg kell őrizni, célszerű az own-deklarációkat összegyűjteni, és egy, a programot tartalmazó külső blokkban deklaráltnak tekinteni. - Ez a külső blokk feltételezése a címkézett programok használatát is egyértelművé teszi /lásd [11] /. Az own-mennyiségek címének különválasztására az újabb relativ cím alkalmas.

2.1.1.3. Tipus és tömbdeklaráció

A deklarációk feldolgozása során a fordító mindenegybes

mennyiségnél megállapítja, hogy milyen relatív címet használ, és ezt rendeli hozzá az azonosító listában. A megfelelő relatív cím tartományokból az INTEGER és BOOLEAN típusú változók egy-egy címet, a REAL típusúak pedig kettőt foglalnak le. A tömbdeklarációk a típustól és a lehetséges own-tól függetlenül dimenziószám + 2 címet foglalnak le az információs vektoroknak.

Az indexkorlátokban található kifejezéseket az elemző feldolgozza és a végén egy tömbdeklarációt jelző mutatót helyez el az S sorozatban. A jelzőt több paraméter is követi. Ezek megmutatják a tömb típusát, dimenzióját, azt, hogy ugyanazok az indexkorlátok hány tömbre vonatkoznak, és az információs vektor/több tömb esetén az első/kezdőcímét. OWN-tömb esetén, hogy a deklarációk során a tömb mindig ugyanarra a helyre kerüljön, a fordító kiszámítja előre a helyigényét, és lefoglalja a relatív cím tartományból helyüket. A tömb elemei közvetlenül az információs vektor után kerülnek.

2.1.1.4. Eljárás-deklaráció

Mint a 2.1.1.2-ben láttuk, a fordító az eljárásokhoz külön relatív címet rendel. Ezt a címtartományt az egyes eljárás-deklarációk előlről kezdik használni.

Minden eljárásnál egységesen a következő sorrendben oszlanak meg a relatív címek:

- Az első két cím a függvényeljárások értékének van fenntartva. A többi eljárás ezt a részt üresen hagyja.
- Mindenegyik paraméter öt-öt címet foglal le. Ez az öt szó a futás során a paraméterekről fog információt tárolni.
- Ezután az eljárás első csoportbeli változói következnek.

Magát az eljárástörzs utasításait az elemző feldolgozza, és hogy ezeket el lehessen különíteni a többi utasítástól, minden eljárástörzs elejét és végét jelzők mutatják az S sorozatban.

2.1.2. Konstansok

A forrásprogramban előforduló számokat a fordító kettes számrendszerbe konvertálja, és egy táblázatba gyűjti. A táblázatba minden szám egyszer kerül be és a helye meghatároz egy címet. Ezt a címet - mint relatív címet - használja fel a fordító arra, hogy a számot a típusával és relatív címével helyettesítse be az S sorozatba. A programban előforduló idézeteket a fordító szintén a konstans táblázatba gyűjti, s a hozzájuk rendelt relatív címet külön tipusszámmal látja el az S sorozatban. Itt említem meg, hogy a fordító a true és false logikai értékeket az 1 és 0 számértékekkel reprezentálja.

2.1.3. Címkek

Az azonosító lista két változatának megfelelően másként kezeli az egyes változatoknál a fordító a címkeket. Az első változatnál, ha egy címke deklarációhoz ér,/azonosító :/ az azonosítót elhelyezi a listában, relatív címet és típusát pedig az S sorozatban. A második menetben így nem okoznak gondot a postdefinit-címkek, mivel a teljes azonosító lista ismeretében az egyes címkek visszakeresése problémamentes.

A második változatnál nem lehet ilyen egyszerűen megoldani a címke kezelést. Mivel egy belső blokkban használt címkéről nem lehet eldönteni a megfelelő blokkok lezárása, vagy címkek deklarációja előtt, hogy postdefinit-címke-e vagy pedig egy külső blokkban már deklarált címke-re vonatkozik, a fordító minden új blokkban a címke használatot

postdefinitnek tekinti. Így, hogy a fordító nyilvántartassa a deklarált és postdeficit-cimkéket, szintén az azonosító listában helyezi el őket. Ha egy postdefinit-cimke egy külső blokkban deklarált címke is vonatkozhat, a postdefinit-cimke relatív címe értékül a külső blokkbeli címke relatív címét kapja. /Ezt a harmadik menetben használja fel/.

Mivel egy blokk végén az azonosító lista megfelelő része törlődik, a blokkban használt postdefinit-cimkék a címke deklarációja előtt eltűnhetnek. Hogy megőrizzük őket, a fordító mindenegyes postdefinit címket egy postdefinit-cimkelistában is elhelyez. E lista felépítése megegyezik az azonosító listáéval, csupán a típus helyett annak a külső blokknak a szintszáma szerepel, amelyben deklarált címke vonatkozhat. /Ha ilyen még nincs, akkor a szám nulla./

Egy címke deklarációja során a fordító megvizsgálja a postdefinit-cimkelistát a blokkszintnek megfelelő kezdőponttól. Ha a címke történt hivatkozás, a postdefinit-cimkék relatív címeinek értékét beállítja a deklarált címke relatív címére, majd törli őket a listából.

A blokkvége utsítás a megszűnő blokkban található még fel nem dolgozott postdefinit-cimkéket a külső blokk postdefinit címkejéhez csatolja. Ha egy postdefinitnek feltételezett címkeről kiderül azonban, hogy nem az, akkor a blokkvége utasítás törli a cimkelistából.

A postdefinit-cimkelista a hibajelzésre is alkalmas, ugyanis a deklarálatlan címkek a program végén a listában maradnak.

2.1.4. Kifejezések és utasítások

A prioritási szintszámok alapján az elemző egységesen tud vizsgálni minden kifejezést és utasítást.

A kifejezéseket a Revers Polish formára alakítja, amely a kódgenerálás szempontjából nagyon előnyös, az egyes utasításokat pedig a határolókkal és speciális jelek bevezetésével jelzi az S sorozatban. A továbbiakban egy-egy példán keresztül nézzük meg, hogy az utasításoknak milyen alak felel meg az S sorozatban.

Értékadó utasítás:

az $a := b := c$ utasításnak az
 $a, b, c, T := , :=$ alak felel meg.

a " $T :=$ " jelölés azt mutatja, hogy a tárolt eredményt ismét tárolni kell.

Vezérlésátadó utasítás:

a $go\ to\ cinke$ utasításnak a
 $cinke, go\ to$ alak felel meg.

Feltételes utasítás:

az $if\ a < b\ then\ a := b\ else$
 $a := (if\ a = b\ then\ c\ else\ d)$

utasításnak az

$a, b, < , if, a, b, := , then\ S, a,$
 $a, b, =, if, c, then\ E, d, else\ E, else\ S$
alak felel meg.

Itt az S és E betűk a then és az else után azt jelzik, hogy a then és else egy utasításnak vagy kifejezésnek a része.

Ha a feltételes utasítás pl. az

$if\ a > b\ then\ a := b,$ akkor a hozzátartozó S
sorozat $a, b, > , if, a, b, :=, then\ V.$

Itt a then V azt jelzi, hogy az utasításban nem

szerepel else.

Ciklusutasításnál pl a

```
for i := 1 step 1 until n, 2, i + 1 while  
d < loo do D := D + i utasításnak a következő  
S sorozat felel meg:
```

```
for, i, for =, for S, 1, step, 1, until, n,  
for', for A, 2, for', for W, i, 1, +, while,  
d, loo, <, do, D, D, i, +, :=, fr
```

A sorozatban az utasítást a for és fr jelek határolják, a for S, for A és for W pedig a ciklusutasítás egyes alaptípusait jelölik.

Az összetett utasításnál található begin és end határolókat a fordító kihagyja az S sorozatból, de blokkok esetén megtalálhatók a sorozatban is.

Ha az utasítás egy eljárásutasítás, pl.

P /a,b/ akkor a megfelelő S sorozat:

P, EB, a, PE, b, EBE,

Itt az EB és EBE a paraméterlistát határolja, a PE pedig a paraméter elválasztót jelöli.

Az itt ismertetett, az utasítások felírásánál használt határolóknak és speciális jeleknek a második menetben külön - külön rutinok felelnek meg.

2.1.5. Új lehetőségek

Az ALGOL nyelv, amely a használhatóság szempontjából nincs teljesen definiálva, lehetővé teszi, hogy az egyes gépi reprezentánsoknál bővítéseket vezessünk be, kihasználva az adott gép nyújtotta lehetőségeket és szem előtt

tartva a felhasználók igényeit. Itt igények alatt első sorban azokat a kívánságokat értem, amelyek különböző jellegű feladatok megoldása során gyakran fellelnek. Sokszor jelent problémát az, hogy csak korlátozott méretű programok futtathatók, s kívánatos, hogy más program részleteit is beépíthessük az ALGOL programba.

Az ilyen jellegű kívánságoknak tesznek eleget az alább ismertetett lehetőségek.

A program egyszerűsítését, rugalmassága növelését célozza a statement változók és kifejezések bevezetése; a library deklaráció a programméret növelését teszi lehetővé, és gépkódu utasítások alkalmazásával szintén egyszerűsíthető a program.

2.1.5.1. Statement változók és kifejezések

Érdekes és hasznos megállapítást tett Kalmár László az ALGOL nyelv használhatóságával kapcsolatban, amit [10] -ben ismertetett. A megállapítás lényege, hogy az ALGOL nyelv azzal a fogyatékokossággal rendelkezik, hogy nem nyílik lehetőség az utasítások megváltoztatására, amit viszont a számítógépek programozása lehetővé tesz. Ha a számítógép használhatóságának más kérdésben teljes értékűen eleget tud tenni az ALGOL nyelv, célszerű ezt a lehetőséget, az utasítás megváltoztatást is kiaknázni.

Ez az algoritmusok készítése szempontjából is hasznos, hiszen így szabadabban építhető fel az algoritmus és nagyobb programok terjedelme esetleg csökkenthető, egyes részeik menet közbeni változtatásával.

A [10] -ben ötletet is látunk a gyakorlati megvalósításhoz. A megoldást statement változók és kifejezések bevezetése szolgáltatja, melyekkel az ALGOL módosított szintaktikai szabályai a következők lesznek:

$\langle \text{dekларátor} \rangle ::= \text{own} \mid \text{boolean} \mid \text{integer} \mid \text{real} \mid \text{statement}$
 $\text{array} \mid \text{switch} \mid \text{procedure}$

$\langle \text{típus} \rangle ::= \text{real} \mid \text{integer} \mid \text{boolean} \mid \text{statement}$

$\langle \text{kifejezés} \rangle ::= \langle \text{aritmetikai kifejezés} \rangle \mid \langle \text{logikai kif.} \rangle \mid$
 $\langle \text{helymegjelölő kifejezés} \rangle \mid \langle \text{statement kif.} \rangle$

$\langle \text{egyszerű statement kif.} \rangle ::= \langle \text{utasítás} \rangle \mid \langle \text{változó} \rangle \mid$
 $\langle \text{függvénykifejezés} \rangle \mid \langle \text{statement kif.} \rangle$

$\langle \text{statement kif.} \rangle ::= \langle \text{egyszerű statement kif.} \rangle \mid \langle \text{feltételrész} \rangle$
 $\langle \text{egyszerű statement kif.} \text{ else statement kif.} \rangle$

$\langle \text{értékkadó utasítás} \rangle ::= \langle \text{baloldali lista} \rangle \mid \langle \text{aritmetikai kif.} \rangle$
 $\langle \text{baloldali lista} \rangle \mid \langle \text{logikai kif.} \rangle$
 $\langle \text{baloldali lista} \rangle \mid \langle \text{statement kif.} \rangle$

Mint a szintaxisból látszik, használhatunk indexes utasítás-változókat és utasítás értékű eljárásokat is.

Egy statement kifejezésnek az értéke egy cím lesz, amely a kifejezés által meghatározott utasítás programbeli helyét fogja futás közben mutatni.

A statement változó mint egy utasítás is szerepelhet a programban. Hatására az értékének megfelelő utasítást hajtja végre a program.

Ezt a javasolt megoldást jelen fordítóprogram is módosítás nélkül alkalmazza. Beillesztése a szintaxisba nem módosította jelentősen a fordító felépítését.

A fordítóprogram következőképpen fordítja a statement változókat ill. kifejezéseket: az egyszerű statement kifejezéseket /ha nem változók vagy függvény kifejezések/ zárt szubrutinnak fordítja le. A kifejezés értéke

egyenlő a szubrutin kezdő címével. Ezt a kezdő címet a statement változók tartalmazzák. Ha utasításként hivatkozunk rájuk, a futás során az aktuális szubrutint fogja végrehajtani a program.

Hogy a statement kifejezést egyértelműen meghatározza a változó, még egy címet, - a kifejezést tartalmazó eljáráshoz tartozó memória kezdőcímét - tárolnia kell a változónak.

A statement deklarációk fordítása hasonló a típusdeklarációkéhoz. Mivel minden statement változó két címet tárol, egy változóhoz két rekesz kell rendelni. Így a statement típusú változók a deklarációk első csoportjába tartoznak. A statement típusnak megfelelően külön típusjelzőt kellett bevezetni a statement, a statement array és a statement procedurek számára.

A fordító a kifejezéseket az első menetben formálisan vizsgálja meg. Ha egy jobboldali kifejezés vizsgálata során utasításra bukkan, megnézi, hogy a zárójelpár között helyezkedik -e el. Ha nem, akkor hibát jelez, különben az S sorozatban a zárójelek helyére egy statement kifejezés kezdet ill. vége jelet helyez el.

Pl. az $S := /go\ to\ L/$ utasításnak a

$S, ukk, L, go\ to, ukv, :=$ alak felel meg.

Amennyiben egy zárójelpárban nem állhat statement kifejezés, a másik menet típus ellenőrzése felismeri a hibát.

2.1.5.2. Library-deklaráció

Hogy a futtatható programok méretét növeljük, a fordítóprogramok lehetővé teszik a programok szegmentálását. Ezt a lehetőséget próbálja a library-deklaráció megvalósítani, amely lehetővé teszi a gép programkönyvtárában elhelyezett programok használatát. E könyvtári

programok ALGOL eljárások lehetnek, amelyeket a használó programoktól függetlenül önállóan lehet lefordítani. Minden ilyen eljárásra library-procedure vagy pl. a library real procedure és eljárásnév deklarációval hivatkozhatunk.

A fordító e deklarációt is a blokkstrukturának megfelelően kezeli. Egy ilyen eljárás csak abban a blokkban érvényes, amelyben deklarálták. A deklaráció hatására a futtató rendszer /PR/ futás alatt a dinamikus memóriába helyezi el az eljárás utasításait. Mivel a blokkstrukturának megfelelően több eljárás kerül egymás mellé vagy kerül átirásra, a szegmentálási megoldás az overlay-technika egy, az ALGOL-nak megfelelő módosítása. Magára az eljárásra úgy hivatkozhatunk a programon belül, mintha az eljárás az adott programban lenne deklarálva.

A library-deklarációt az első menetben kiszűri a fordító a forrásprogramból, az S sorozatba pedig egy "library" jelzést és az eljárás nevét helyezi el.

2.1.5.3. Gépikódu utasítás

Bizonyos problémákat, feladatrészeket hatékonyabban programozhatunk, ha az ALGOL kötöttségeit kikerülve gépi kód-ban adunk meg néhány utasítást. Bár ez magában rejti annak a veszélyét, hogy nagyobb hibalehetőség, egyes esetekben mégis célszerű a gépi kód használata.

Az EMC gépnek megfelelően egycimes gépi kódu utasításokat írhatunk. Code - szóval kell kezdeni az utasítást, és ebben vesszővel elválasztva több gépi kódu utasítás szerepelhet. Egy gépi kódu utasítás két részből áll:

- műveleti jel
- címrész

A műveleti jel a gép bármelyik utasítására vonatkozhat. A címrészen egy azonosító, egy előjel nélküli szám, relatív vagy abszolút cím állhat. Az azonosító egyszerű vál-

tozó /vagy érték szerint hívott formális paraméter/, címke, vagy tömbnév lehet. Egyszerű változó és címke esetén a megfelelő relatív címet másolja a fordító a műveleti jel mellé, tömbnévnél pedig a tömb első elemét tároló rekesz relatív címét.

Ha számot helyezünk el a címrészen, akkor azt konvertálja, elhelyezi a konstans - táblában, majd a kapott relatív címet másolja a műveleti jel mellé. Ha a cím egy relatív cím / a gépi kódú utasítás helyéhez viszonyítva/, a célprogram kezdetéhez viszonyítva alakítja ki a relatív címet./ Ezt a második menetben csak az utasítás aktuális sorszámaival kell növelni./ Az abszolút címet változtatás nélkül a műveleti jelhez másolja.

A címrészen nem lehet név szerint hívott formális paraméter, switch- vagy procedure-név, és indexes változó sem, mert ezek sorra olyan tevékenységet igényelnek címük meghatározása előtt a fordítótól, amelyek hívó utasításait be kell építeni a célprogramba.

Az így kialakított utasítások az S sorozatba kerülnek. Eléjük egy kódjelző, vagy ha relatív címes utasítás volt, egy "relatív" kódjelző kerül.

2.1.6. Input - output utasítások

Az ALGOL - jelentés nem tartalmazza az input-output utasítások leírását, megvalósításukat az egyes gépi reprezentánsokra bizza. Így ezeknek igen sok változata készült el. 1964-ben megjelent egy olyan közlemény [21], amely általános és egységes input-output utasításokat javasol úgy, hogy minden tipushoz és adathoz külön input, output utasítás tartozik. A fordítóprogram azonban nem követi a cikkben javasolt megoldásokat, mivel felesleges, hogy különböző típusok esetén más és más input utasításokat használjunk. Ez a szétválasztásos megoldás - ha figyelembe vesszük azt, hogy a deklarációk segítségével amugyis megállapítható az utasításban szereplő változó vagy kifejezés típusa -

felesleges.

Másik ellenvetés a javaslattal szemben, hogy egy utasítás mindig csak egy adatra vonatkozik, vagyis nem nyújt csoportosítási lehetőséget. Két másik megoldás ([2] , [3]), melyekhez hasonlóan, de részleteiben nem azonosan készültek el a fordítóprogram input-output utasításai, lehetővé teszik több adat mozgatását egy utasítás segítségével.

A [2] , [3] -ban található megoldások, s a jelen fordítóban alkalmazottak kissé a FORTRAN input-outputjához hasonlítanak. Lényeges eltérés azonban, hogy FORMAT-utasítás nem található bennük.

A fordítóprogramnál minden egyes input utasítás READ szóval kezdődik, amely után tetszőleges változólista állhat. Az output utasításnál pedig a WRITE - szó után állhatnak a kifejezések, vesszővel elválasztva. A WRITE utasításnál az egyes kifejezések után a nyomtatási képre is adhatunk információkat, /az egész- illetve tizedesek számára/.

Fordításuk az első menetben úgy történik, hogy a READ vagy WRITE lista egyes elemei után az S sorozatba bekerül egy jelzőszám, amely a READ, WRITE vagy formátum - tevékenységet jelöli.

Használhatunk továbbá formátumvezérlő utasításokat is, mint a LINE, SPACE - utasítás, vagy mágnesszalag /MT/ illetve mágneslemez /ML/ - kezelő utasítást is. Ezeknek a fordítása a standard függvényekéhez hasonlóan történik oly módon, hogy az S sorozatba az utasításokat ill. függvényeket jelző számok a megfelelő kifejezések vagy paraméterek után következnek.

2.2 A program második menete

A második menet az elsőtől inputként az S sorozatot kapja, amely műveleteket, azonosítókat, utasításokat ill. strukturát mutató számokból áll. E sorozat szekvenciális

vizsgálata során a második menetet egy relatív címes gépi-kódu programmá alakítja a metanyelvű programot.

E menetet leírásánál, hogy az ALGOL nyelvtől ne szakadjunk el, a metanyelvű programelemek fordítása helyett az egyes forrásprogrambéli részeknek megfelelő metanyelvű program-részek fordítását kövessük nyomon.

2.2.1. Tipuskezelés

A fordító írásakor arra törekedtem, hogy minden kifejezést gépi-kódu utasításra fordítsak le, s lehetőleg ne hagyjak típuskezelési problémákat a futásra. Így a fordítás alatt kell eldönteni, hogy az egyes változókat szükséges-e egyrészről - valóssá, vagy valósról egészzé konvertálni, ill. eldől, hogy egyes műveletek fix - vagy lebegőpontos utasításokkal valósíthatók meg. A típuskezelésnek ezt a módját statikus típuskezelésnek hívjuk, ellentétben a dinamikussal, ahol csak futás alatt dől el, hogy egy kifejezésben fix- vagy lebegőpontos utasításokkal fogunk továbbszámolni. A statikus típuskezelés érdekében a [7] -ben adott típuskezelésre javasolt megoldásokra támaszkodtam, apróbb módosításokkal.

A típuskezelés általában problémamentes, így csak azokat az eseteket nézzük pontokba rendezve, ahol többféle megoldás lehetséges.

1. a^n -t $\exp(n \cdot \ln(a))$ -ra fordítja, ha n nem előjel nélküli egész szám, különben n -elmű szorzatot képez.
2. $(\text{if } B \text{ then } E_1 \text{ else } E_2)$ típusa
real, ha feltételes aritmetikai kifejezés,
boolean, ha feltételes logikai kifejezés, és
statement, ha feltételes utasítás-kifejezés.
3. Specifikálatlan változó, indexes változó típusát egy aritmetikai kifejezésben valósnak feltételezi.

4. Specifikálatlan függvényeljárás típusa szintén valós egy aritmetikai kifejezésben.
5. Specifikálatlan baloldali változó és a jobboldali kifejezések típusa közti kapcsolat vizsgálata futás alatt történik.

Mint látjuk, az 5. pontban a típuskezelés csak futás alatt történik meg. Ennek "elodázása" azért nem okoz gyakorlatilag problémát, mert ez az értékadás nem befolyásolja a további típuskezelést a fordítás menetében.

A fordító nem tesz szoros megkötéseket a formális és aktuális paraméterek típusai között. Megengedi, hogy valós típusnak egész, ill. egésznek valós típusu paraméter feleljen meg. Ha a futtatás során ilyen eltérő típusok fordulnak elő, az FR automatikusan a kívánt típusra konvertálja a kifejezéseket. Így az FR méreteit növeljük természetesen, de ez a lehetőség a paraméterek típusának kezelését rugalmasabbá teszi.

2.2.2. Kifejezések

A fordító az S sorozat szekvenciális vizsgálata során kiszűri a sorozatból az azonosítókat. Ezeket előfordulásuk sorrendjében egy operandusz-verembe /OV/ helyezi el. Az OV-ben tehát egy elem egy relatív címét és egy hozzátartozó típust tartalmaz. Amennyiben az S sorozatban műveleti jel következik, a művelet szubrutinja az OV tetejéről előveszi a megfelelő számú /1-2 db/ operanduszt, hogy kialakítsa relatív címük segítségével a gépkódu utasításokat. Az utasítások kialakítása előtt típusellenőrzést hajt végre. Ha a típusok egyenlők, akkor kialakítja a fix vagy lebegőpontos utasításokat, ha nem egyenlők megnézi, hogy azonos típusúvá konvertálható-e az operanduszok. Ha nem, hibát jelez, különben pe-

dig konvertáló utasításokat iktat a kialakítandó gépiódu utasítások elé.

Az ALGOL kifejezések fordításánál általában tárolni kell a kifejezések egyes részeredményeit is. E részeredményeket munkarekesz sorozatban /MR/ helyezi el a fordító, mindig a legelső szabad rekeszt használva fel. Ha a gépiódu utasítások részeredményt állítanak elő, mindig egy tároló utasítás követi őket, amely e munkarekeszbe helyezi a részeredményt. Az OV-ből kikerült operanduszok helyére azután a munkarekesz adatai /tipusa és sorszáma/ kerülnek. Egy kifejezés feldolgozásának végén az OV tetején a kifejezés értékét tartalmazó munkarekesz adatai találhatóak.

Az operanduszok egységes kezeléséből "kilógnak" a tömbelemek és a formális paraméterek, mivel értékük meghatározása csak az FR megfelelő rutinjainak hívásával történhet. Hogy az egyes műveleteknél az operandusz kezelésében ne okozzon ez eltérést, a tömbelemeknek és formális paramétereknek az értékét előfordulásuk alkalmával rögtön egy munkarekeszbe helyezi át a fordító és ez kerül az OV-be.

Ha egy operandusz tömbelem, először az indexkifejezéseket számítja ki, és sorra egymás után következő MR-elemekbe helyezi értékeiket.

Utána egy szubrutinhívó eljárás kerül a programba, amelynek formája a következő:

js	tömbelem értéke
tipus	információs vektor kezdőcíme
dimenzió	az indexkifejezések első címe
-	tároló MR címe

E szubrutinhívás hatására az FR a tömbelem értékét a tároló MR címére helyezi futás alatt.

Hasonlóan a formális paraméterek értékét is egy szubrutin-hívással kapjuk:

js	f-paraméter értéke
tipus	f-paraméter címe
-	tároló MR címe

Az operanduszokhoz hasonlóan a műveletek közt is találunk olyanokat, amelyeket eltérően kell kezelni a többbitől. Ezek azok a műveletek, amelyeknek nincs megfelelőjük a gép utasításkészletében. Ilyenek a reláció, az egyes logikai műveletek és a hatványozás művelete. Ezeket több gépikódu utasítás vagy az FR segítségével oldja meg a fordító.

A reláció operátorokat egységesen kivonásra vezeti vissza. Például az a $\langle b$ relációt, amelynek S-beli alakja a, b, $\langle$ a következő gépikódu utasításokra fordítja:

ra	a	
as	b	
jp	$\equiv + 4$	Az MR_1 munkarekeszbe a reláció logikai értéke kerül / 0 vagy 1 /
ra	1	
wa	MR_1	
ju	$\equiv + 2$	
wz	MR_1	

A logikai műveletek közül csak az AND-nek van gépikódu megfelelője, így a többbit mind ki kell alakítani. Az ekvivalenciát, antivalencia és tagadás segítségével fordítja, a diszjunkciót és implikációt pedig FR-rutinok segítségével. A szubrutinhívás a következő:

js	IMPLIKÁCIÓ /vagy DISZJUNKCIÓ/
-	első operandusz címe
-	második operandusz címe

- tároló MR címe

A hatványozást szintén az FR-rel oldja meg a fordító. A többi művelet fordítása a megszokott módon történik.

2.2.3. Értékadó utasítás

Az értékadó utasítás az OV tetején elhelyezkedő két elem közt alakít ki fix- vagy lebegőpontos tároló utasítást. Az utasítás kialakításánál figyelembeveszi típusait, s ha szükséges /vagy lehetséges/ a baloldal típusára konvertálja a jobboldali kifejezés értékét.

Egyes esetekben a tárolás nem ilyen egyszerű. Ha a baloldal formális paraméter vagy tömbelem, a tárolást ismét csak az FR segítségével tudja végrehajtani.

Formális paraméter esetében a kialakítandó szubrutinhívás a következő:

js	f-paraméternek értékadás
a tárolandó	a tároló f-paraméter címe
típusa	a tárolandó címe

Ha tömbelem a baloldal, akkor először az indexkifejezéseket számítja ki, majd a jobboldal kiszámítása után, egy szubrutint épít a programba, amely az FR-t hívja:

js	tömbelembe tárolás
a tárolandó típusa	az információs vektor kezdőcíme
dimenzió	az indexkifejezések első MR címe
-	a tárolandó címe

/Ha a tömb maga formális paraméter, a szubrutin címe megváltozik és az információs vektor kezdőcíme helyére a for-

mális paraméter címe kerül./

Az értékadó utasítás, a gépkódu utasítás kialakítása után, törli az OV két legfelső elemét.

Ha ugyanazt a jobboldali kifejezést másik baloldali változónak is értékül kell felvenni, akkor az OV tetején levő elemet az eggyel alatta lévő elem helyére tolja a fordító.

2.24. Feltételes kifejezés és utasítás

Egy feltételes kifejezés fordítása a benne szereplő kifejezések típusának vizsgálata szempontjából érdekes. A fordítónak ellenőriznie kell ugyanis, hogy a megfelelő then és else után álló kifejezések azonos típusuak-e, vagy ha nem azonosak, úgy alakítani azokat, hogy a feltételes kifejezés eredménye egyértelműen meghatározott típusu legyen. Ez a statikus típuskezelés szempontjából fontos.

Nézzük a fordító működését egy példán keresztül.

Az /IF L then E_1 else E_2 / kifejezésnek az

L, IF, E_1 , THEN E, E_2 , ELSE E S-sorozat felel meg, ahol L logikai, E_1 feltétlen, E_2 pedig tetszőletes E_1 -gyel azonos típusu kifejezés lehet.

Az IF az L logikai értékétől függő ugró-utasítást alakít ki.

Az E_1 és E_2 kifejezések típusát egy verem /IV/. segítségével vizsgálja a fordító. A then E hatására az E_1 típusát elhelyezi az IV-be. Ha a típus egész, akkor valóssá konvertálja, hogy egységesen lebegőpontos műveletekkel tudjon tovább számolni. / Ha a kifejezésnek egész értékűnek kell lennie, automatikusan visszakonvertálja egészszé. /Az ELSE hatására E_2 típusa bekerül az IV-be. Utána megnézi, hogy az

IV tetején található utolsó két típus azonos-e. Ha nem, hibát jelez. Egyezés esetén az IV utolsó elemét törli. Ezzel a típusellenőrzéssel összetett feltételes kifejezések is vizsgálhatók.

A feltételes utasítás hasonlóan fordítható a feltételes kifejezéshez, ha a típuskezeléstől eltekintünk.

Az utasításban található THEN S - akárcsak a THEN E - hatására ugró utasítást helyez a fordító a programba. Az ugrás címét a THEN S-hez ill. THEN E-hez tartozó ELSE S ill. ELSE E határozza meg. Ha a fordító THEN V- talál az S sorozatban, nem kell újabb ugró utasítást beépítenie.

2.2.5. Ciklusutasítás

A ciklusutasítás fordítása valamivel bonyolultabb feladat mint a feltételes utasítások vagy kifejezések fordítása. Hogy könnyen áttekinthessük, milyen utasításokat generál ciklusutasításoknál a fordító, egy példán keresztül, ALGOL-utasítások felhasználásával szemléljük működését.

Legyen az utasítás a következő:

for v = a step b until c, d do ciklusmag;

A kódgenerálás folyamatát az alábbi ALGOL-utasítások szemléltetik / S egy statement változó./

```
S := /go to L2/;  
L0 : ; S ;  
L1 : v:= m1 ; go to L7;  
L2 : m1 := a ; m2 := b ; S:= /go to L3 / ;  
      go to L4;  
L3 : m2 := b ; m1 := v + m2;  
L4 : m3 := c ;  
      if /m1-m3/ x sign m2 < 0 then go to L1;
```



```
S: = /go to L5 / ; go to L0;  
L5 : m1 := d ; S := /go to L6/ ; go to L1;  
L6 : go to L8 ;  
L7 : ciklusmag ; go to L0 ;  
L8 : ...
```

A fordításnál a statement értékadásnak egy utasítás módosítás felel meg, a statement utasításnak pedig egy ug-róutasítás.

Az L0 címkénél található üres utasítás helyére indexes ciklusváltozó esetén kerülnek utasítások. Ebben az e-setben, hogy a kifejezések kiértékelésének balról jobbra szabályát ne sértsük meg, az index kifejezések kiszámi-tása itt történik.

2.2.5. Blokkok és deklarációk

A blokkok központi helyet foglalnak el az ALGOL programok-ban, mivel segítségükkel lehet az egyes deklarációkhoz tar-tozó memória-területeket kijelölni. Minthogy ez a kijelölés általában futás alatt történik, az egyes blokkok kezdetét és végét /begin, end/ szubrutinhívó utasításokkal kell je-lezni az FR számára.

A blokkokban található típusdeklarációkat a fordítás első menete már kiszűrte. E menetre azonban megmarad még a tömb, switch, könyvtári eljárás, valamint a procedure-deklarációk fordítása.

A tömbdeklarációkat a fordító a korlátpárlista aritmeti-kai kifejezéseinek fordításával kezdi. Mindenegybes kife-jezés értékét a soronkövetkező MR-elemhez rendeli. Ezek a kifejezések az információs vektor kialakításához szük-ségesek, de ennek az előállítása a tömb helyfoglalásával egyidőben történik, futás alatt. Az S sorozatban elhelye-zett paraméterek felhasználásával a tömbdeklaráció végre-

hajtása érdekében szubrutinhívást alakít ki a következőképpen:

js	tömbdeklaráció
tipus	információs vektor kezdőcíme
-	a korlátpárlista első MR címe
atsz	dimenzió

Az atsz az azonos korlátpárlistájú tömbök számát jelöli.

A switch-deklaráció kialakításánál igen lényeges, hogy a kapcsoló kifejezések segítségével könnyen elérhessük őket. Mindenegyik kapcsoló kifejezés egy meghatározott kapcsolólista elemre vonatkozik, amelynek sorszámát - a könnyebb átadás érdekében - indexregiszterbe tölti ki a fordító.

A switch-deklaráció lényegében véve három részből tevődik össze: az első részben történik az indexregiszter vizsgálata. Itt azt kell eldönteni, hogy az indexregiszter értéke nem vonatkozik-e egy definiálatlan értékű kapcsolólista elemre. /Ha igen, akkor a futó program kiugorhat majd egy "visszatérő-utasítás" révén a hívóutasításhoz./

A második rész egy kapcsolótáblázat kialakítása. Futás során a program az indexregiszter értékének megfelelő elemre ugrik a táblázatba. Egy-egy elem a megfelelő sorszámu kapcsolólista elemre való ugró-utasítást tartalmaz.

A harmadik részt a kapcsolólista elemek kialakítása képezi. Az olyan kapcsolólista elemeket, amelyek önmagukban is kapcsoló kifejezések, visszatérő-utasítással zárja le a fordító. Ez azért szükséges, mert ha futás közben nem definiált értékű kapcsoló-elemre hivatkoznánk valahol, vissza kell jutnunk az eredeti ugró-utasításhoz. A visszatérő-utasítás ennek a "visszautnak" a megtételében játszik szerepet.

A könyvtári eljárás deklarációja csak egy jelzés gyakorlatilag arra vonatkozóan, hogy futás közben az adott helyen kell a könyvtári eljárást a dinamikus memóriába elhelyezni.

Ezt az elhelyezést, amit a futtatórendszer fog végrehajtani, egy szubrutin-hívás kialakítás váltja ki majd futás alatt. A szubrutin-hívás a következő:

js	könyvtári eljárás
	a könyvtári eljárás neve
-	
-	

Az eljárásdeklarációban szereplő formális paramétereket az első menet dolgozza fel. Egyedül az érték szerint specifikált formális paraméterek kezelése marad a második menetre is. A fordító az eljárástörzs kialakítása során ezeket a paramétereket már mint lokális változókat kezeli. Azonban, hogy futás alatt a megfelelő értéket felveshessék, egy szubrutin-hívást alakít ki, amelynek segítségével az érték szerint hívott paraméterek aktuális értékeiket a paraméternek kijelölt helyen tárolhatják. Mivel az eljárások futás során indexregiszterek segítségével használják a dinamikus memória megfelelő területeit, a kifejezések fordításánál ezt figyelembe kell venni. Ezért minden egyes MR-elem és az eljárástörzsben deklarált változó, valamint az eljárás formális paraméterei is ugyanazzal az indexregiszterrel módosítva szerepelnek a kialakítandó utasításokban.

Az EMG számítógépen három indexregiszter található. Ez a szám kevés ahhoz, hogy minden egyes eljáráshoz külön indexregisztert rendeljünk, így ugyanazt az indexregisztert használja valamennyi eljárás. Ha egy eljárástörzsben pedig másik eljárásban deklarált paraméterre vagy változóra hivatkozunk, ezek használatát meg kell előznie egy indexregiszter-értéket beállító utasítás kialakításának.

Az eljárástörzs elején a fordító egy információs rekeszt helyez el, amely az eljáráshoz tartozó szintszámot valamint a blokkszintszámot tartalmazza. Ezt a két informá-

ciót a fordító a később ismerttetendő ugróutasítások vizsgálatánál használja fel. Az eljárástörzs végére pedig egy ugró-utasítást helyez a fordító, amely az FR-nek fogja majd az eljárás végét mutatni.

2.2.7. Eljárások aktivizálása

Az eljárások fordítása az egyik legösszetettebb feladat egy ALGOL fordítón belül, ami a gyakorlatban működő ALGOL fordítók hatékonyságát is nagy mértékben meghatározza. A kulcsfontú kérdésként a formális és aktuális paraméterek kapcsolata. A fordítók általában egységesen egy-egy eljárásnak fordítják le az aktuális paramétereket, s így egy névszerű paraméter előfordulása mindig az éppen futó eljárás megszakítását, valamint a régebbi eljárás /esetleg főprogram/ elővételét teszi szükségessé.

Fordítónk ezzel szemben úgy próbálja megoldani a paraméter kezelést, hogy ahol csak lehetséges ne kelljen a futó eljárást félbeszakítani, s egy még be nem fejezett eljárást elővenni. Ha az aktuális paraméter kifejezés, vagy indexes változó, más megoldást nem választhatunk, hiszen a kifejezések értékét csak a kifejezést tartalmazó eljárás /főprogram/ elővétele után számíthatja ki a futó program.

Egyszerű változók, tömbök, eljárások, kapcsolók, konstansok, idézetek esetén azonban nem szükséges megszakítani az eljárást ha futás során a program rendelkezésére állnak aktuális címeik. Éppen ezért a fordító olyan gépkódu utasításokat generál, amelyek segítségével a futó program ki tudja majd alakítani az aktuális címeket, ill. a szükséges információkat.

Az első menetben láttuk az eljárásdeklarációnál a paraméterek helyfoglalását. Futás során a fenttartott

helyeken alakítja ki a program a paraméterek szükséges információit. A fenntartott öt helyből az aktuális paraméterek egy csoportja csak kettőt használ fel /kivé-
tel a kapcsolónév, amely hármat/ az alábbi módon:

js	FR "aktuális paraméter" címe
tipus	aktuális paraméter címe
/-	az eljáráshoz tartozó memóriarész kezdőcíme/

Ha az aktuális paraméter névszerű paraméter, adatai a futó programnak nem szükségesek. Ekkor a paraméternek fenntartott öt helyből a célprogram csak egyet használ fel. E hely futás alatt az előző aktuális paraméter cí-
mére való ugró-utasítást fog tartalmazni.

Kifejezés és indexes változó esetén az aktuális paramé-
ternél található kifejezést vagy indexkifejezéseket a fordító egy szubrutinba foglalja, amely szubrutint az FR hívhat meg. A meghíváshoz szükséges információkat szintén a paraméterhelyeken alakítja ki a fordító.

Az információ sorozat kifejezésnél:

js	FR "aktuális paraméter" címe
a kif.	-
tipusa	a kifejezés szubrutinjának címe
-	az eljáráshoz tartozó memória kez- dőcím

Indexes változónál:

js	FR "aktuális paraméter" címe
tipusa	az indexkifejezés szubrutinjának címe
-	az eljáráshoz tartozó memóriarész kezdőcíme
dimenzió	az indexkifejezéshez tartozó első

MR címe

információs vektor kezdőcíme

Az információs sorozatoknál található szubrutinhívás, amely a futtatórendszer aktuális paramétercímére szól, a névszerinti paraméterek miatt szükséges. Ugyanis futás során - ha egy aktuális paraméter névszerinti paraméter - a formális paraméter címére kiadott ugró-utasítás, ugró-utasítások láncává alakul mindaddig, amíg egy az FR-hez szóló szubrutinhíváshoz nem ér. E hívás a megfelelő aktuális paraméter információ-sorának kezdőcímét fogja szolgáltatni, mely sorozat kiértékelésére már a futtatórendszerre tartozik.

Az aktuális paraméterek vizsgálata után a fordító eljárás-aktivizáló utasítást alakít ki. Ez szintén szubrutinhívás:

js
típus

eljárásaktivizálás
eljárás kezdőcíme

Ha az eljárásaktivizálás egy függvényeljárás aktivizálását szolgálja, a fordítást egy értékadó utasítás zárja. Ez az értékadás a függvénykifejezés értékét tárolja egy MR-elembe.

2.2.8. Címkék és a go to - utasítás

A címkék és a go to utasítások fordítása a második menetben már egységesen történik. Fordításuknál arra kell ügyelni, hogy ha egy címke-re való hivatkozás kivezet a blokkból, futás során a blokkhoz tartozó memória területet meg kell szüntetni. Hogy futás során e szempontból ellenőrizni lehessen a go to-utasításokat, a fordító az FR számára mindenegyes címke-deklarációt szubrutinhívásként fordít, melynek paraméterei a címkét tartalmazó eljárásnak és blokknak a szintszáma. Futás

alatt a paraméterek és a go to-utasításhoz tartozó szint-számok összehasonlításával történik az ellenőrzés. Minden címke-deklarációnak a szubrutinhívás generálása mellett a címkekre tartozó relatív cím értékül a következő gépi-kódu utasítás címét kapja. Így a relatív címes célprogram mellett - a címkek relatív címének - egy táblázatot készít a második menet a címkek abszolút címeinek meghatározásához.

Magukat a go to-utasításokat ugró utasításokra fordítja a fordító, kivéve a kapcsoló-elemre történő ugrásokat, melyeknél szubrutinhívást generál. A megkülönböztetés azért szükséges, mert definiálatlan kapcsolóelem esetén a go to-utasítás utáni utasításra kell adni a vezérlést /üres utasítás lévén/.

2.2.9. Statement-változók és kifejezések

A statement-kifejezések fordítását a második menetben az S sorozatbeli uv és uw jelek jelzik. Hatásukra az uv és uw közt elhelyezkedő utasításokat szubrutinba zárja a fordító. A kifejezés "értékét", a két címet /a szubrutin kezdőcímét és a hozzá tartozó eljárás dinamikus memória területének kezdőcímét/ pedig egy-egy MR-elem kapja értékül. A fordító a következő formában generálja az utasításokat:

wk	MR ₁	
ra	$\mu-1$	/Az eljáráshoz tartozó dinamikus
wa	MR ₂	memória terület kezdőcíme futás
ju	ν	alatt a k indexregiszterben talál-
dn	μ	ható./
μ : dn	0	
	. utasítások	
ju	μ n	
ν :		
	folytatás	

Ha a programban egy statement-változó mint utasítás fordul elő, a fordító egy szubrutinhívást generál, amelynek egyetlen paramétere a változó címe.

2.2.10. Input-output utasítások

A fordító az input-output utasításokat egységesen az FR segítségével oldja meg. Az egyes utasításoknak, a bennük szereplő kifejezésektől vagy változóktól függően, más-más szubrutinhívó gépkódu utasításokat generál. Így különböző szubrutin kell a következő tevékenységekhez:

- egész-ill. valós szám beolvasás
- tömbelem beolvasás
- egész ill. valós tömb beolvasás
- specifikált formális paraméter beolvasás
- szöveg beolvasás
- egész ill. valós szám kiírás
- szöveg kiírás
- mágnesszalag, mágneslemez kezelés
- periféria kijelölés
- soremelés, szóközök kitétele

Az egész ill. valós számok kiírásánál lehetőség van a formátum megadására, amely formátum több szám kiírására is érvényes lehet. Hogy mindig ugyanarról a helyről vehesse elő a futtatórendszer a formátumot jelző értékeket, a fordító két rekeszt jelöl ki az FR-ben azzal a céllal, hogy az egyik az egész jegyek számát, a másik pedig a tizedesekét jelölje. Ha a programban formátum-jelző kifejezéseket talál a fordító, olyan gépkódu utasításokat generál, amelyek a kifejezések értékét e két rekeszben helyezik el.

2.3. A program harmadik menete

A fordítóprogram harmadik menete terjedelmében alatta marad az előző kettőnek, fontosságában azonban egyenrangú velük. E menet során alakul át a relatív címes gépikódu célprogram abszolút címes programmá, itt történik a tár felosztása és végül kész állapotban kapjuk a célprogramot.

A program elhelyezésével az EMG számítógép tárfelosztását veszi alapul. Az EMG gép két tárrésze közül az elsőbe a célprogram utasításai kerülnek, amelyeknek végéhez a fordító hozzácsolja a konstansok táblázatát, a második tárrészben pedig a dinamikus memória kap helyet. A dinamikus memória legelején helyet foglal a főprogram relatív címes változóinak, valamint a második menetben használt munkarekeszeknek. E helyfoglalást a felhasznált relatív cím tartományok alapján határozza meg. Ez után kezdi a fordító a relatív címes gépikódu utasításokat abszolút címesekké alakítani. Minden egyes relatív címnél kiszámítja a nekik megfelelő abszolút címet, majd rendre átírja a gépikódu utasításokat.

A címek relatív címei azért érdemelnek külön említést, mert a megfelelő címek kiszámításához - indirekt voltak miatt - a második menetben kialakított táblázatot is igénybe kell venni.

Ha a célprogram átalakítása megtörtént, a fordító a célprogramot a memória első tárrészébe másolja, majd behívja az FR-t. Ezután elindítható a futtatás.

Könyvtári eljárások fordítása esetén ügyelni kell arra, hogy futás alatt a rendszer információkat igényel a program terjedelmére és a memória foglalásra vonatkozó-

an. Ezen információkat a kialakított célprogram elejére helyezi el.

2.4. A futás szervezése

A harmadik menet során futásra kész állapotba került a célprogram. Mint a második menet ismertetésében láttuk, számos tevékenységet a futás alatt az FR segítségével kíván a fordító megoldani. Ezen tevékenységnek rendre egy-egy szubrutin felel meg az FR-ben. E szubrutinok legtöbbje azt a célt szolgálja, hogy a program terjedelme ne nőjön túl nagyra, és vannak köztük olyanok is, amelyek "adminisztrációs" feladatokat látnak el a program futása alatt.

A terjedelem csökkentését szolgálják olyan szubrutinok, mint pl. az egyes műveleteknek / hatványozás, logikai műveletek / megfelelő szubrutinok, vagy az input-output utasítások rutinjai. Ez utóbbiak tevékenysége a gép periféria kezelő rendszerével való kapcsolat megteremtésében merül ki.

A szubrutinok másik csoportja a blokkok és deklarációk, valamint eljárások adminisztratív jellegű információit / pl. a dinamikus memória szervezése, a be nem fejezett eljárások adatainak nyilvántartása, stb./ tárolja. Feladatai közé tartozik a formális és aktuális paraméterek közötti kapcsolat megteremtése, továbbá a go to-utasítások ellenőrzése is.

E tevékenységek közül nézzünk néhányat részletesen!

2.4.1. Blokkok

A fordító a futás alatt az egyes blokkokhoz tartozó dinamikus memóriaterületeket két rekesz, DM_1 és DM_2 , valamint a hozzájuk tartozó, a dinamikus memóriában lefoglalt rekeszsorozatok segítségével tartja nyilván. A DM_1 a dinamikus memória legelső szabad helyét mutatja, a DM_2

pedig az utolsó beginnél talált első szabad helyet tárolja.

Ha a fordító egy begin utasításhoz ér, az FR a következő lépéseket végzi el:

$$(DM_2) \Rightarrow ((DM_1))$$

$$(DM_1) \Rightarrow (DM_2)$$

$$(DM_1) + 1 \Rightarrow (DM_1)$$

END utasításoknál pedig:

$$(DM_2) \Rightarrow (DM_1)$$

$$((DM_2)) \Rightarrow (DM_2)$$

Hogy az FR a go to-utasításokat is ellenőrizni tudja, nyilvántartja az egyes blokkszintszámokat is. E blokkszintszámok megegyeznek a fordítás alatti blokkszintszámokkal.

2.4.2. Tömbök, indexes változók

A tömbök helyfoglalását valamint az indexes változókhoz tartozó címeket az FR rutinjaival számítja ki a program. Mind a helyfoglaláshoz, mind a címkszámításhoz a [20] -ben található, következő algoritmust használja:

Legyen A egy tetszőleges tömbdeklaráció:

$$\text{array } A \left[l_0 : u_0, l_1 : u_1, \dots, l_{n-1} : u_{n-1} \right] ;$$

amelynek valamely elemére az

$$A \left[s_0, s_1, \dots, s_{n-1} \right] \text{ indexes kifejezéssel hivatkozunk.}$$

Képezzük a következő sorozatot:

$$c_0 = 1$$

$$c_i = c_{i-1} \cdot (u_{i-1} - l_{i-1} + 1) \quad i = 1, \dots, n-1$$

Ha ismerjük a tömb $A[0, \dots, 0]$ elemének címét $/b/$, minden további elem címe a c_i -k segítségével a következő módon számítható ki:

$$\text{cim} = b + s_0 \cdot c_0 + s_1 \cdot c_1 + \dots + s_{n-1} \cdot c_{n-1}$$

Általában nem a b címe ismert, hanem az a w cím, amelytől a tömb elemeit elhelyezi a fordító. Ebben az esetben a b a következő képlettel számítható ki:

$$b = w - / l_0 \cdot c_0 + l_1 + \dots + l_{n-1} \cdot c_{n-1} /$$

Az FR mindenegybes tömb deklarációjánál kiszámítja a megfelelő $c_0, c_1, \dots, c_n$, b sorozatot. A helyfoglaláshoz a c_n ismerete elegendő, a címkiszámításhoz pedig kialakítja az információs vektort. Az információs vektor dimenziószám+2 elemből áll, s elemei rendre a következőket tartalmazzák:

$$w, \text{ dimenzió és } c_n, c_1, c_2, \dots, c_{n-1}, b$$

2.4.3. Eljárások aktivizálása

Hogy a gép által lehetővé tett indirekt címzést kihasználhassuk, a program futása alatt előforduló információkat egy állandó, rögzített területen tároljuk, amit munkaterületnek $/MT/$ fogunk nevezni. E területen helyezkednek el azok az információk is, amelyek egy eljárás futásához ill. aktivizálásához szükségesek.

Az MT következtében az eljárás aktivizálása könnyebben kezelhető, mert a csoportba gyűjtött információk szükség esetén könnyebben áthelyezhetők más területre.

Egy eljárás aktivizálása 3 lépésből áll:

- 1./ az éppen futó eljárás félbeszakítása,
- 2./ az új eljárás jellemző adatainak megkeresése,
- 3./ az új eljárás futásának elindítása.

Az éppen futó eljárás félbeszakítása úgy történik, hogy az eljárás jellemzőit /amik az MT-ben találhatók/ elraktározzuk a dinamikus memóriába, oly módon, hogy azok szükség esetén újból elővehetők legyenek.

Az eljárást jellemző információk: az eljárás szintszáma, blokkszintszáma, a dinamikus memóriára vonatkozó adatok / DM_1 , DM_2 /, az eljáráshoz rendelt indexregiszter értéke, az aktivizált, de még be nem fejezett eljárásokhoz tartozó indexregiszter-értékek, az előző eljárásnál kimentett adatok helye, valamint a visszatérés címe. Az eljárás félbeszakítását kiválthatja valamilyen szubrutinhívás is, így a szubrutin paramétereit / melyek még nincsenek kiértékelve / szintén ki kell menteni, azaz az MT-ben kell tárolni.

A második lépésnél az MT-be a kimentett információk helyére az új eljárás adatai kerülnek.

Az új adatok tárolását a következő sorrendben végzi az FR:

- rögzíti az első lépésben kimentett adatok új helyének címét;
- feljegyzí az első lépésben félbeszakított eljáráshoz tartozó indexregiszter értékét;
- kijelöli az indexregiszter értéket;
- helyet foglal az első csoportba tartozó

- deklarációk számára / ld. első menet / ;
- beállítja az új eljárás és blokk szintszámot;
 - tárolja a visszatérés címét.

A harmadik lépés előtt - amennyiben az aktivizálandó eljárás formális paraméter - bővül a második lépés az aktuális eljárás megkeresésével.

2.4.4. Eljárás vége

Hogyha a futás az eljárás végét jelző utasításhoz ér, az FR előveszi a legutoljára félbeszakított eljárás adatait, és visszahelyezi őket az MT-be. Utána beállítja az előző indexregiszter értéket, majd a program a befejezett eljárás visszatérési címére adja a vezérlést.

2.4.5. Formális és aktuális paraméterek

Mint a második menetben láttuk, az aktuális paraméterek információit a futó program helyezi el a formális paramétereknek fenntartott helyeken. Ha valamely formális paraméter értékére vagy címére szükség van futás közben, akkor az FR keresi meg az aktuális paramétert.

Ez a megkeresés egy a formális paraméter címére való ugró-utasítás kiadásával kezdődik.

Ha az ugró utasítás, vagy az ugró utasítások láncolata az aktuális paraméter adatához ér, az ott lévő szubrutinhívás segítségével az FR megszerzi az adatok elhelyezésének címét. Az adatok között megtalálja az aktuális paraméter típusát és a paraméterként szereplő változó címét. Ha az aktuális paraméter kifejezés vagy indexes változó, akkor az aktuális paramétert tartalmazó eljárást újból aktivizálni kell. Ez az eljárásaktivizálás azonban különbözik némileg a 2.4.3-ban ismertetett ak-

tívizálástól. A különbség az aktivizálás 2. lépésétől kezdődik, mivel ebben az esetben egy már megkezdett eljárás adatait kell kikeresni a dinamikus memóriából és az ezen eljárástömbben található kifejezés végrehajtását kell megkezdeni.

A kifejezés kiértékelése után az aktuális eljárás visszakapja a vezérlést.

Az FR az aktuális és formális paraméter-párok típusait is ellenőrzi futás közben. Ha nem egyenlők a típusok, szükség szerint azonos típusúra konvertálja azokat vagy pedig hibát jelez.

2.4.6. Ugró-utasítások

Az FR egyik fontos feladata, hogy a programban előforduló ugró-utasításokat ellenőrizze. Erre azért van szükség, mert az ugró-utasítások az egyes blokkok és eljárások érvényességét megszüntethetik. Ez az ellenőrzés a címkedeklarációknál történik. Itt összehasonlítja az ugró-utasításhoz valamint a címkehez tartozó eljárás - majd blokkszintszámokat. Ha az eljárásszintszámok nem egyenlők, előkerekíti a még be nem fejezett eljárások közül a címkehez tartozó adatait.

A keresés alatt a nem megfelelő eljárások adatait természetesen törli a dinamikus memóriából.

A blokkszintszámok különbözősége esetén az érvényüket veszített blokkokhoz tartozó dinamikus memória területeket felszabadítja.

Az ugró-utasítások ilyen formában való kezelése eléggé differenciálatlan, olyan esetben is elvégzi az ellenőrzést, amikor az ugró-utasítás, s a hozzátartozó címke azonos blokkban található. Ezt azonban mégis megengedhetjük a

fordító egyszerűbb, egységes kezelése érdekében.

2.4.7. Statement-utasítás

A statement változókhoz tartozó kifejezéseket az FR a változók értékét képező két cím - a kifejezés címe, ill. a kifejezést tartalmazó eljáráshoz tartozó dinamikus memória terület kezdő címe - alapján keresi meg.

A két cím közül az FR először a másodikat vizsgálja meg. Megállapítja, hogy a kérdéses statement kifejezés mely eljárás törzsében volt definiálva. Szükség esetén félbeszakítja az éppen futó eljárást, majd aktivizálja a kifejezést tartalmazó eljárást. Ezen aktivizálás módja megegyezik az aktuális paramétereknél alkalmazott eljárás-aktivizálással. Miután az FR megállapítja, hogy hová tartozik a kifejezés, mint szubrutint hívja.

A kifejezés kiértékelése után az FR visszaadja a vezérlést a statementet követő utasításnak.

2.4.8. Könyvtári eljárás

Könyvtári eljárás deklarációnál az FR működését a következőkben foglalhatjuk össze:

- 1./ az eljárás elhelyezése a dinamikus memóriába,
- 2./ az eljárás utasításainak átcimzése,
- 3./ kapcsolatteremtés a deklaráció és elhelyezés között.

Ez utóbbi úgy történik, hogy az FR kialakít egy az eljárásra ugró utasítást, amelyet közvetlenül a deklaráció után helyez el a célprogramban. Ily módon a futó program az eljárástörzs első utasítását a deklaráció után

találja meg.

Hogy többszöri deklaráció esetén ez az első utasítás ne kerüljön felülírásra ill., hogy ugyanazok az utasítások ne kerüljenek be többször a dinamikus memóriába, az FR számontartja az érvényes könyvtári eljárások címét és nevét is.

2.5. HibaelLENŐRZÉS

Egy fordítóprogram megírásánál mindig lényeges követelmény, hogy a fordító a forrásprogramban fellelhető szintaktikai hibákat fel tudja ismerni. E feladat megvalósítására számos megoldás született. Az egyik legtökéletesebb megoldásnak a program fordítás előtti szintaktikai ellenőrzése látszik, azonban számos fordító összeköti a hibaelőrzést az elemzéssel. Ha összekötjük az elemzést az ellenőrzéssel, talán legegyszerűbben a határolópár technikáknál lehet megoldani a feladatot. Itt ugyanis, ha nem megfelelő párosításu határolók kerülnek egymás mellé, a hozzájuk tartozó mátrix elem egy hiba jelző eljárást vagy hiba javítót aktivizálhat.

Jelen fordító szintén az elemzés alatt hajtja végre a szintaktikai ellenőrzést, azonban a prioritási szint technikánál nem lehet ilyen általános mechanizmust adni, mint a mátrixok alkalmazásánál. Ennek ellenőre viszonylag könnyen ki lehet mutatni a legtöbb szintaktikai hibát. Legegyszerűbb a kifejezések és utasítások szerkezetének elemzése. A kifejezésekénél az a tény, hogy az operanduszok és a műveleti jelek felváltva következnek /egyköt kivételtől eltekintve/, felhasználható a típus vizsgálattal együtt az ellenőrzésre. A határoló jeleket pedig, akárcsak az utasítások strukturáját a verem segítségével ellenőrzi. Pl. amikor az until /vagy a do/ kerülne be a verembe, a verem tetején a step /vagy until/ határolót kell az elemzőnek találnia. Ilyen párok figyelemzésével az utasítások szerkezete ellenőrizhető.

A deklarációk vagy számok helyességét azonban már néhány mutató bevezetésével tudja csak ellenőrizni.

Hiba esetén kiírja a fordító a forrásprogram azon sorának számát, amelyben a hiba előfordult, valamint kiírja a hiba jellegét is. E művelet után felfüggeszti az elemzést és a következő pontosvessző, else vagy end-ig komment-

nek tekinti a program szövegét.

A legtöbb fajta hibát az első menetben jelzi ki a fordító. A második menetben hibák főleg a típusellenőrzésnél adódhatnak, de a harmadik menetben is előfordulhatnak, azonban ezek már a program méretével függnek össze. Futás alatt szintén jelezhet hibát a fordító ill. az FR. Ezek a hibák a formális és aktuális paraméterek megfeleltetésénél tűnhetnek ki.

A hibaellenőrzés megvalósítását még nem tekintem lezártnak mai állapotában. A későbbiek során hibajavító eljárásokat szükséges az elírások korrigálására beiktatni.

3. Az EMG - 83o ALGOL leírása

Ebben a fejezetben az előbbiekben részletesen tárgyalt fordítóprogram által megvalósított ALGOL nyelvet írom le. E reprezentáció akárcsak a legtöbb gépi reprezentáns / [9], [3] /, bizonyos korlátozásokat és bővítéseket tartalmaz az ALGOL-jelentéshez képest. A továbbiakban ezeket a korlátozásokat, bővítéseket, valamint néhány új utasítást fogok bemutatni. A tárgyalást, ahol szükséges, példákkal illusztrálom.

3.1. Korlátozások

Az EMG - 83o ALGOL gyakorlatilag az ALGOL jelentésben megadott nyelv-definíciót valósítja meg, de a fordítóprogram felépítéséből adódóan, valamint néhány nem egyértelmű meghatározás miatt / [11] /, korlátozásokat vezet be.

A korlátozások a következők:

Cimke érték szerint nem hívható.

Integer címke nem lehet aktuális paraméter.

Own-tömbök index-korlátai csak előjel nélküli egész számok lehetnek.

Az azonosítók deklarációjának meg kell előznie azok használatát.

Csak egyszerű idézetek használhatók.

Aritmetikai kifejezésben a specifikálatlan változók, tömbök és függvényeljárások típusát valósnak tekinti.

3.2. Új utasítások illetve bővítések

a. Statement utasítások és kifejezések

Ismertetését lásd a 2.1.5.1.-ben.

b. Gépikódu utasítás használata

Szintaktikus definíció:

```

<gépkódu utasítás> ::= CODE <szimbólikus utasítás
    sorozat> ;
<szimbólikus utasítássorozat> ::= <szimbólikus utasítás>
    <szimbólikus utasítássorozat> , <szimbólikus
    utasítás>
<szimbólikus utasítás> ::= <műveleti jel> <cimrész>
<műveleti jel> ::= <pozitív egészszám>
<cimrész> ::= <azonosító> <relatív cím> <előjel nélküli
    szám> <abszolút cím>
<relatív cím> ::= ± <természetes szám>
<abszolút cím> ::= ? <természetes szám>

```

Példa: SIMPLE -ben felírt utasítások és ALGOL-megfelelőjük:

ra azon	CODE	42	azon,
jz x + 3		48 +	3,
as 2		38	2,
wa 198on		8	' 198o ' ;

A műveleti jel értéke 0-tól 63-ig változhat, az abszolút címé 0-2800 -ig.

c. Könyvtári eljárások használata

Az ALGOL fordító lehetővé teszi különálló eljárások fordítását, amelyeket mágneslemezen szervezett könyvtárba felvihetünk. A lemezeről a memóriába való elhelyezését deklarációval kérhetjük.

A deklarációk szintaktikus leírása:

```

<A könyvtári eljárás deklaráció> ::= LIBRARY PROCEDURE
    <azonosító lista> | LIBRARY <típus>
    PROCEDURE <azonosító lista>
<azonosító lista> ::= <azonosító> <azonosító lista> ,
    <azonosító>

```

Példa: legyen az eljárás a következő:

```
REAL PROCEDURE XYZ /A,B/ ; BEGIN . . . END ;
```


A lefordított és a könyvtárba felvitt eljárást a következőképpen használhatjuk:

```
BEGIN REAL M,N; LIBRARY REAL PROCEDURE XYZ;
```

```
... M:=N + XYZ /M,N/; ... END
```

3.3. Input - output utasítások

A fordítóprogram a számítógép alaprendszere segítségével lehetővé teszi a perifériális egységek használatát. Így a lyukszalagolvasó ill. - lyukasztó, a konzolirógép, a mágnes-szalagos egységek, ill. a cserélhető mágneslemez ALGOL-utasításokkal elérhetők. A perifériakezelő utasítások a következők:

a, Periféria-kijelölő utasítás

$\langle \text{periféria} \rangle ::= \text{PERIFERIA } (\langle \text{aritmetikai kifejezés} \rangle)$
Az aritmetikai kifejezés értékének /egész/ megfelelő lassu perifériát kijelöli.

b, READ - utasítás /lyukszalag, konzol/

$\langle \text{input} \rangle ::= \text{READ } (\langle \text{input-lista} \rangle)$
 $\langle \text{input-lista} \rangle ::= \langle \text{input-listaelem} \rangle | \langle \text{input-lista} \rangle$
 $\quad \quad \quad \langle \text{input-listaelem} \rangle$
 $\langle \text{input-listaelem} \rangle ::= \langle \text{változó} \rangle | \langle \text{tömbnév} \rangle$

c./ WRITE-utasítás /lyukszalag, konzol, sornyomtató /

$\langle \text{output} \rangle ::= \text{WRITE } (\langle \text{output-lista} \rangle)$
 $\langle \text{output-lista} \rangle ::= \langle \text{output-listaelem} \rangle | \langle \text{output-lista} \rangle,$
 $\quad \quad \quad \langle \text{output-listaelem} \rangle$
 $\langle \text{output-listaelem} \rangle ::= \langle \text{aritmetikai kif.} \rangle \langle \text{formátum} \rangle |$
 $\quad \quad \quad \langle \text{logikai kif.} \rangle | \langle \text{idézet} \rangle$
 $\langle \text{formátum} \rangle ::= : \langle \text{aritmetika kif.} \rangle : \langle \text{aritmetikai kif.} \rangle |$
 $\quad \quad \quad : \langle \text{aritmetikai kif.} \rangle | \langle \text{üres} \rangle$

Ha nem adunk meg formátumot, az egész számokat 8 jegyre, a valós számokat pedig 8 egész és 8 tizedesre írja

szöröse lehet.

A műveleti kód értéke 0 vagy 1; 0-nál írás, 1-nél olvasás történik.

g./ DA-utasítás /cserélhető mágneslemez szektorcím beállítása/

DA-utasítás ::= DA (aritmetikai kif.)

Az aritmetikai kifejezés értékének /egész/ megfelelő szektorcímről olvas ill. ír a következő MD-utasításnál.

Ha az MD-utasítást DA-utasítás nem követi, a következő MD-utasítás a legutoljára használt szektor után használja a lemezt.

h./ SPACE-utasítás

SPACE-utasítás ::= SPACE (aritmetikai kif.)

Az aritmetikai kifejezés értékének /egész/ a megfelelő számú szóközt ír ki.

i./ LINE-utasítás

LINE-utasítás ::= LINE (aritmetikai kif.)

Az aritmetikai kifejezés értékének /egész/ megfelelő számú sort emel.

j./ Karakter kezelés

Karakter-input ::= CHI (azonosító)

Az azonosító csak egész típusú változó lehet.

Az utasítás beolvas egy karaktert és elhelyezi a változó alsó byte-jára.

Karakter-output ::= CHO (aritmetikai kif.)

Az aritmetikai kifejezés/egész/ értékét tároló rekesz alsó byte-ján levő karaktert kiírja az output periférián.

k./ String-kezelés

<String-input> ::= INSTRING (<azonosító>)

Az azonosító csak egész típusu tömbnév lehet.

Az utasítás az input periférián két idézőjel közt elhelyezkedő karakter sorozatot olvas be, és sor-
lopfolytonosan elhelyezi a tömb elemeiben.

/A tömb első eleme a karakterekkel töltött elemek
számát mutatja./

<String-output> ::= OUTSTRING (<azonosító>)

Az azonosító csak egész típusu tömbnév lehet.

Az utasítás a tömbben tárolt szöveget kiírja az
output periférián.

1./ Egyéb utasítások

1./ STOP-utasítás

Hatására a célprogram befejezi futását.

2./ PAUSE-utasítás

Hatására a célprogram leáll és kiírja a
\$-jelet. CON-nal tovább indítható.

3.4. Alapjelek

A forrásprogram ISO vagy OPTIMA kódban lyukasztott
lyukszalagon lehet. Az alapjelek használatánál a kö-
vetkező eltéréseket kell figyelembe venni:

ALGOL alapjelek

EMG-830 ALGOL megfelelője

<

LT

≤

LE

>

GT

≥

GE

=

EQ

≠

UEQ

^

AND

<	OR
U	IMP
	EQV
L	NOT
·	DIV
X	x
↑	!
10	E
	=
L	%
L	"
(	(vagy &
)	) vagy ?
' vagy '	,

Az ALGOL alapszavakat nagybetűkkel kell írni és szóközt kell utánuk tenni. Az azonosítók hossza maximum 6 karakter és nem lehet őket szóközzel tagolni.

3.5. A forrás ill. célprogramot jellemző adatok

A fordítóprogram a 16 K szó memória kapacitású EMG-830 számítógépen a programok méretére bizonyos korlátozásokat tesz.

- a./ A célprogram maximum 3600 gépkódu utasításból állhat, melyet az R mezőben helyez el a fordító. A több részletben fordított programok szegmensei szintén maximum 3600 gépkódu utasításból állhatnak.
- b./ Az egyszerre érvényben lévő azonosítók száma 830 lehet.
- c./ A forrásprogramban összesen 700 különböző konstans használható.

A memória felosztása futás előtt:

0n-től 2800n-ig: a β -alaprendszer és a MAT.
függvények

2801n-től 4499n-ig: az ALGOL futtatórendszer

4500n-től max. 8192n-ig: a célprogram a konstansok-
kal együtt

0f-től 8192 f-ig: a dinamikus memória változóknak,
tömböknek, eljárásoknak és a szeg-
menseknek /könyvtári eljárásoknak/.

Irodalom:

- k. Az ALGOL 60 nemzetközi algoritmikus nyelv.
NIM IGÜSZI Budapest, 1971.
2. Az ALGOL 60 nyelv realizálása a "Razdan-3" számítógépre.
[16] melléklete.
3. Az ELLIOTT 803 ALGOL programozási rendszer leírása
[1] melléklete.
4. Feldman, J.A. A formal semantics for computer languages
and its applikation in a compiler-compiler.
Comm. ACM 9/jan. 1966 / /Összefoglalás [5] -ben./
5. ———, Gries, D. Translator writing systems
Comm. ACM 26/febr. 1968./
6. Floyd, R.W. Syntactic analysis and operator precedence
J. ACM 10 /July 1963./ /Összefoglalás [5], [20] -ban./
7. Grau, A.A., Hill, U., Langmaack, H. Translation of ALGOL 60
Springer-Verlag New York 1967.
8. Gross, M., Lentin, A. Mathematische Linguistik
Springer-Verlag Berlin 1971.
9. Számológép, Kiskönyvtár 1971.
10. Kalmár, L. Ist ALGOL wirklich eine algorithmische Sprache?
Automatentheorie und formale Sprachen, Mannheim, Wien,
Zürich. /1969/pp. 305-315.
11. Knuth, D.E. The Remaining Trouble Spots in ALGOL 60
Comm. ACM 7o/okt. 1967./
12. Koster, C.H.A. A compiler-compiler
MR 127/71. Mathematisch Centrum, Amsterdam.
/Összefoglalás [13] -ban./
13. Laborczy, Z. Programnyelvek teljesen formális leírása,
fordítóprogram-író program
Számológép, 1972.3. szám.
14. Lee, J.A.N. The anatomy of a compiler
Van Nostrand Reinhold Company New York 1968.
15. McClure, R.M. TMG- a syntax-directed compiler
Proc. ACM 1965. pp.262-274
16. McCracken, D.D. Bevezetés az ALGOL programozásba
Tankönyvkiadó Budapest, 1971.

17. McKeeman, W.M. An approach to computer language dering
Tech. Rpt. CS 48, Computer Science Dept., Stanford U.
aug. 1966. /Összefoglalás [5]-ben./
18. — , Horning, J.J., Wortman, D.B. A compiler generator
By Prentice-Hall, Inc, Englewood Clitts, N.J. 1970.
19. Naur, P. /Ed./ Revesid report on the algorithmic
language ALGOL 60 Comm.ACM 6/jan.1963./
20. Randell, B., Russel, D.I. ALGOL 60 Implementation
Academic Press, London 1964.
21. Report on Input-Output Procedures for ALGOL 60
Comm.ACM 7/okt. 1964./
22. Samelson, K., Bauer, F.L. Sequential formula translation
Comm. ACM 3/febr. 1960./
/Összefoglalás [5] , [20] -ban./
23. SIMPLE 70 assembler és szervező rendszere
EGSZI 1971.
24. Wirth, N. Weber, H. EULER- a generalization of ALGOL,
and its formal definicion Comm.ACM 9/jan., febr. 1966/
Összefoglalás [5]-ben./
25. Floyd, R.W. A descriptive language for symbol
manipulation. J. ACM 8/okt. 1961./

1.sz. Melléklet

Az EMG-830-20 számítógép utasításai

Az EMG-830-20 szó szervezésű számítógép. / Egy szó 24 bitből áll. / Utasításai egycímes utasítások, amelyek felépítése a következő :

0 - 13 bit	cím rész
14 - 15 bit	operandusz rész
16 - 21 bit	műveleti jel
22 - 23 bit	index rész

A cím értéke 0 - 8192 között változhat. Attól függően, hogy a központi memória két tárrésze közül - R ill. F - melyikben található a cím, az operandusz n ill. f. / Gépi kódban 01 ill. 10. / Az R részbeli címeket indirekt is címezhetjük, az operandusz ilyenkor i/11/ Ha az operandusz d, vagy nincs kiírva, akkor a cím egy számot jelent.

A számítógépnek négy indexregisztere van. Jelölésük: K, L, RM, RF. / Az RM csak az R, az RF pedig csak a F tárrészben használható. /

A műveleti jel 64 féle értéket vehet fel. Ezek közül csak azokat soroljuk fel, amelyeket a fordítóprogram is felhasznál. Mivel a program a gép SIMPLE-70 assemblerén készült, az assemblerbeli jelölésüket is feltüntettem.

A műveleti jelek és jelentésük:

Gépi kód tízes számrendszerben	Betű kód	jelentés
0	dn	üres utasítás
2	js	szubrutin hívás
3	wz	zérust ír a címrészen adott rekeszbe

4	wk	a K tartalma kerül a cím- részen adott rekeszbe
5	wr	az RN-t tárolja
6	wl	az L -t tárolja
7	wf	az RF -et tárolja
8	wa	az akkumulátor tartalmát tárolja
9	pw	az akk. alsó byteját írja ki a cím alsó bytejára
10	wq	a szorzó regiszter tartalmát tárolja
11	sw	az akk. alsó byteját írja ki a cím középső bytejára
13	tw	az akk. alsó byteját írja ki a cím felső bytejára
20	rk	olvasás a K regiszterbe
21	rr	olvasás az RN regiszterbe
22	rl	olvasás az L regiszterbe
23	rf	olvasás az RF regiszterbe
26	rq	olvasás a szorzó regiszterbe
28	sh	az akk., a szorzó vagy mindkét regiszter léptetését végzi
32	ba	az akk. és a címrész tartalmá- nak bitenkénti konjunkcióját veszi.
33	pr	a címrész alsó byteját az akk. alsó bytejára írja
34	bu	az akk. és a cím tartalmának bitenkénti antivalenciája
35	ar	a cím középső byteját az akk. alsó bytejára írja
36	aa	aritmetikai összeadás
37	tr	a cím felső byteját az akk. alsó bytejára írja
38	as	aritmetikai kivonás
40	an	aritmetikai negatív kivonás
42	ra	olvasás az akk.-ba

44	am	aritmetikai szorzás
46	ad	aritmetikai osztás
48	jz	ugrás, ha az akk. tartalma 0
49	jv	ugrás, ha az akk. tartalma nem nulla
50	jn	ugrás, ha az akk. tartalma negatív
56	jo	ugrás, ha az akk. tulesordult
57	ju	feltétlen ugrás
58	jp	ugrás, ha az akk. tartalma pozitív
62	lj	ugrás, ha az L tartalma nem nulla

Lebegőpontos számokkal a gép alaprendszerében található szubrutinok segítségével számolhatunk. /Itt az akk.-nak a pseudoakkumulátor felel meg,/ A SIMPLE-ben makro utasításokkal hivatkozhatunk rájuk. Ezek a következők:

Név	Jelentés
PAR	a pakk.-ba olvasás
PAW	a pakk. tartalmát ki- írja a megadott címre
RAD	lebegőpontos összeadás
RSU	lebegőpontos kivonás
RMU	lebegőpontos szorzás
RDI	lebegőpontos osztás

Az assembler a periféria kezelést is szubrutinokkal valósítja meg. Az egyes makrok a következők:

Karakterkezelő input-output makrók: CHI, CHO.
Egész ill. valós szám input INI, REI és output INO, REO.
Szöveg input-output: TXI, TXO.

2. Melléklet

A határolókhoz rendelt prioritási számok

Határolók:	f/x/	g/x/
[(	0	-
IF, BEGIN, DO	0	-
SIN, COS, LN, EXP, ATG		
SQRT, ABS, ENTIER, SIGN	0	-
PERIFERIA, READ, WRITE	0	-
SPACE, LINE	0	-
MT, MD, TU, DA	0	-
CHI, CHO, INSTRING		
OUTSTRING	0	-
] , ; END	-	1
)	-	2
THEN E	0	2
THEN S, ELSE S	1	2
ELSE E	2	2
GO TO	2	-
:=, T:=	2	12
-	3	3
=	4	4
>	5	5
<	6	6
^	7	7
]	8	8
> < ≤ = ≠	9	9
+ -	10	10
÷ / NEG	11	11
↑		

3. Melléklet

Első menet

A fordító a metanyelvű program kialakításánál az ALGOL alapszavakat ill. a műveleti és elválasztó jeleket kódolt formában használja. A metanyelvben alkalmazott kódszámok a következők:

kódszám:

jelentése:

1	indexes változót jelöl /1/
2	smitch olemre hivatkozást jelöl /1/
3	utasítás vége / ; /
4	statement kifejezés kezdete / (/
5	blokkkezdő /Begin/
6	statement kifejezés vége /) /
9	CODE utasításban relativ cím következését jelzi.
10	CODE utasításban azonosító vagy konstans következését jelzi
11	sin standard függvény
12	cos "- "-
13	ln "- "-
14	exp "- "-
15	sgrt "- "-
16	arctg "- "-
17	abs "- "-
18	entier "- "-
19	sign "- "-
20	beolvasást ír elő /minden azonosító után/
21	kiírást jelöl /minden kifejezés után/.
22	SPACE
23	LINE
24	PERIFERIA
25	indexkifejezést, vagy korlátpár elemet jelöl

26	THEN V
27	ELSE E
28	ELSE S
29	GO TO
30	:= /ha nincs már több érték-
	adás az utasításban/.
31	:= /ha van még értékadás
	az utasításban/
32	EQV
33	IMP
34	OR
35	AND
36	CODE utasításban műveleti
	jel következik.
37	CODE utasításban abszolút
	cím következik.
38	NOT
39	LT
40	LE
41	UEQ
42	EQ
43	GE
44	GT
45	+
46	-
47	
48	*
49	DIV / ÷ /
50	NEG /előjel/
51	! /hatványozás/
52	IF
53,54	THEN E
55	FOR
56	UNTIL és cikluslista
	elválasztó
57	:= /ciklusváltozónak
	értékadás/

58	STEP-UNTIL cikluslista elem kezdete
59	STEP
60	aritmetikai cikluslista elem következik
61	WHILE- cikluslista elem kezdete
62, 63	DO
64	TRUE
65	FALSE
66	cimke deklaráció <code>://</code>
67	ciklusvége <code>/;/</code>
68	OWN tömbdeklaráció
69	tömbdeklaráció
70	eljárás szintszám mutató következik
71	olyan változót jelöl, amely értéket kap
72	switch elemek kezdete <code>:=/</code>
73	aktuális paramétert jelöl
74	switch deklarációban elválasztó
75	eljárás behívás paramétereinek következik
76	formátum jelző kifejezés következik
78	blokkvége <code>/END/</code>
79	eljárástörzs kezdete
80	switch deklaráció vége
81	eljárás beleírás paramétereinek vége
82	value listán szereplő változót jelöl
84	kocsivissza-soremelés
85	PAUSE
90	Library-deklaráció
91	STOP

92	eljárástörzs vége
93	value listán szereplő tömböt jelöl
94	függvényeljárás paraméterei következnek
96	READ
97	WRITE
98	INSTING
99	OUTSTRING
100	MT
101	MD
102	TU
103	DA
105	CHI
106	CHO

/ A műveleti jeleknél a kódszám értékéhez hozzáadódik
a tárolt prioritási szint. 2^7 ./.

A metanyelvben az azonosítókat és konstansokat a típusok,
valamint a relatív címük jelöl. Általános alakjuk:

TIPUS . 2^{17} + relatív cím

A programban deklarált első csoportbeli mennyiségek
relatív címe 1000-től, a címkekhez és eljárásokhoz
rendelt 500-tól, az OWN változóké 9000-től, a konstanso-
ké pedig 1500-tól indul.

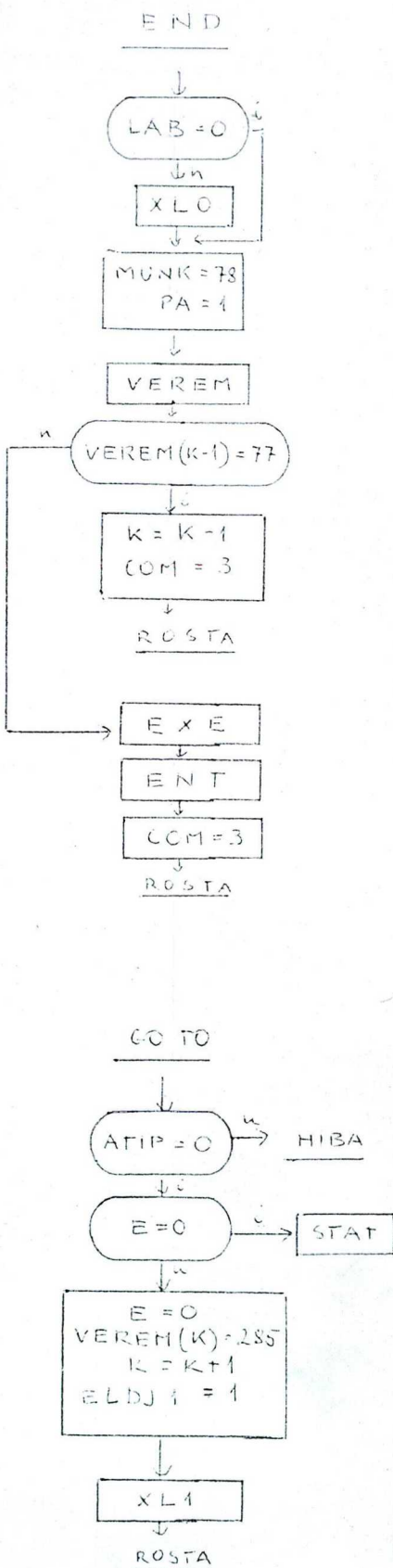
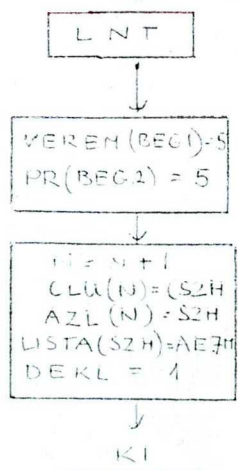
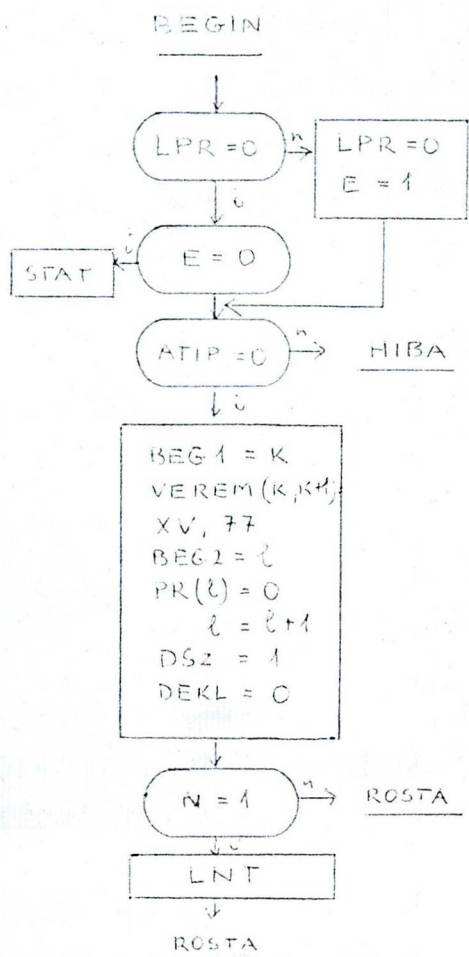
A TIPUS értéke és jelentése a következő lehet:

1	integer
2	real
3	boolean
4	statement
9	procedure
10	címke
11	integer procedure
12	real "
13	boolean "

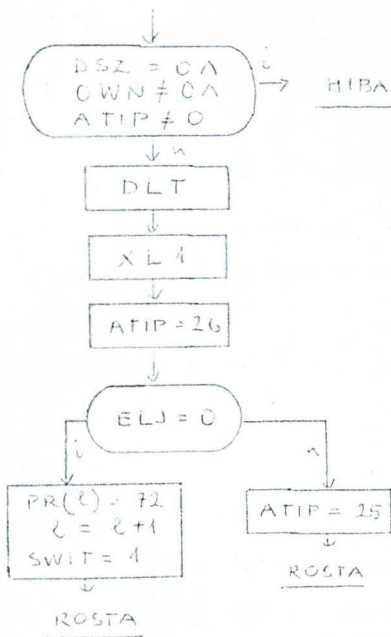
14	statement procedure
15	specifikálatlan paraméter.
16	integer specifikált paraméter.
17	real " "
18	boolean " "
19	statement " "
20	label " "
21	integer array
22	real "
23	boolean "
24	statement"
25	switch specifikált paraméter
26	switch
29	integer array spec. paraméter
30	real " " "
31	boolean "
32	statement "
33	procedure spec.paraméter.
34	integer procedure spec.paraméter
35	real " "
36	boolean "
37	statement "
39	string

Fontosabb változók:

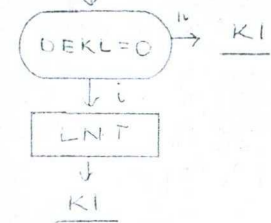
LPR	programkezdetnél 1 az értéke, később nulla
E	kifejezés fordításánál nulla az értéke
DSz	ha értéke 1, deklaráció következhet
PR	metanyelvi programvektor
N	blokkszintszám
CSZH	cikluslista pointer
SZH	azonosító lista pointer
LISTA	azonosító lista
AEJM	eljárás szintszám mutató /aktuális/
ATIP	típus
XV	relatív cím
PH	prioritási szint
PLEV	eljárás szintszám
C LISTA	címke lista
CLV	" " belépési pontok vektora
AZLV	azonosító lista " " "
SZLIST	Konstansok táblázata
KSz	" táblázatánál pointer
XLA1	címkeknél relatív cím.
XPV1	eljárásoknál " "
TÁBLA	rutin, amely az egyes karaktereket felismeri
DIM	indexes kif. dimenziója
ADL	azonos korlátpáru tömbök száma



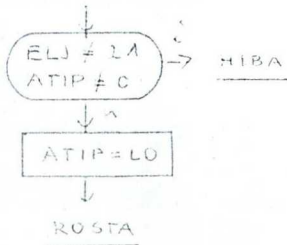
SWITCH



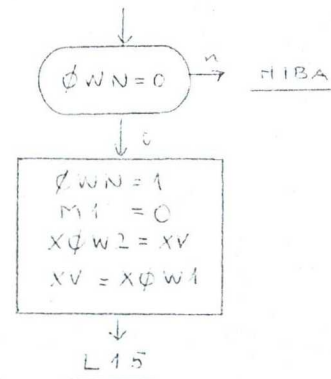
DLT



LABEL



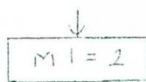
OWN



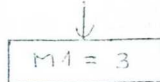
INTEGER



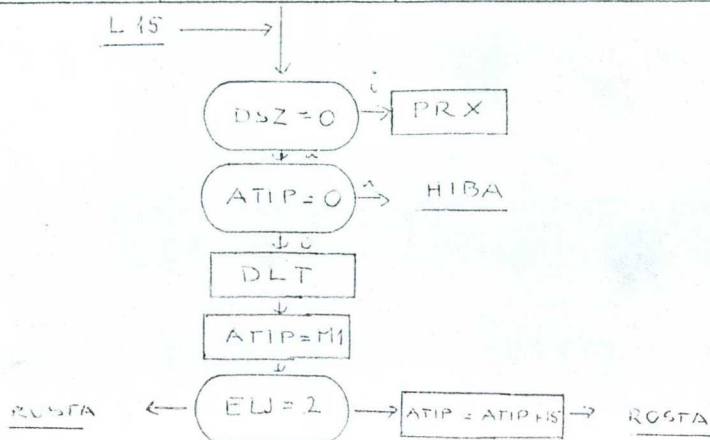
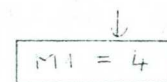
REAL

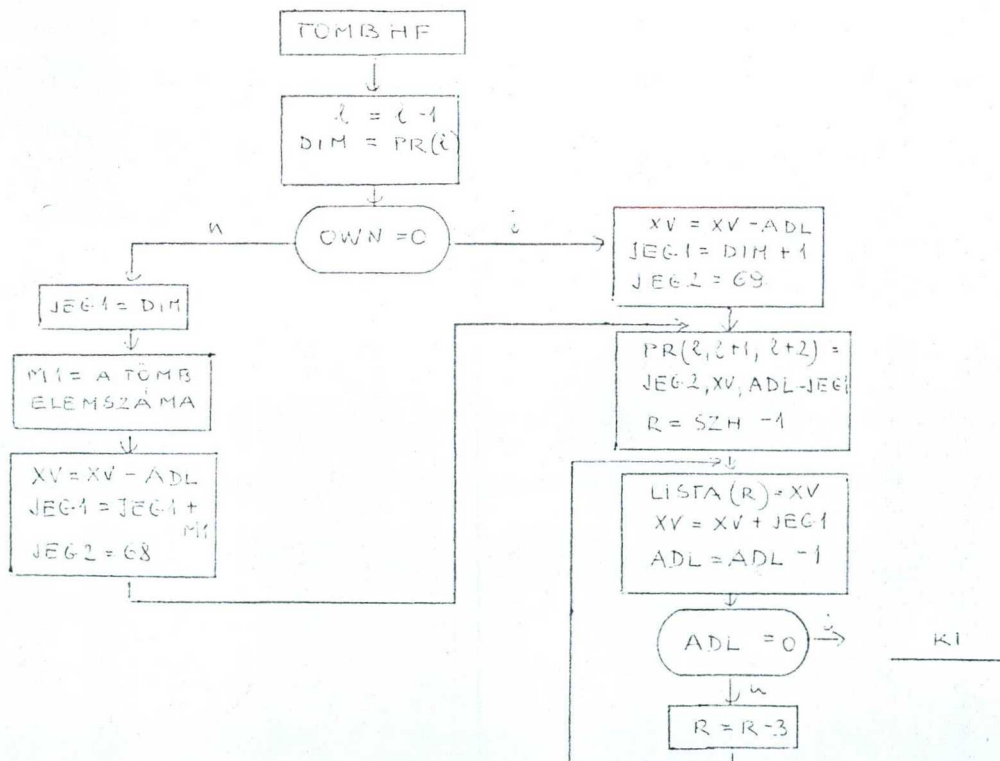
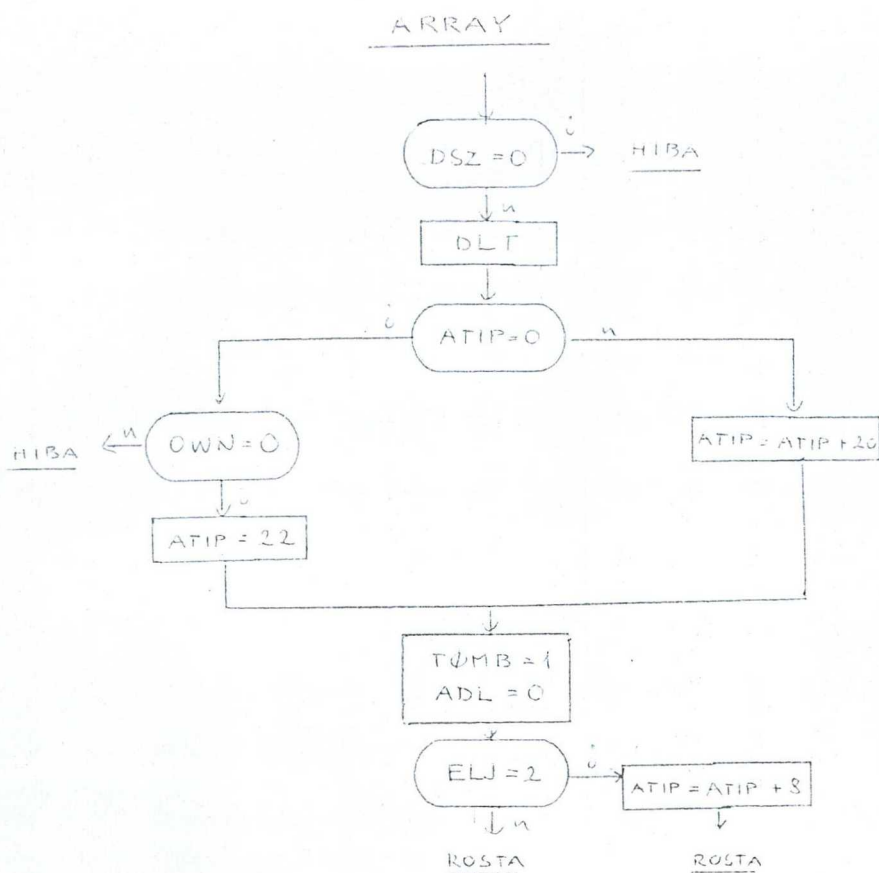


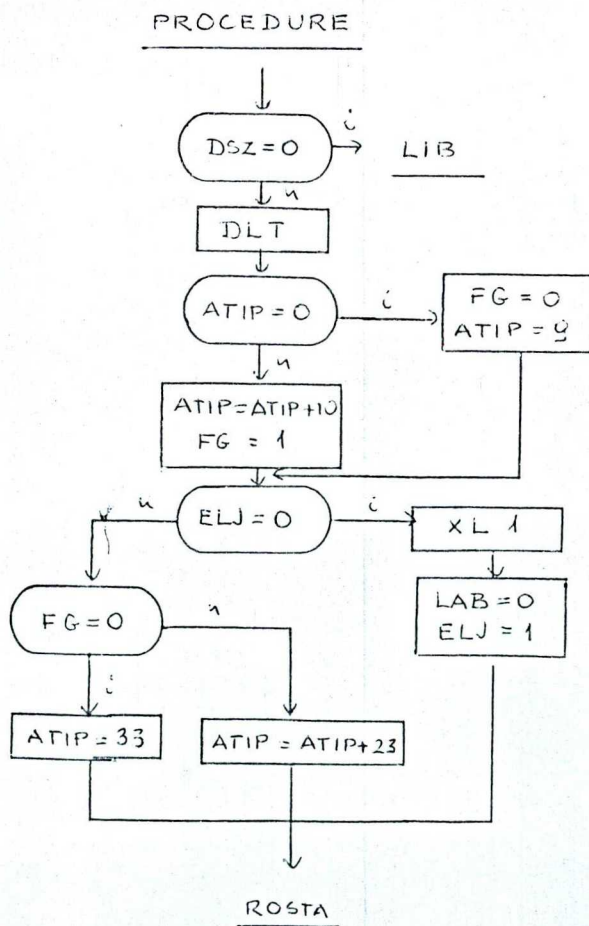
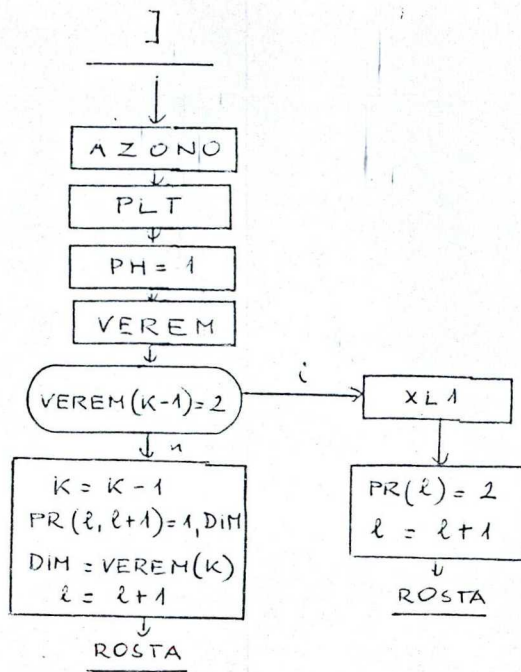
BOOLEAN

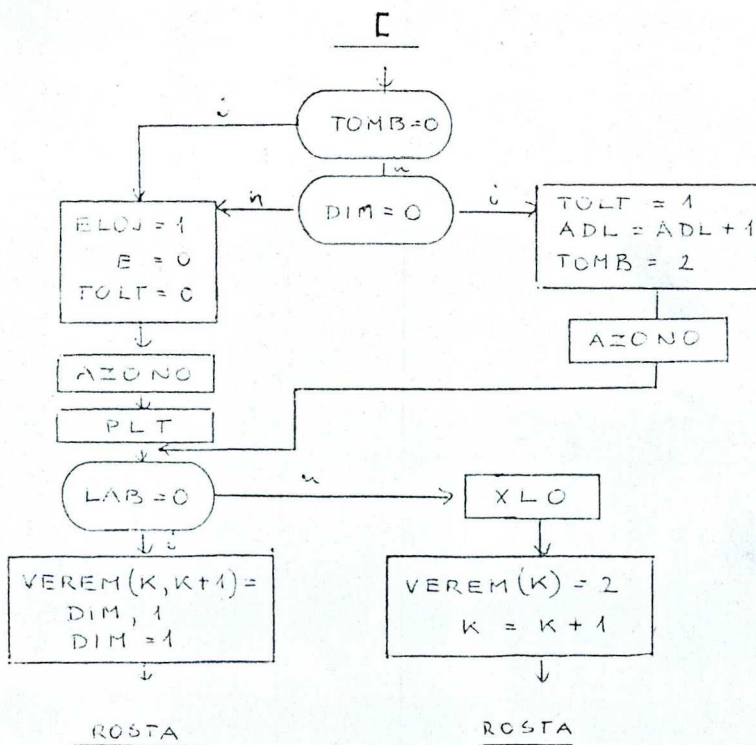
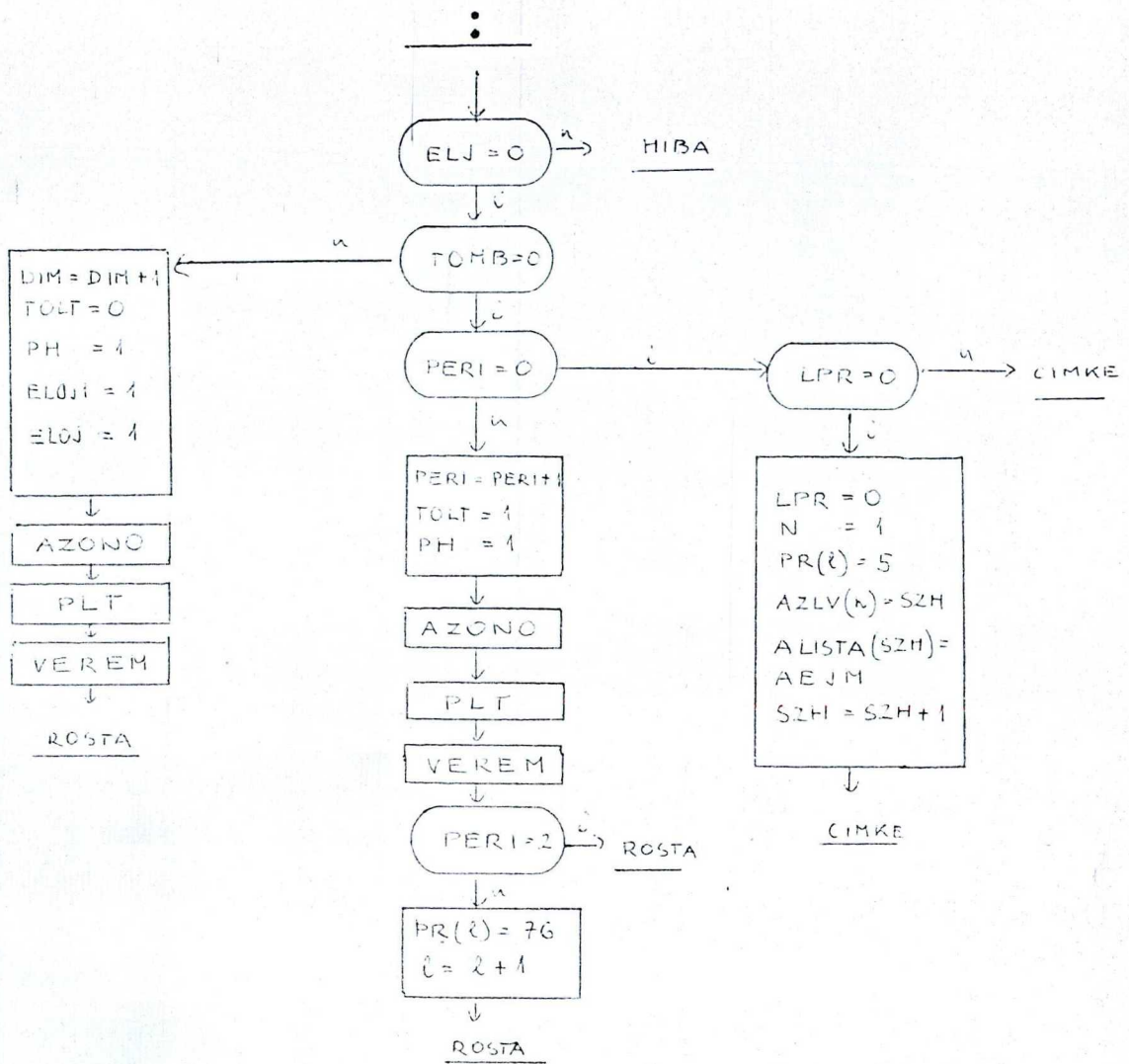


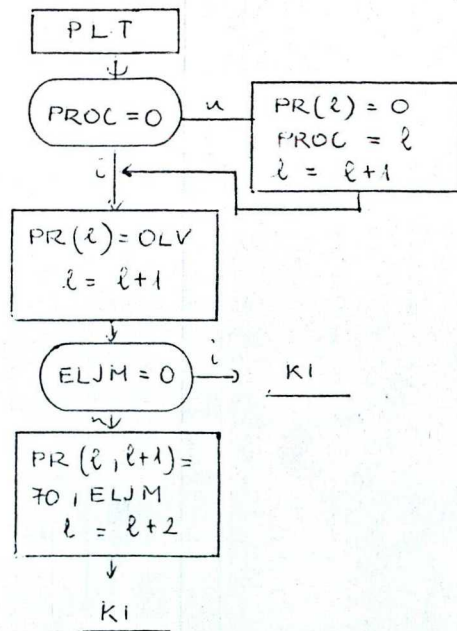
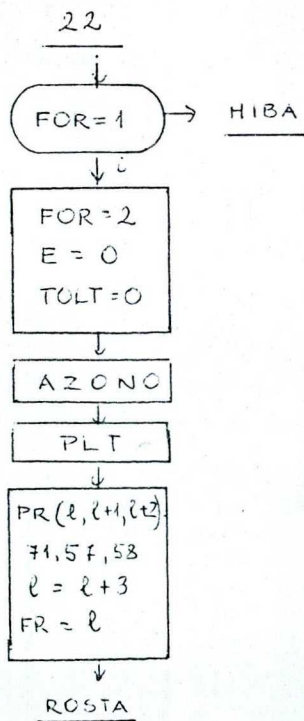
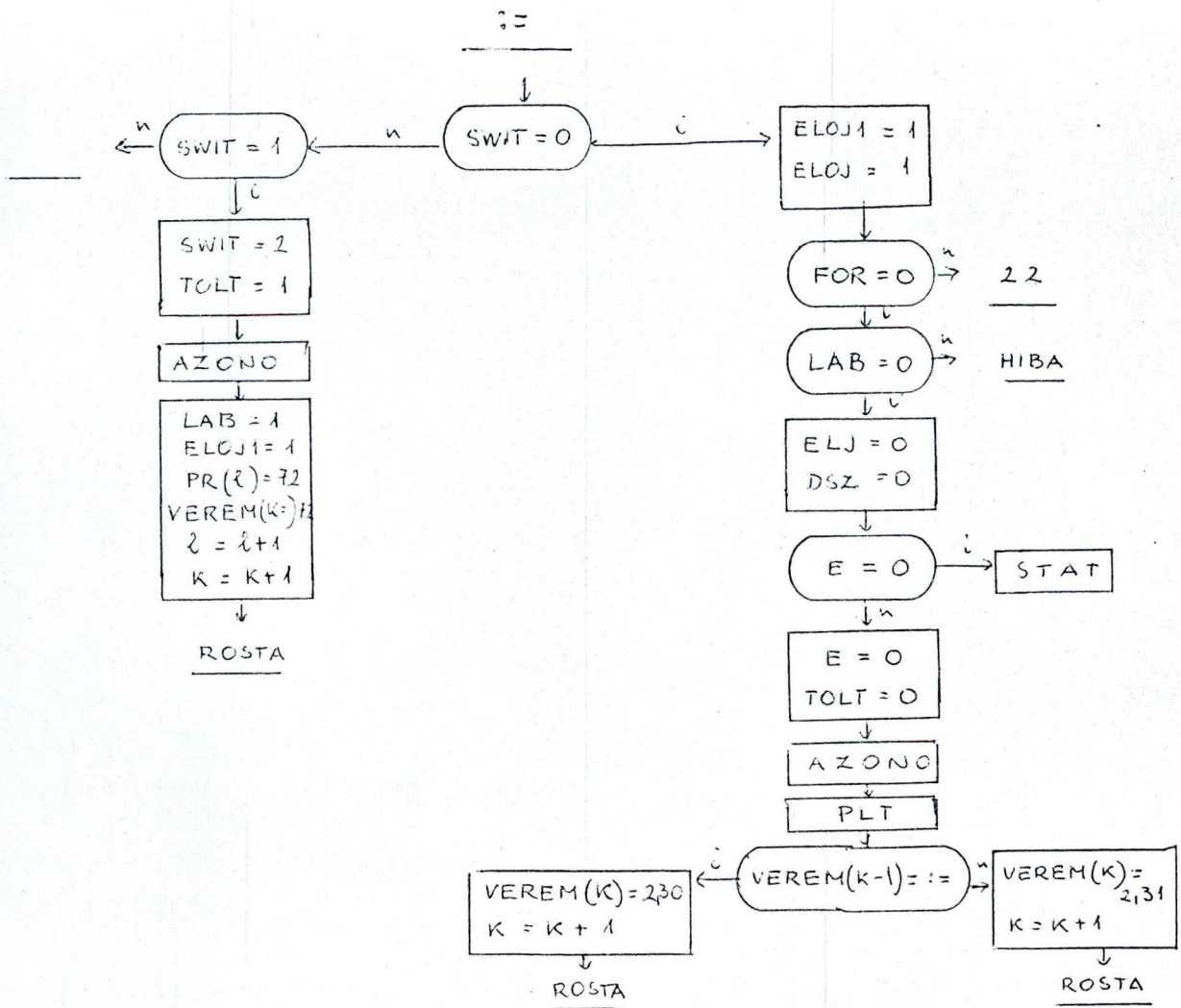
STATEMENT

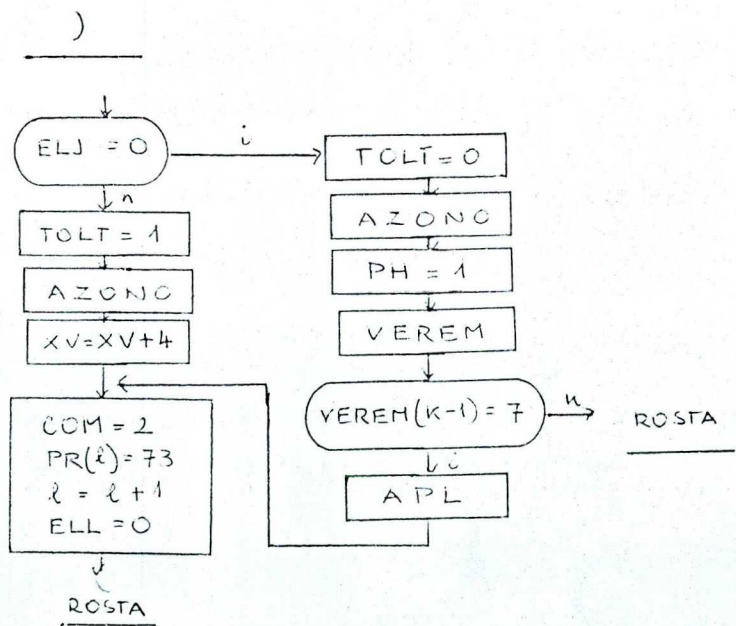
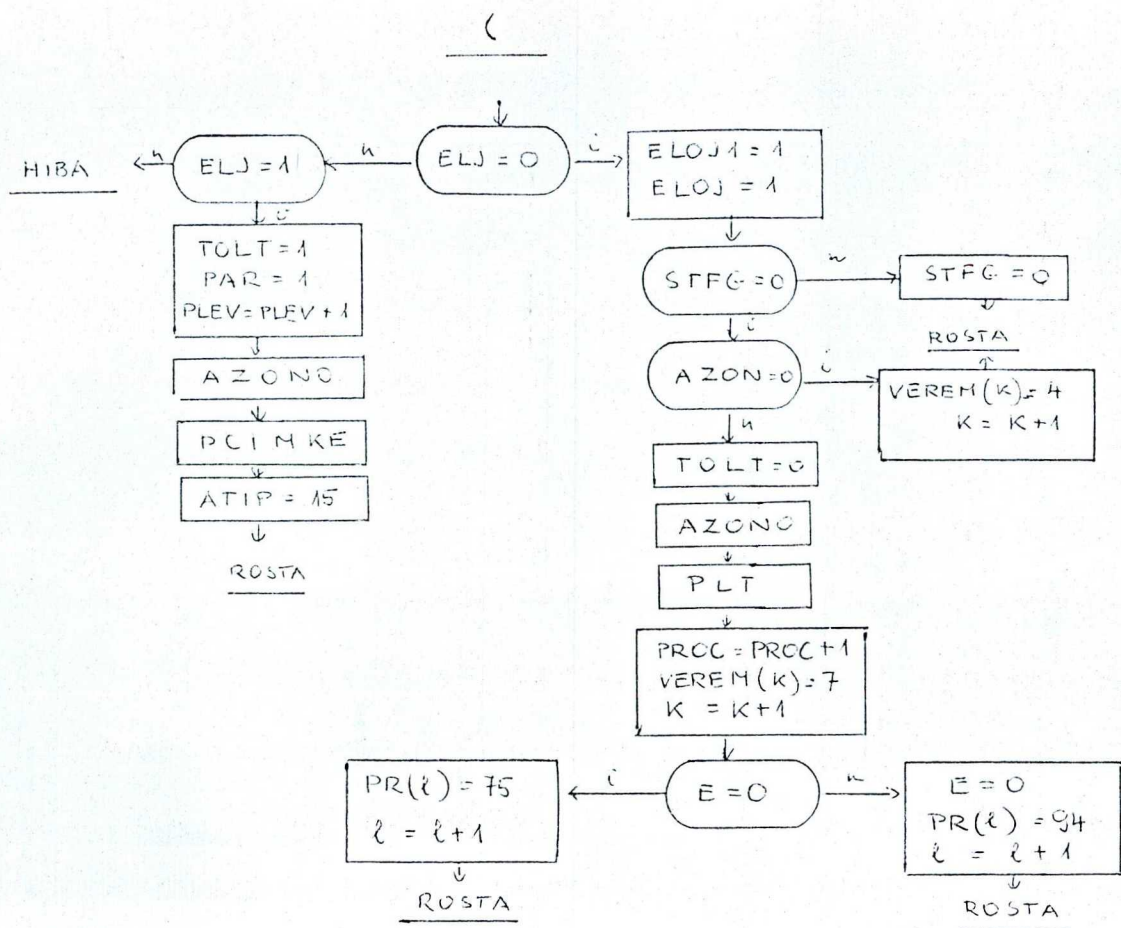


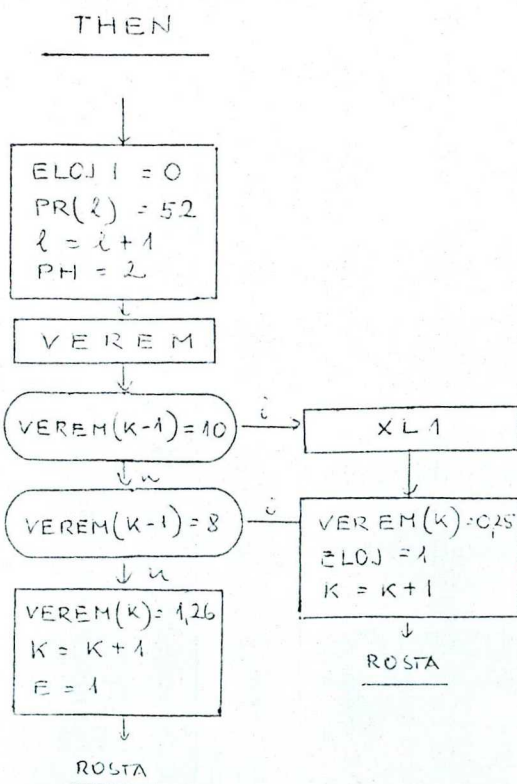
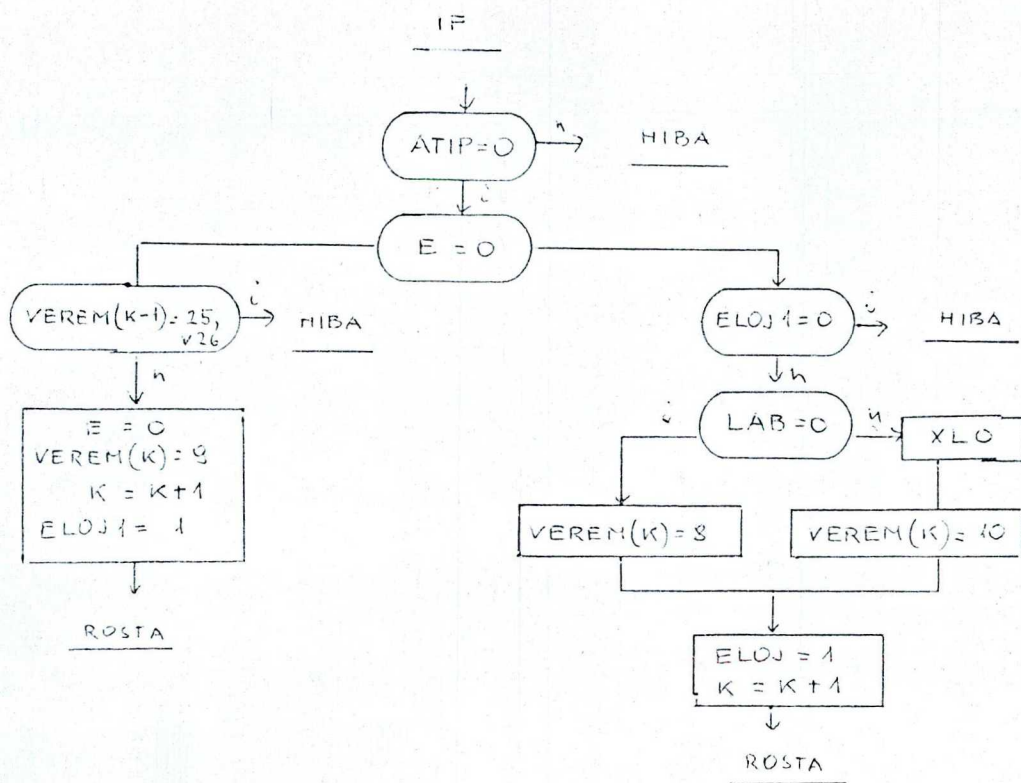


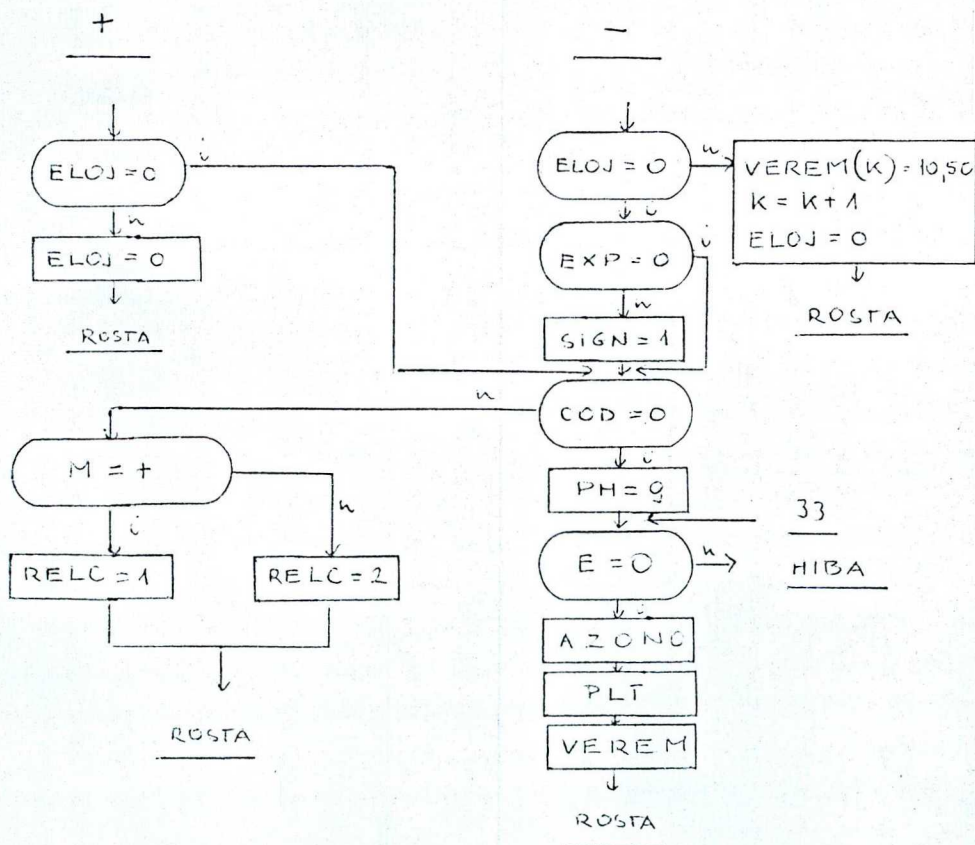
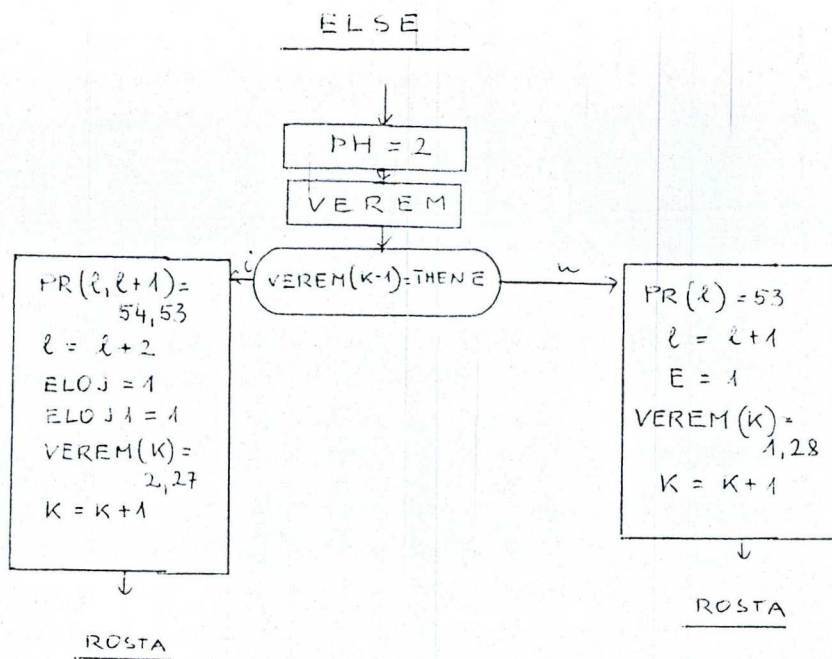


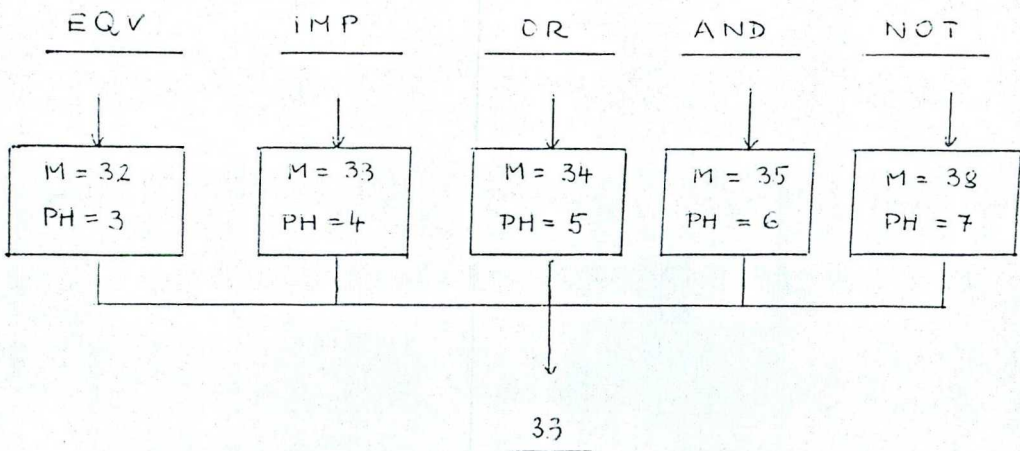
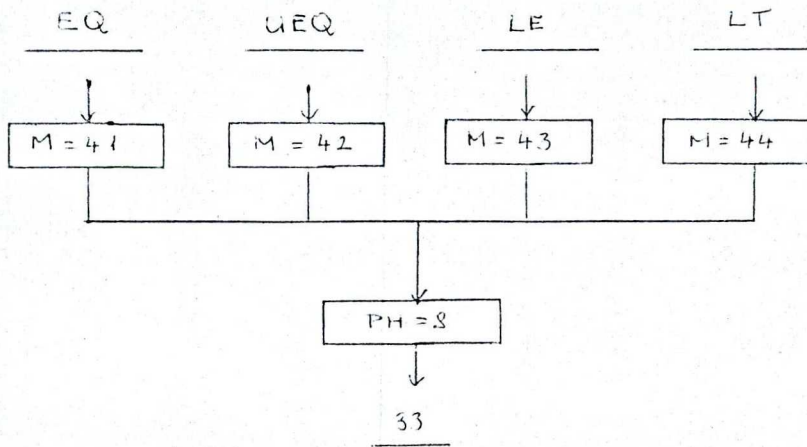
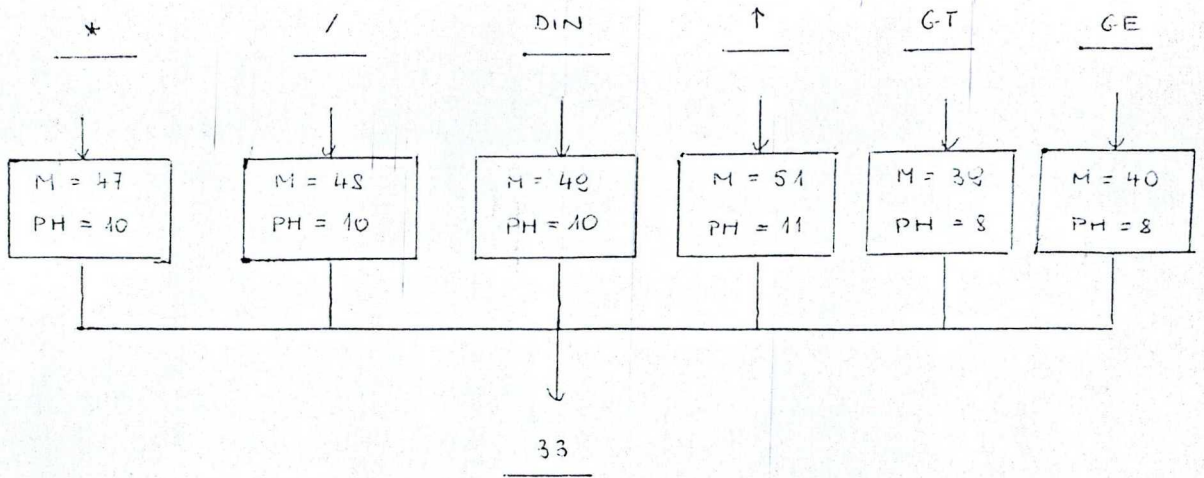




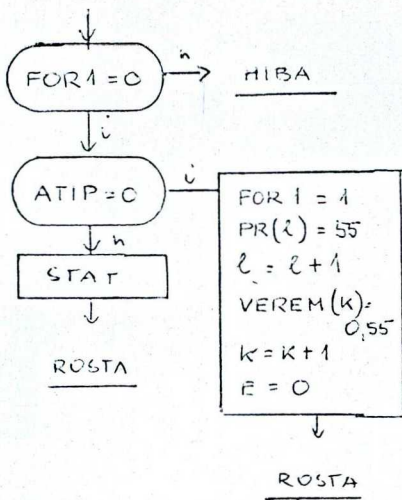




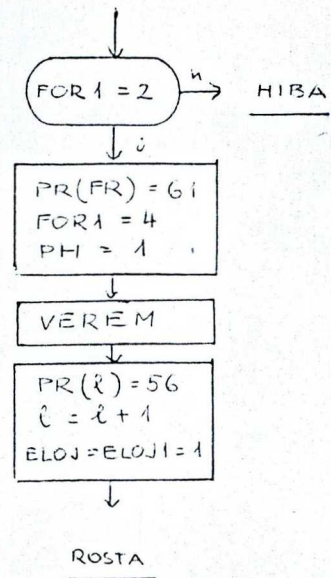




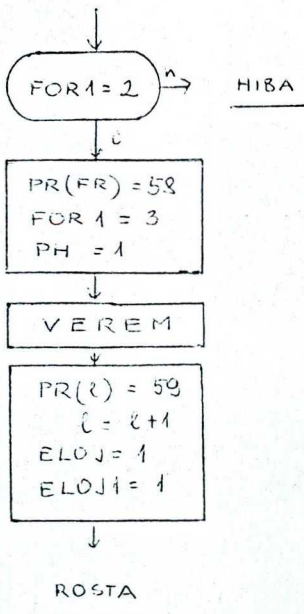
FOR



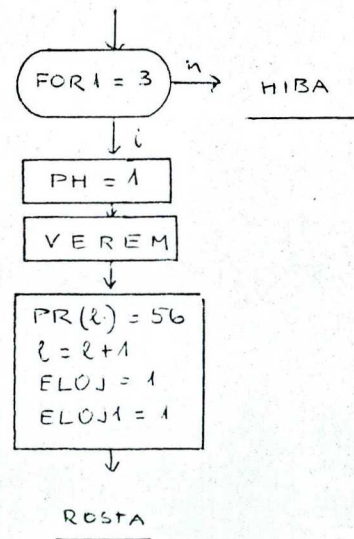
WHILE

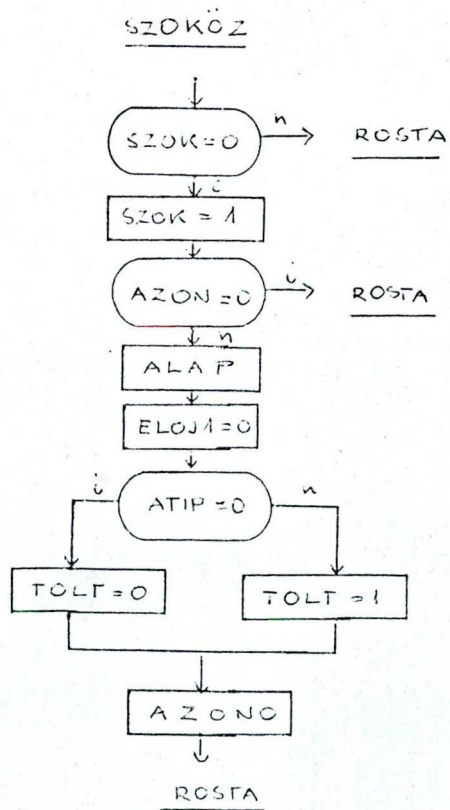
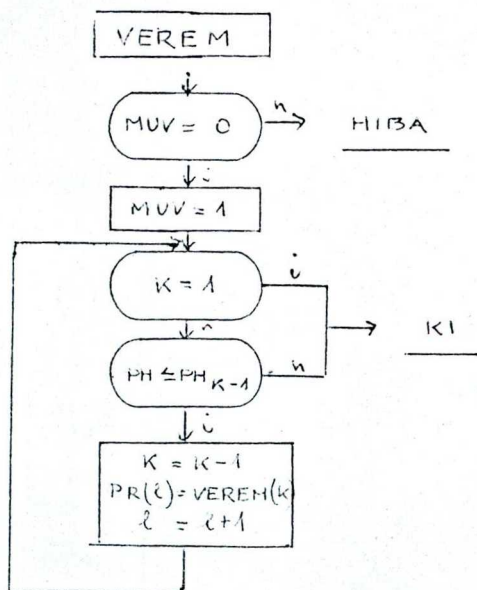
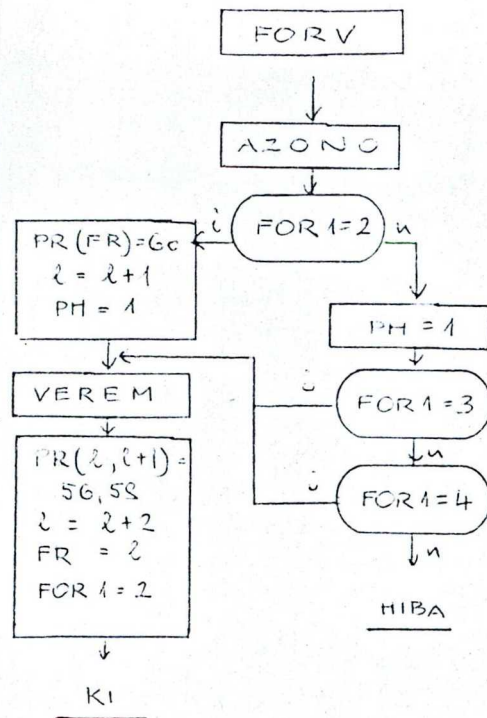
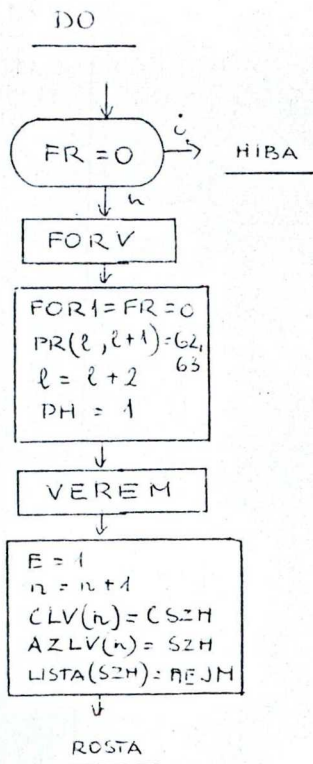


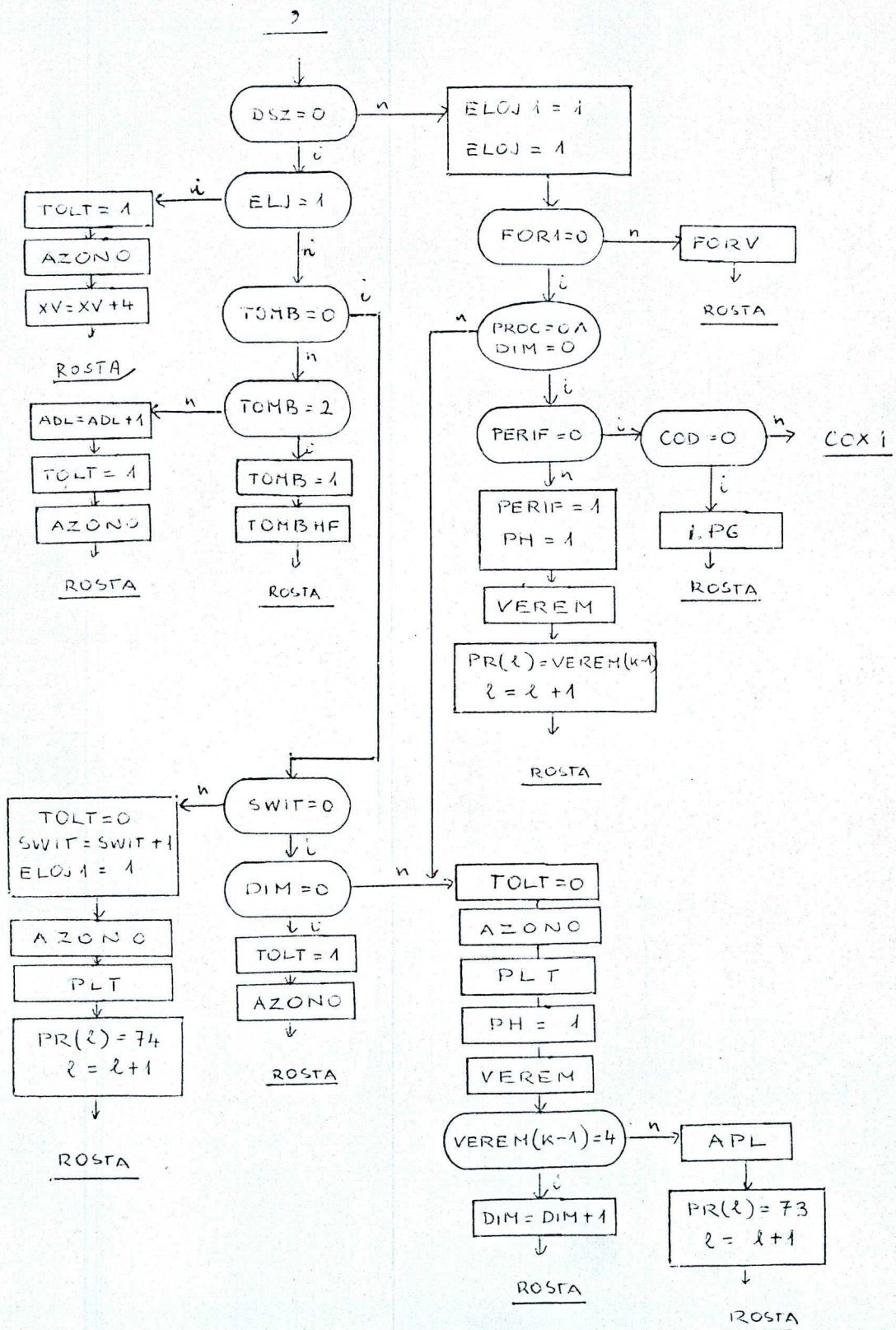
STEP

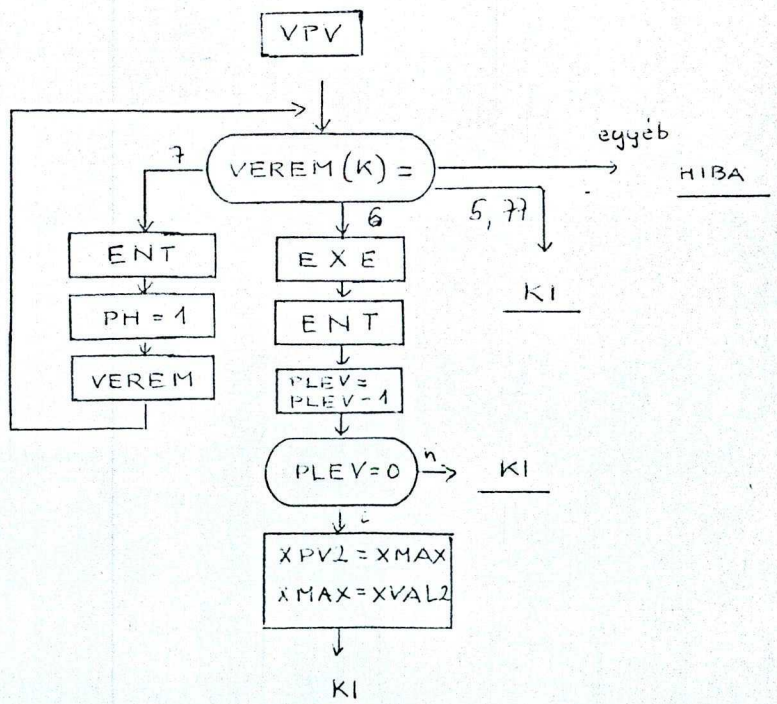
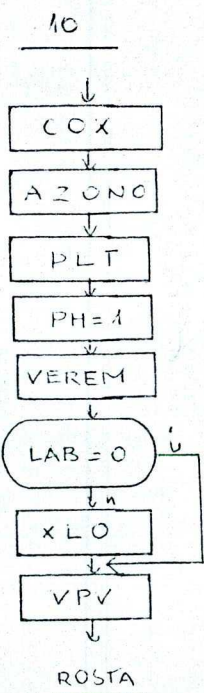
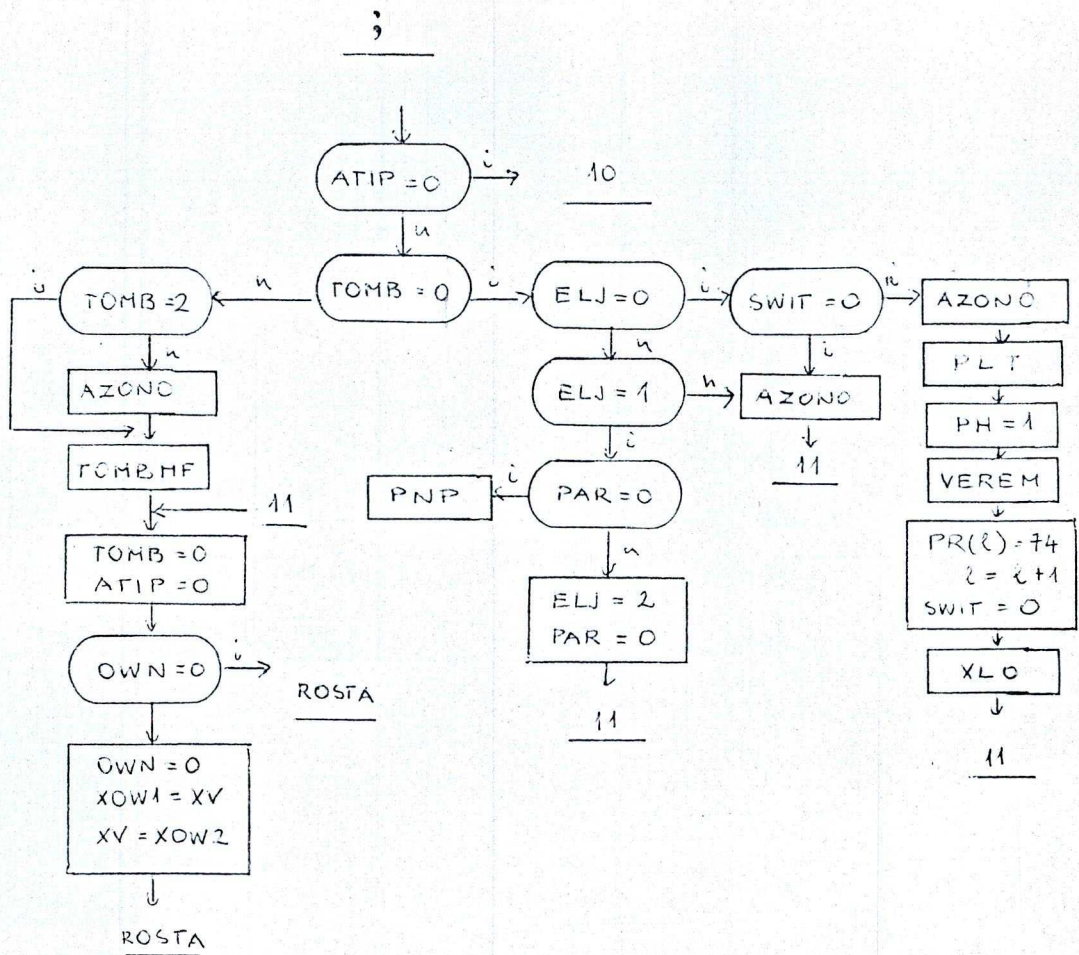


UNTIL

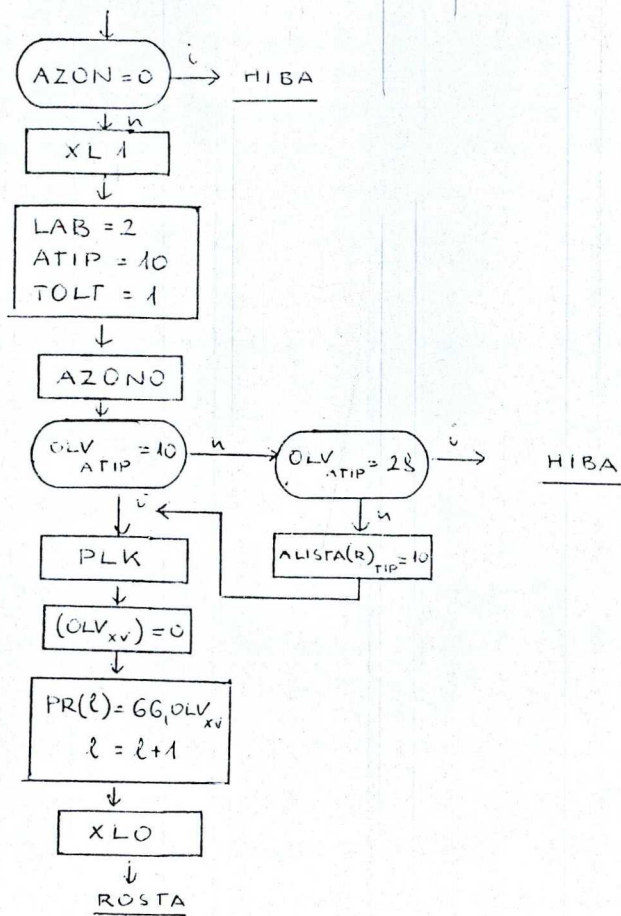




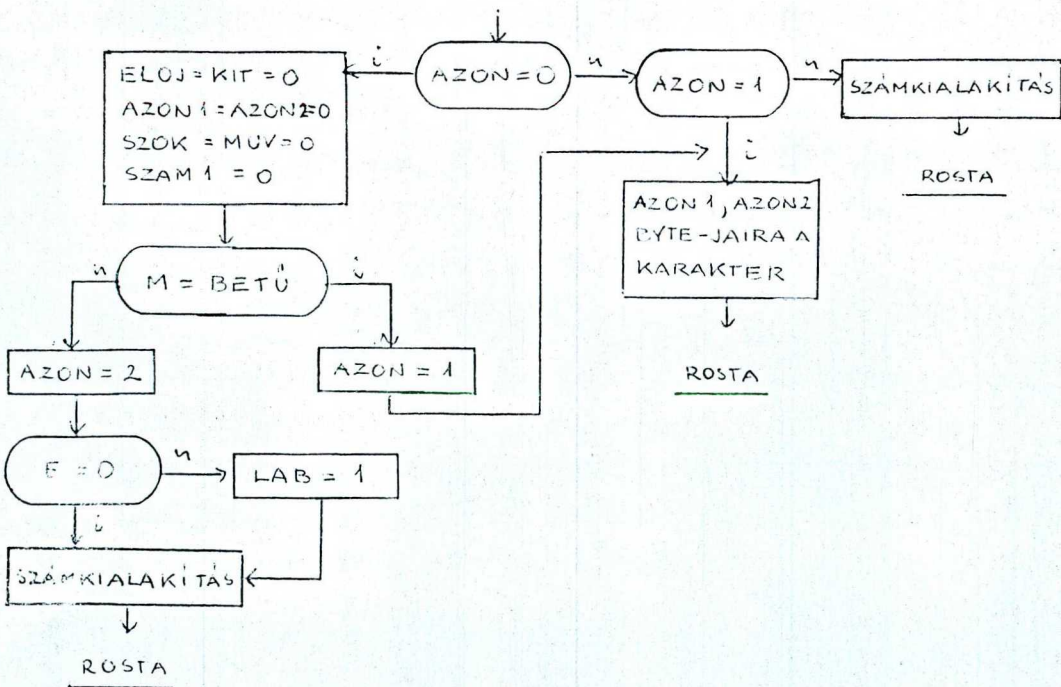




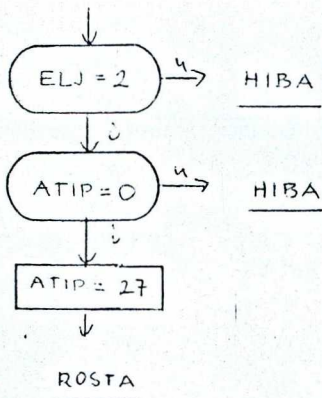
CIMKE



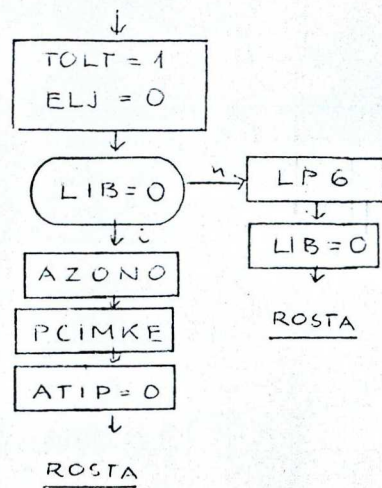
IDEN



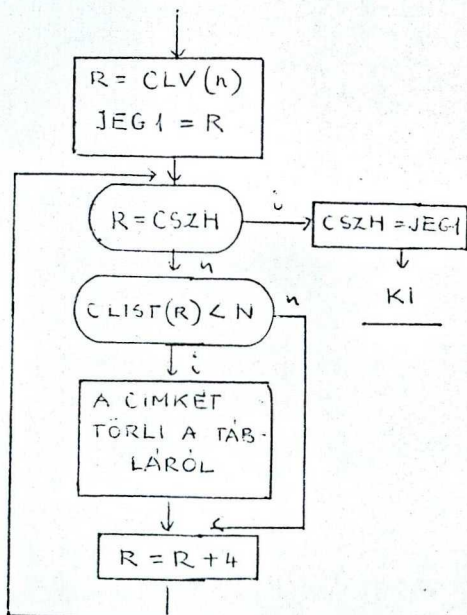
VALUE



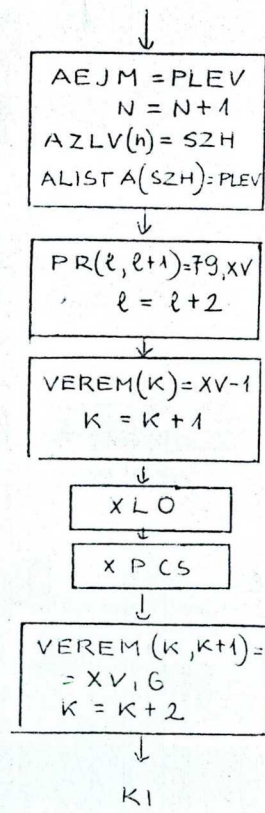
PNP

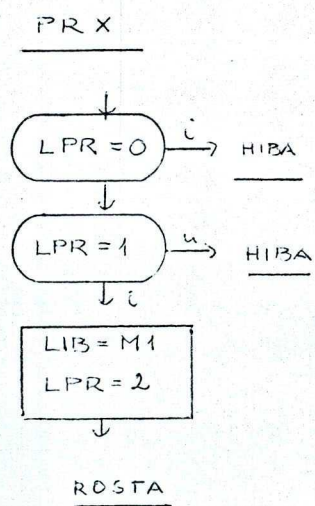
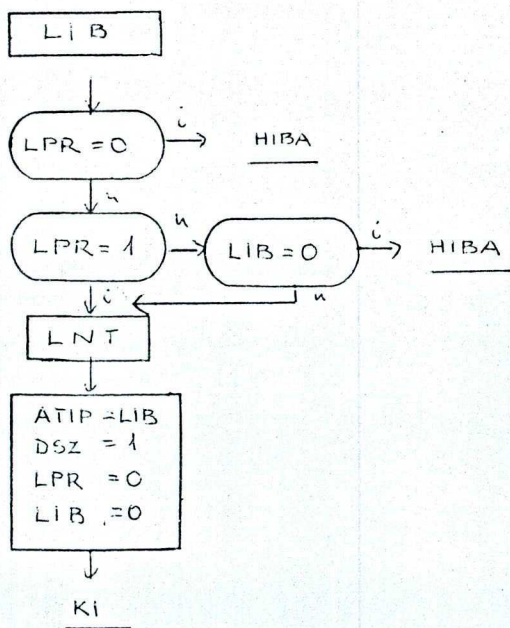
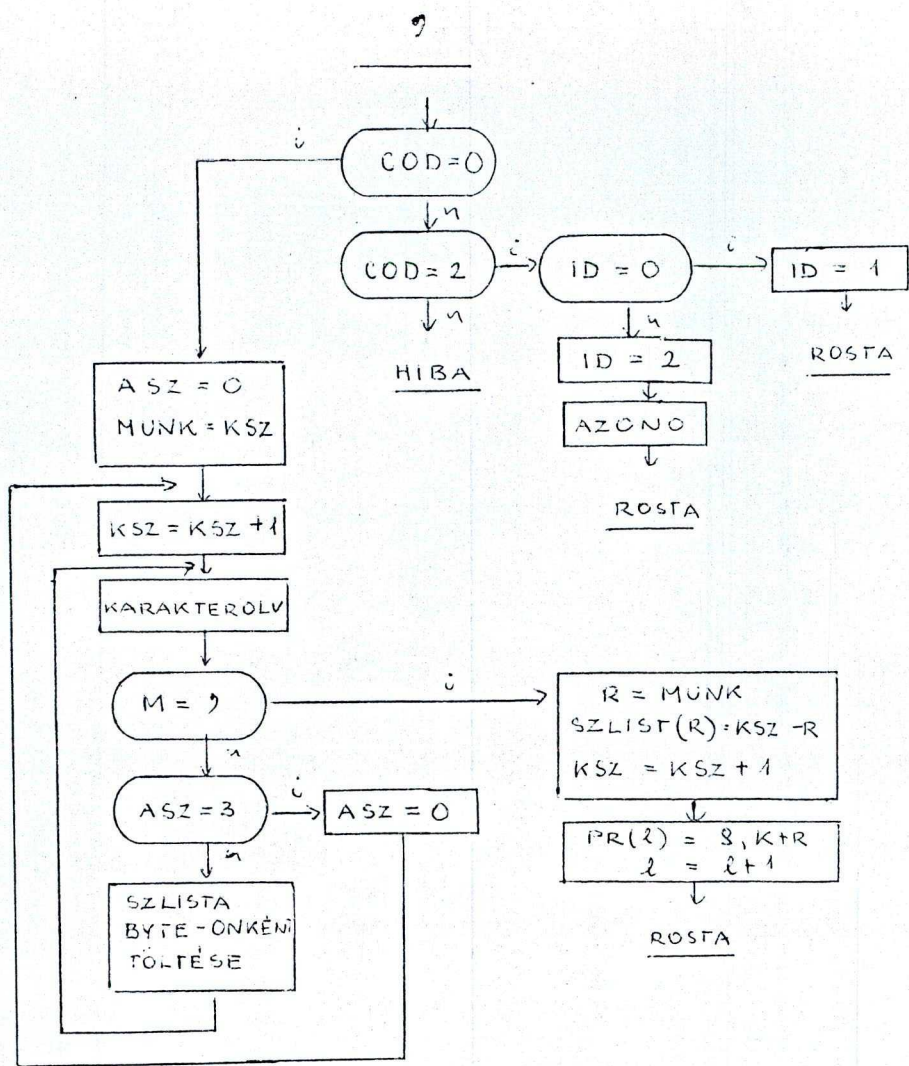


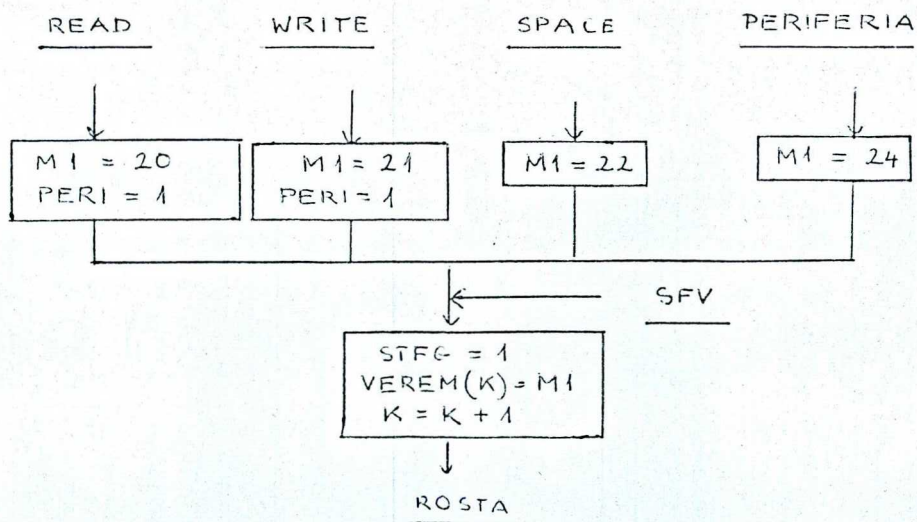
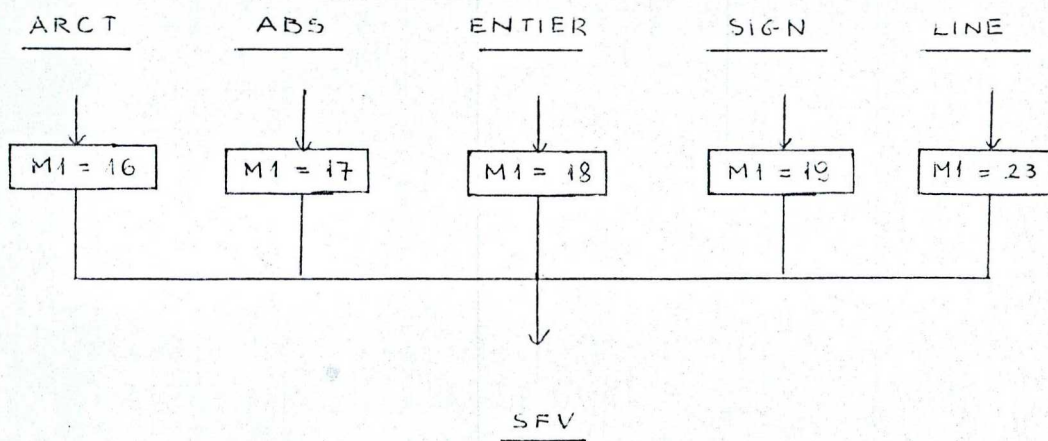
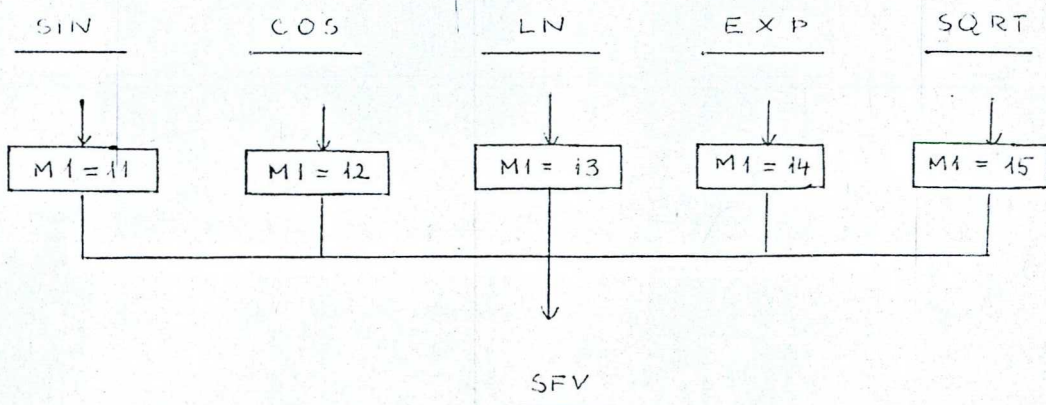
ELTOL

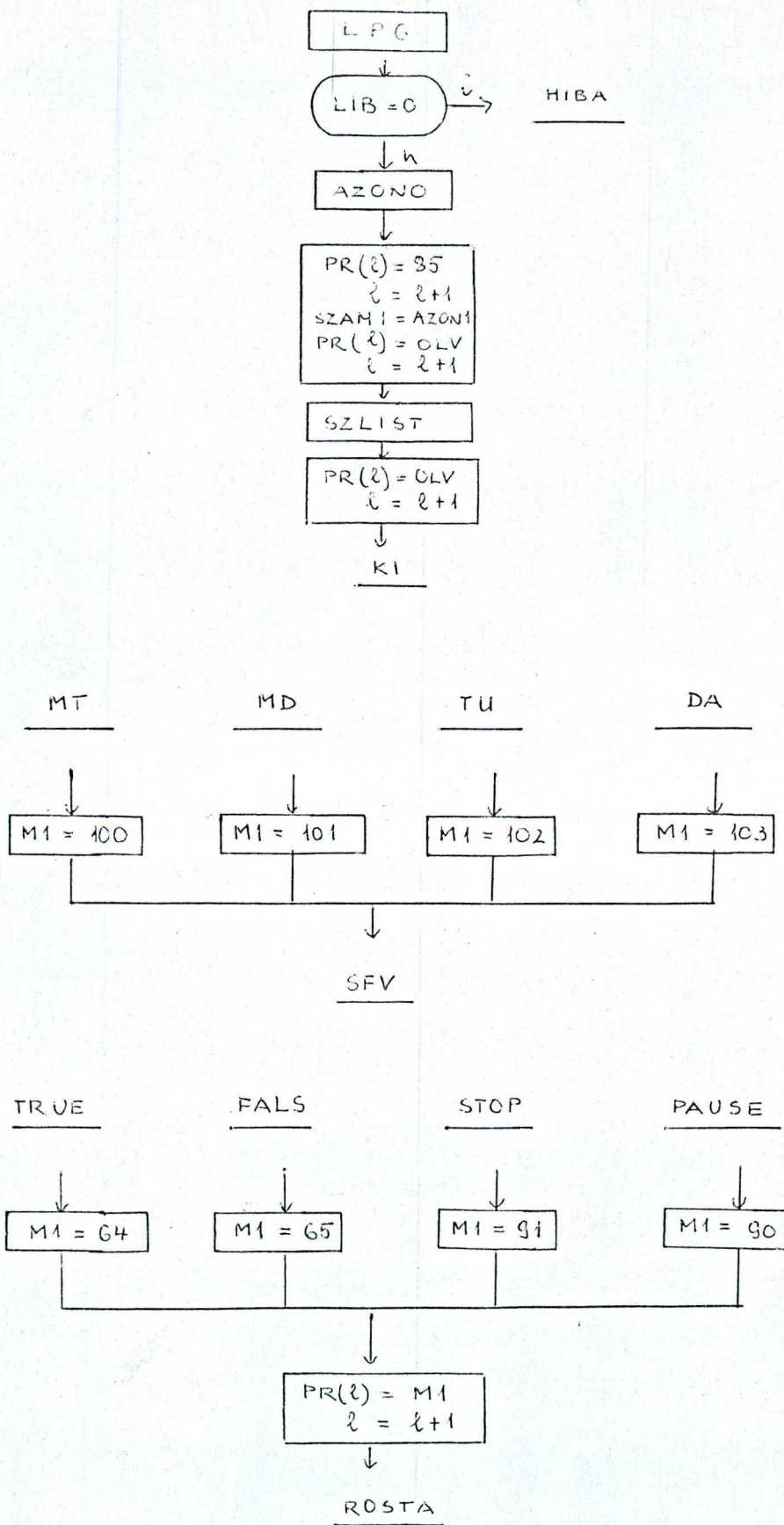


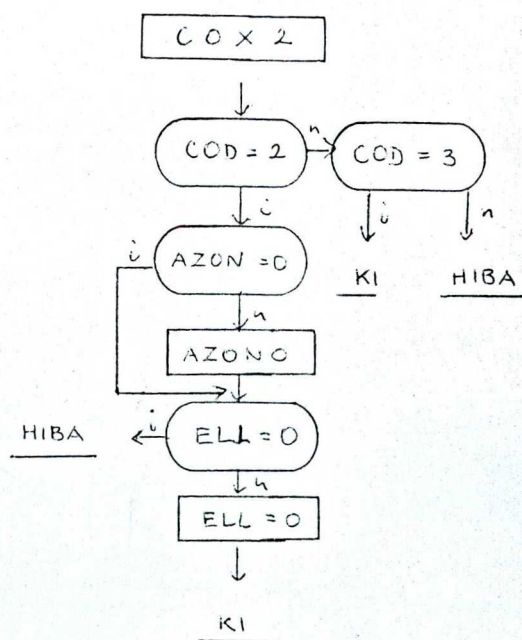
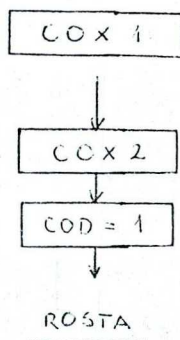
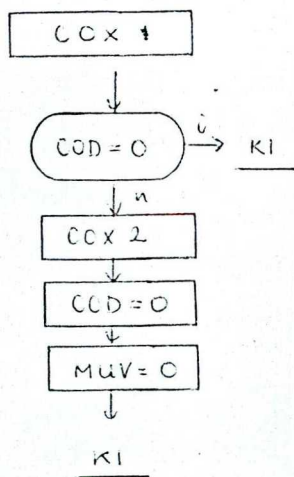
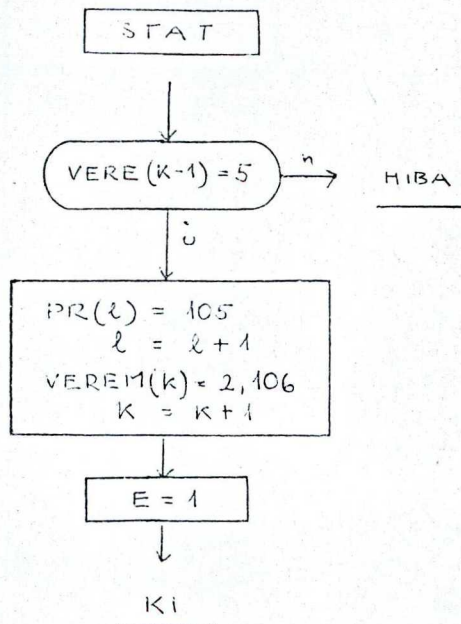
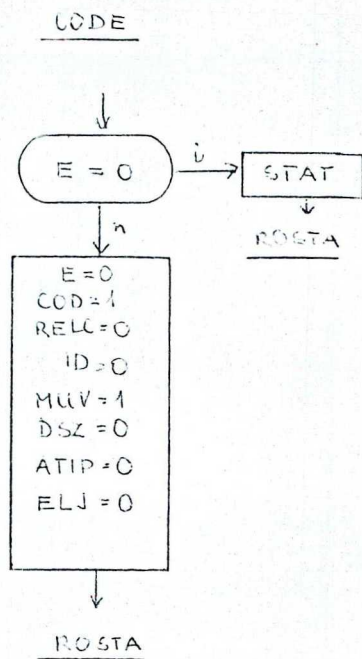
PCIMKE

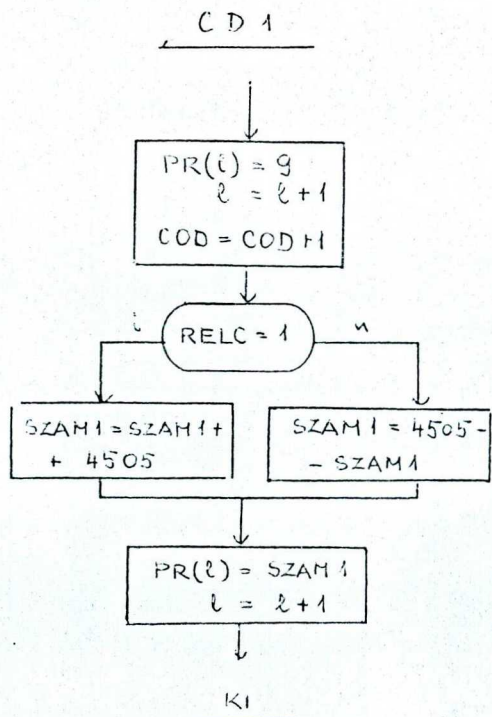
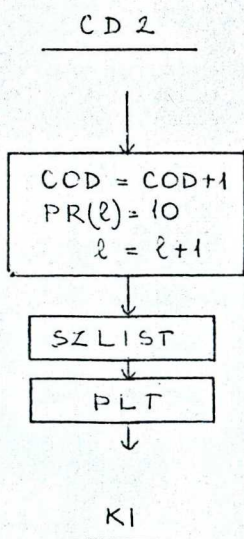
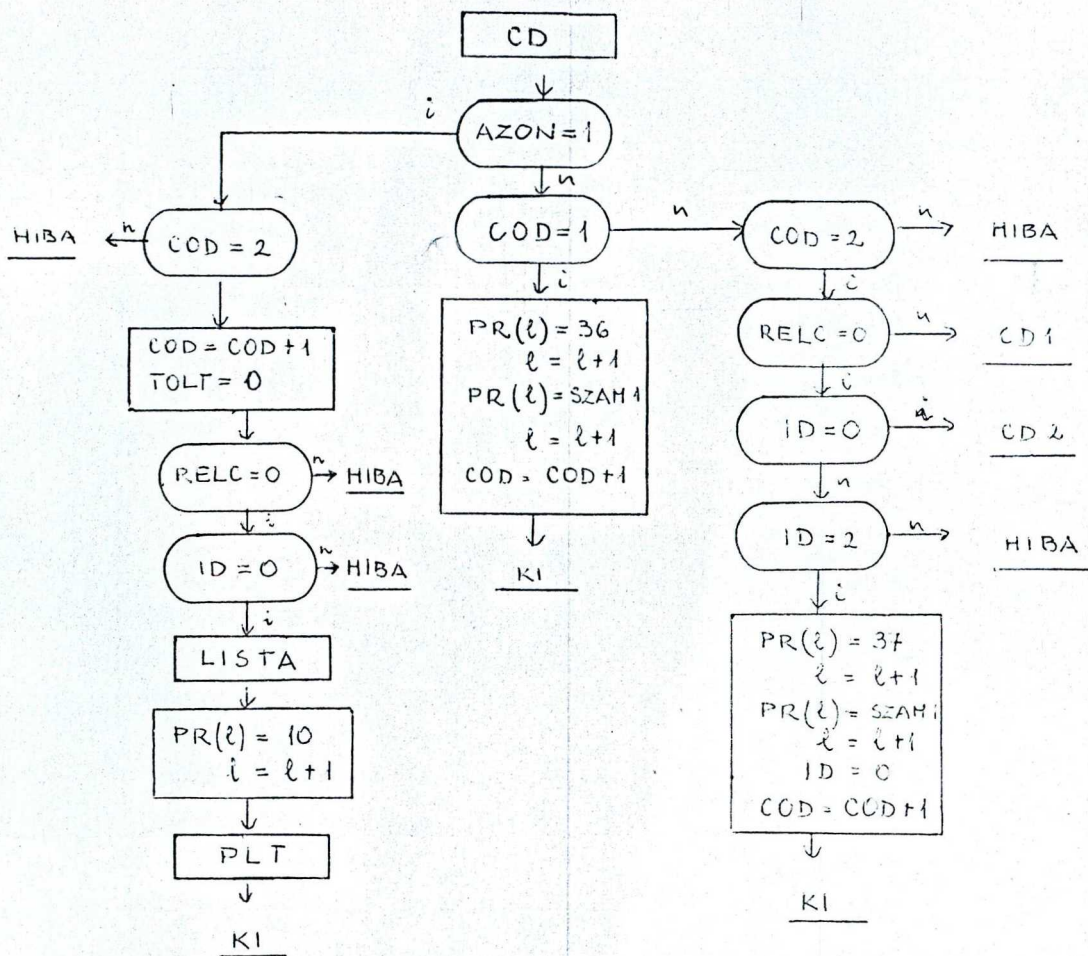


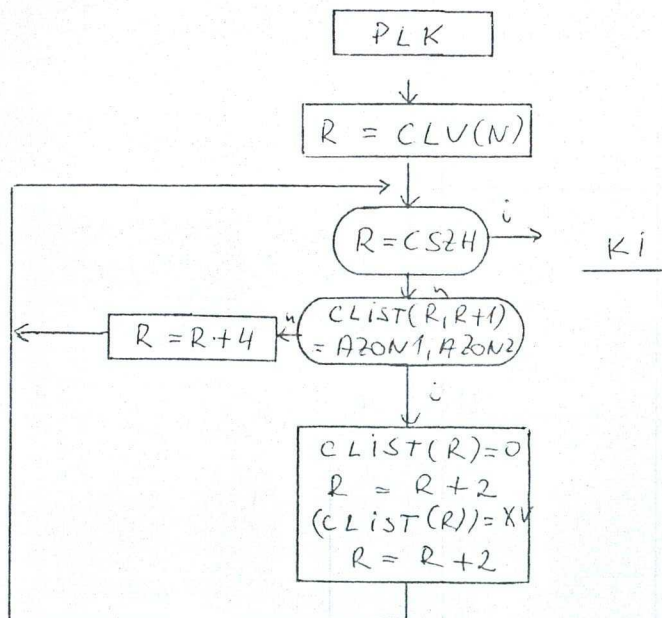
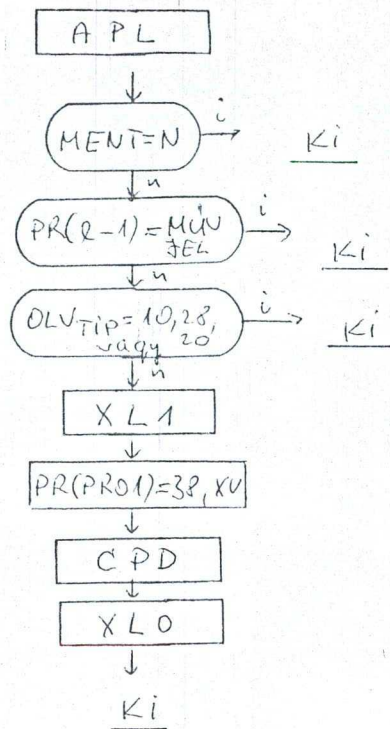




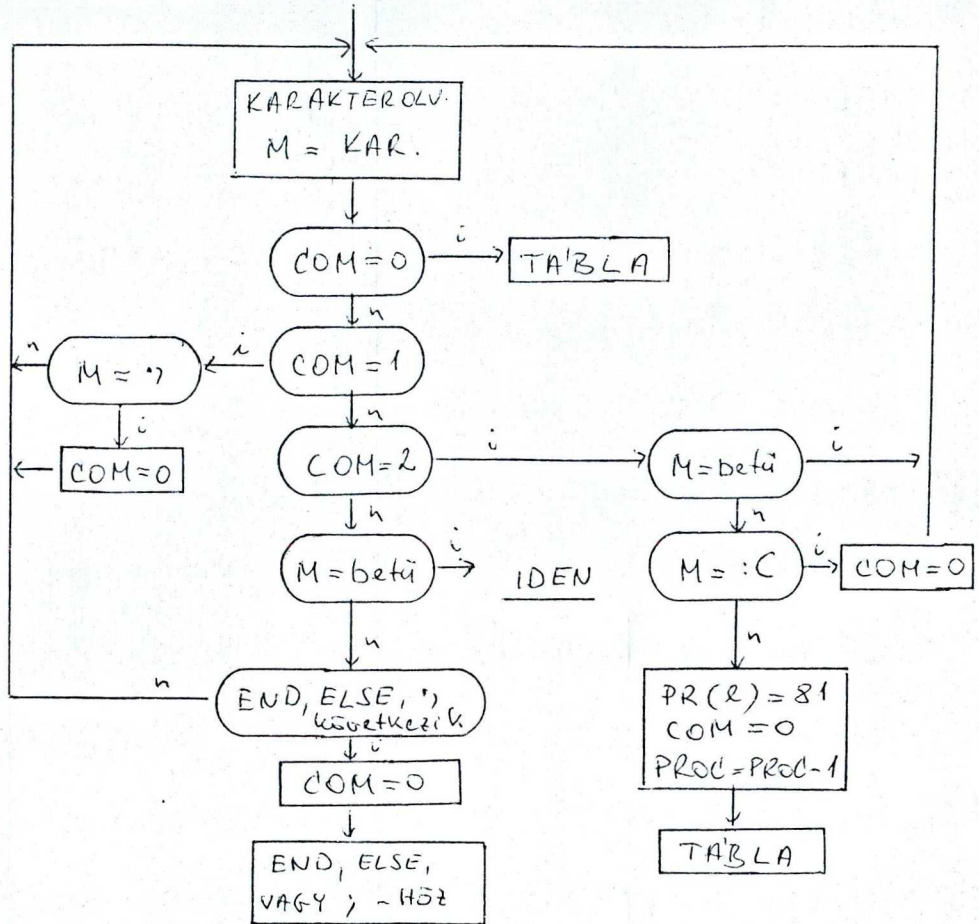




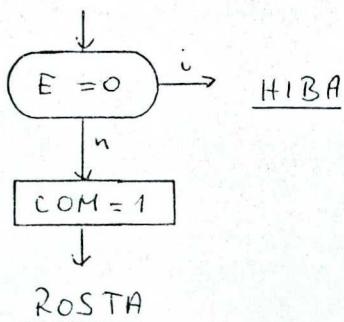




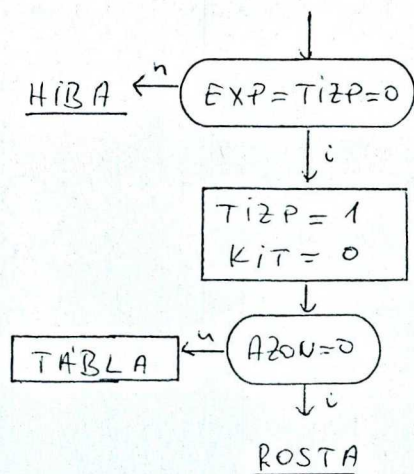
ROSTA

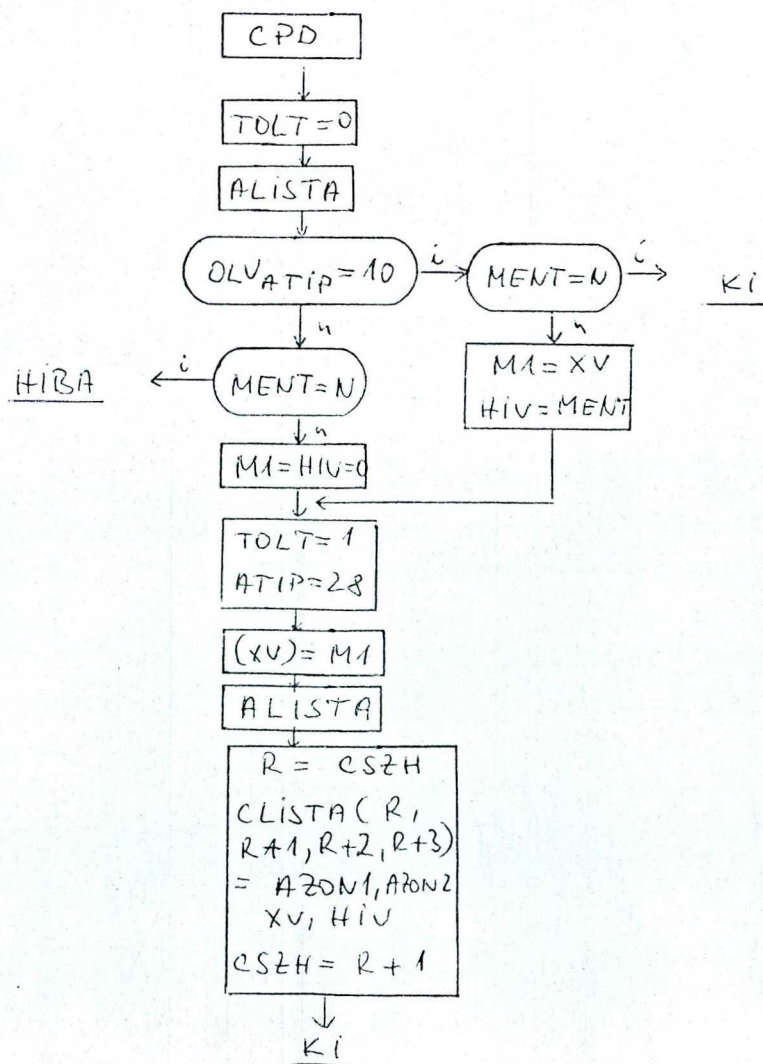
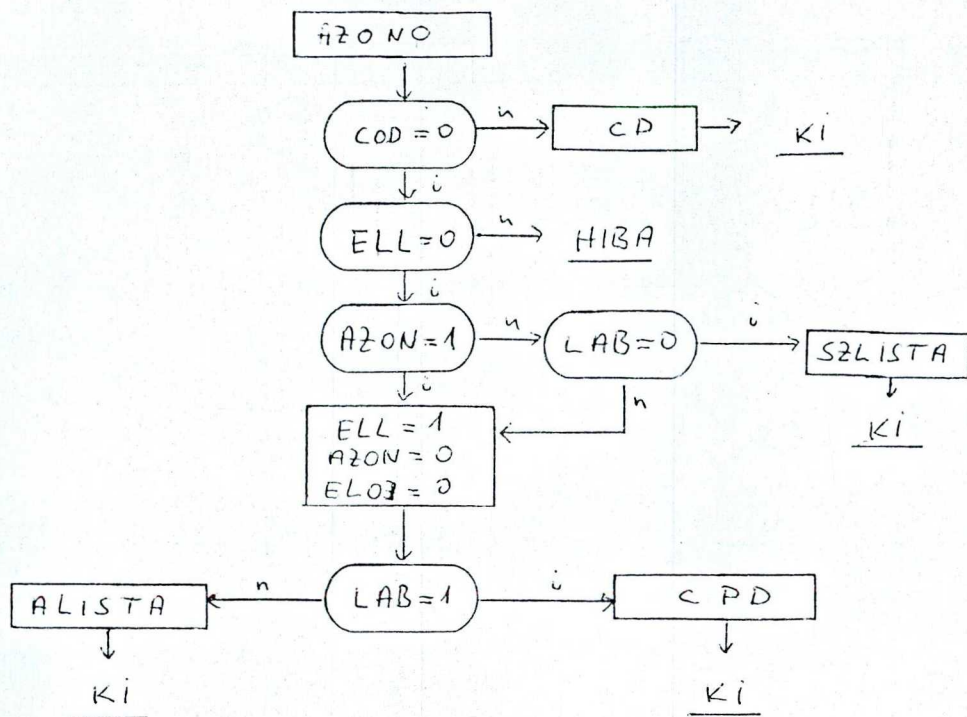


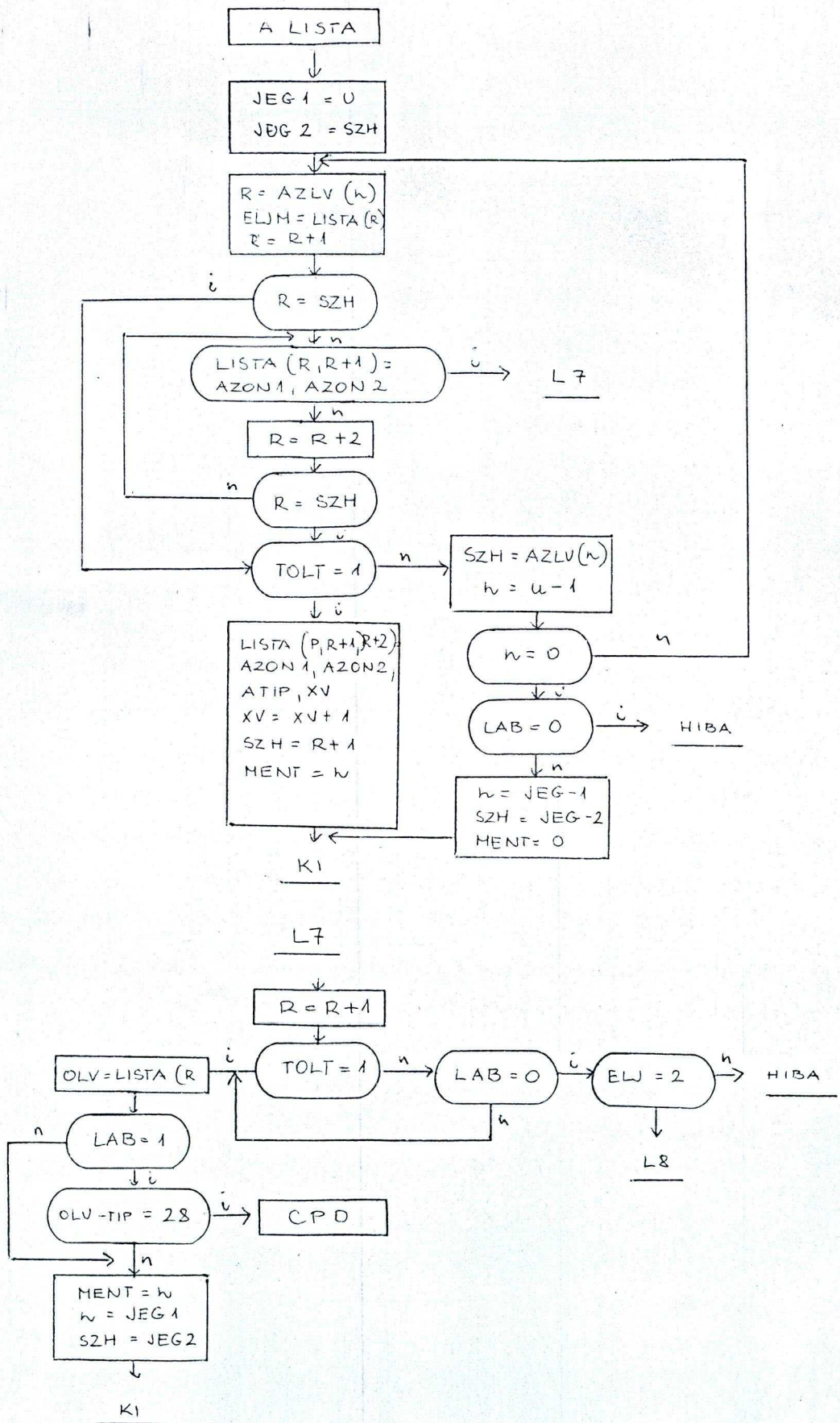
COMMENT

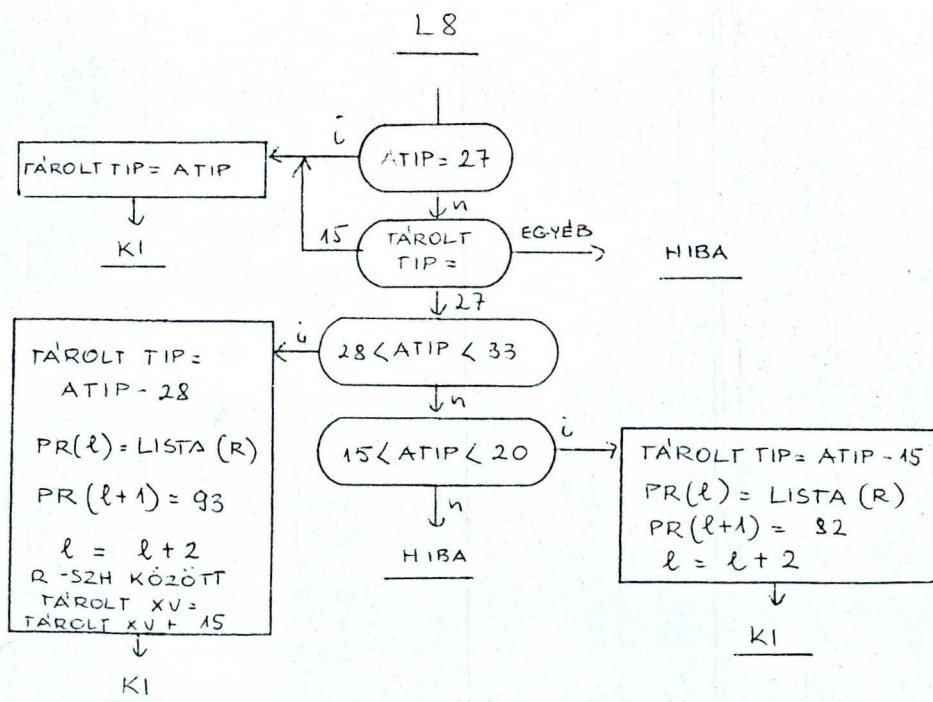
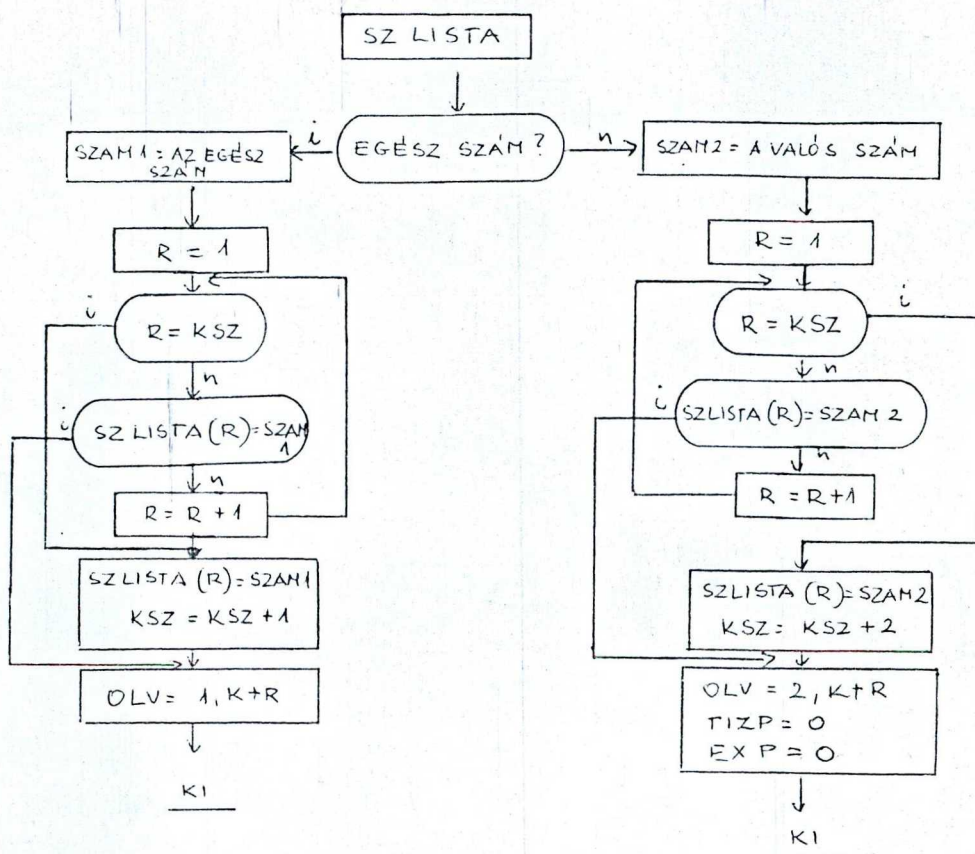


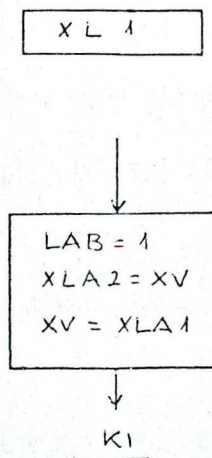
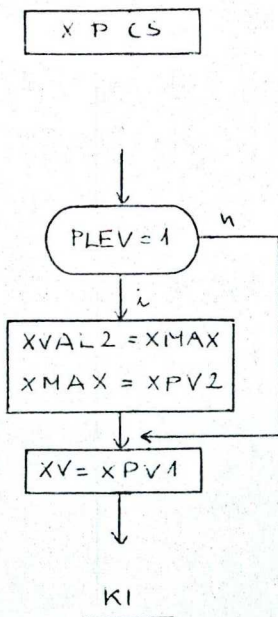
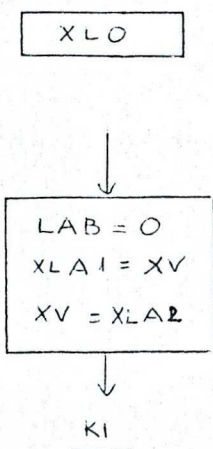
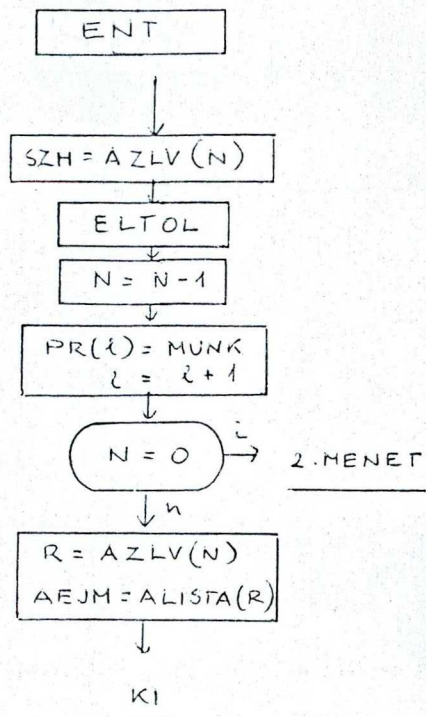
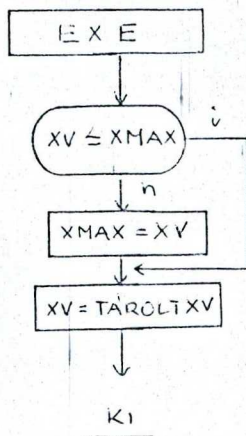
TIZEDES PONT











Második menet

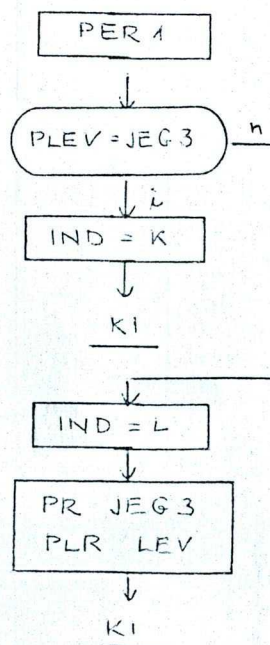
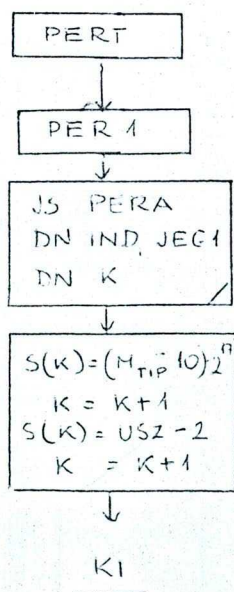
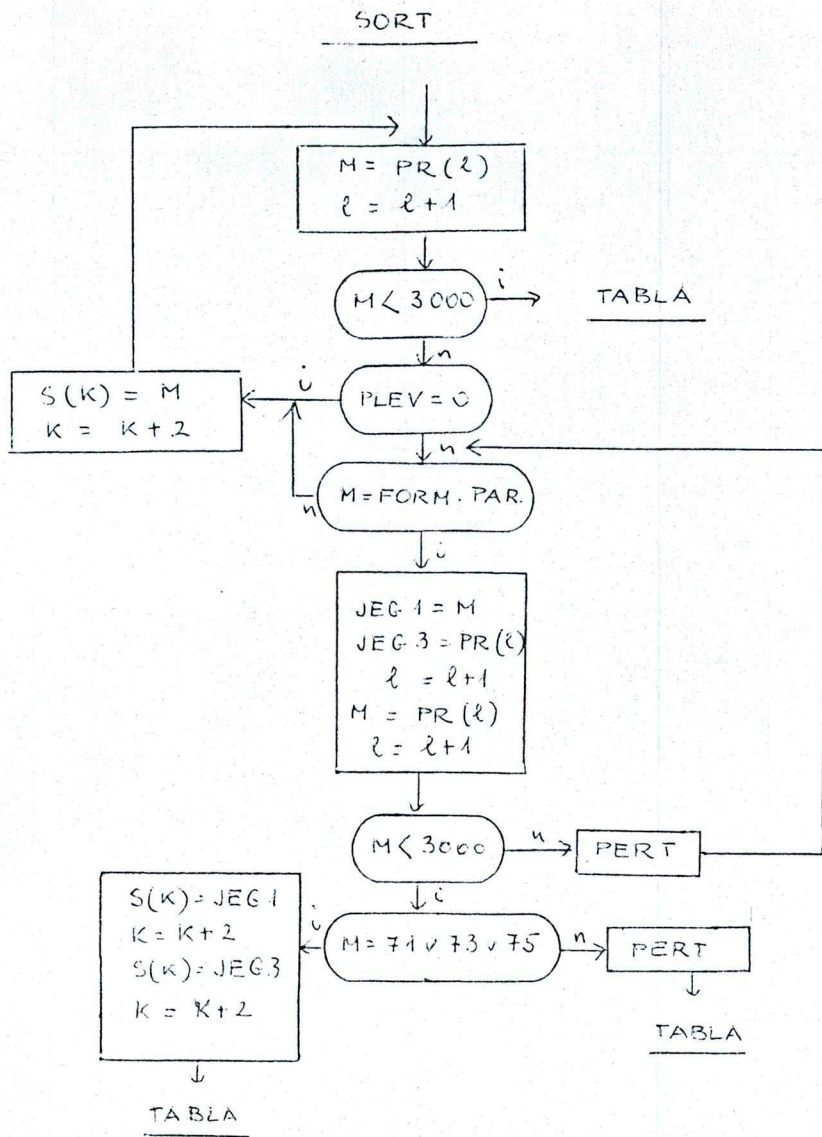
Az első menetben kialakított metanyelvű program kódszámaihoz rendre egy-egy rutin kapcsolódik.

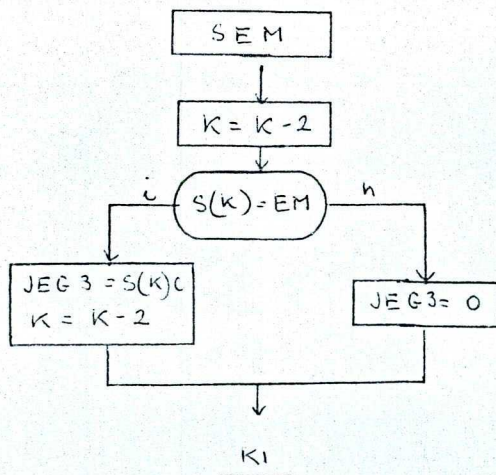
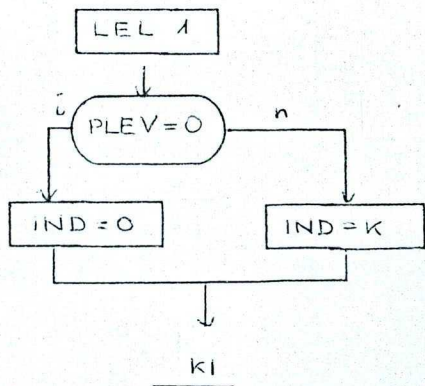
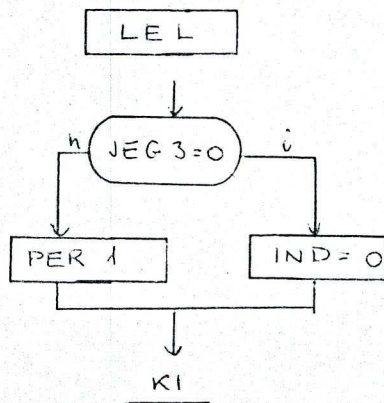
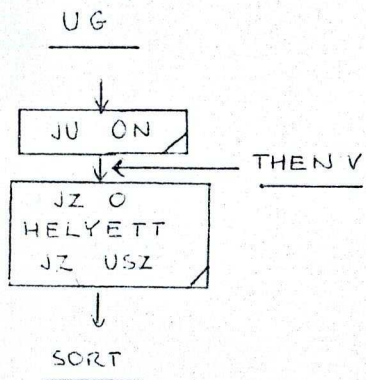
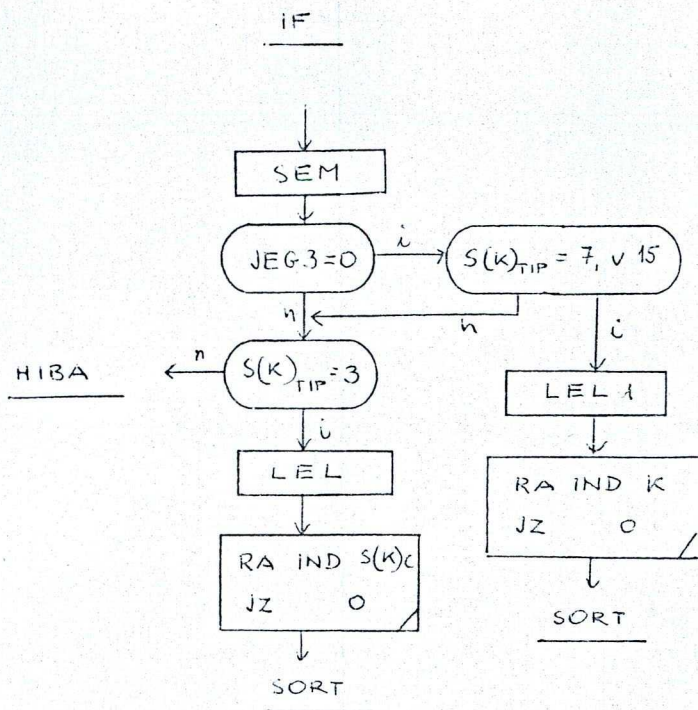
A kódszámok és rutinok a következők:

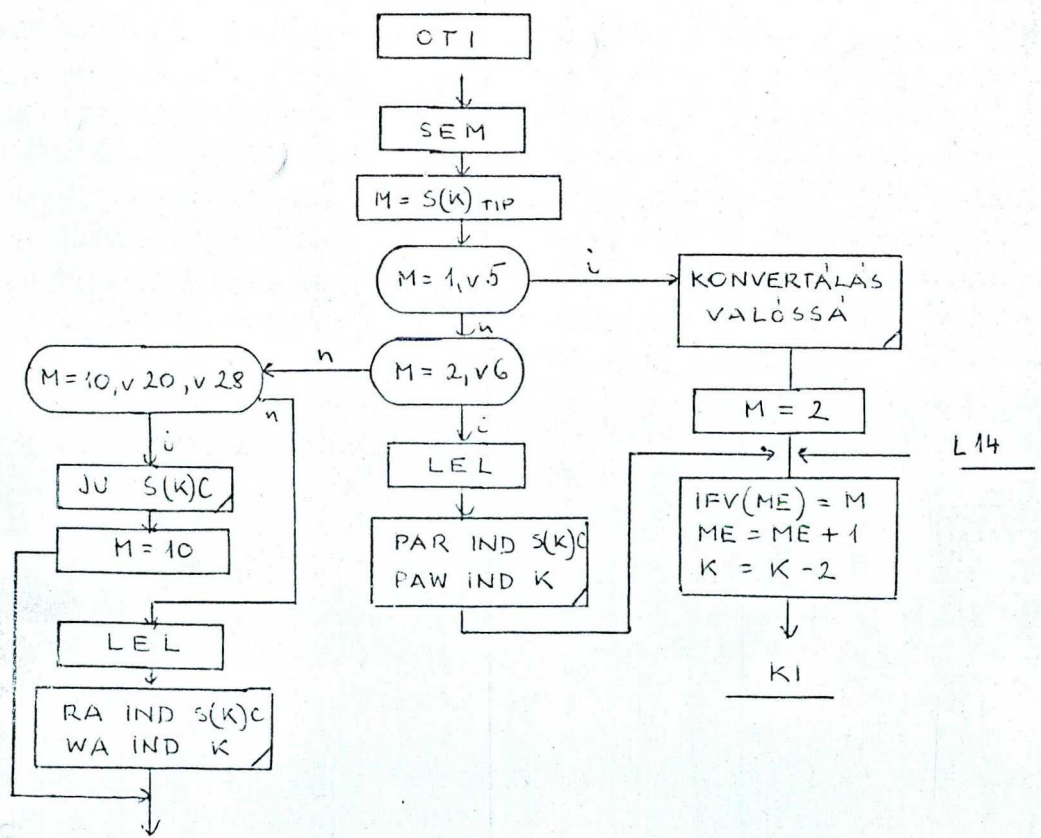
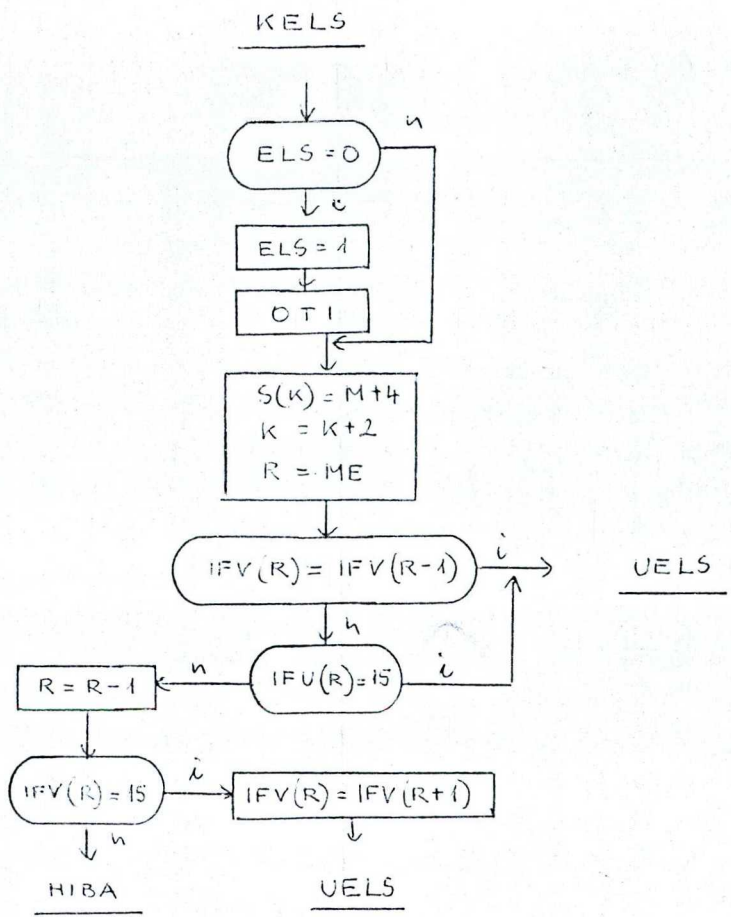
1	2	4	5	6
ZJ	KAP	uv	BEGIN	uw
9	10	11	12	13
R40	R41	sin	cos	ln
14	15	16	17	18
exp	sqrt	artg	abs	ent
19	20	21	22	23
sign	READ	WRITE	SPACE	LINE
24	26	27	28	29
PERIF	THENV	KELS	UELS	GOTO
30	31	32	33	34
ITA	TA	EQV	IMP	OR
35	36	37	38	39
AND	R42	R43	NOT	LT
40	41	42	43	44
LE	UEQ	EQ	GE	GT
45	46	47	48	49
+	-	*	/	DIV
50	51	52	53	54
NEG		IF	uG	OTI
55	56	57	58	59
FOR	FL	FE	FS	PSL
60	61	62	63	64
FA	FW	CE	FD	TRUE
65	66	67	68	69
FALSE	LABEL	FR	OTOM	TOM
70	72	73	74	75
EM	SW BEGIN	PE	SWV	EB
76	78	79	80	81
PERK	END	FB	SWEND	EBEND
82	85	90	91	92
VNA	LW	PAUSE	STOP	PRE

93	94	96	97	98
VNAT	EBF	RED	WIT	INS
99	100	101	102	103
OUT	MT	MD	TU	DA
105	106			
CI	CO			

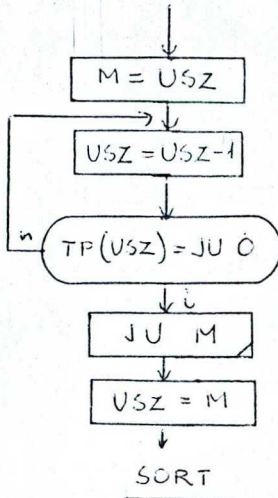
A blokkdiagrammokban a gépi kódú utasítások kialakítását a téglalap jobb alsó sarkában lévő törtvonal jelzi.



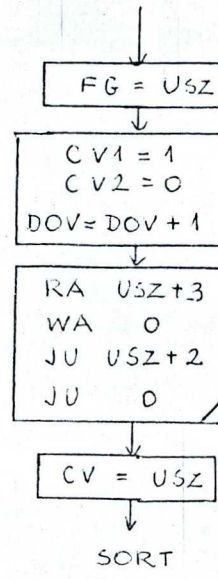




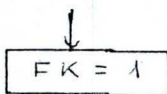
UELS



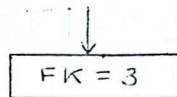
FOR



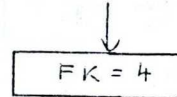
FS



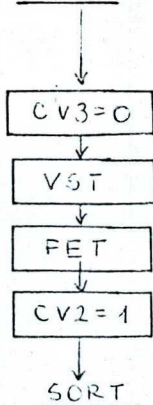
FA



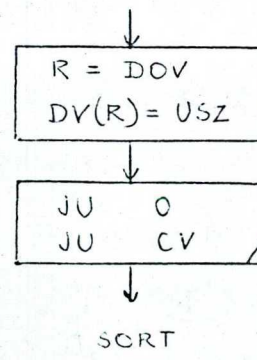
FW

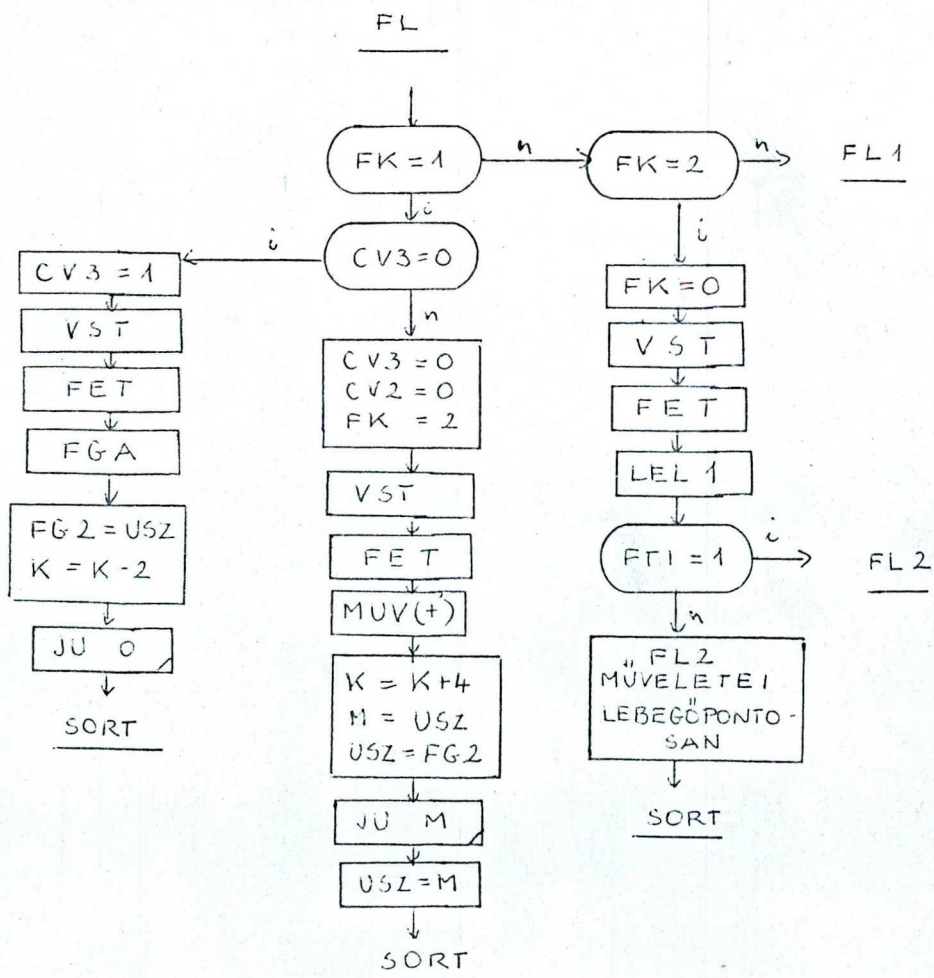
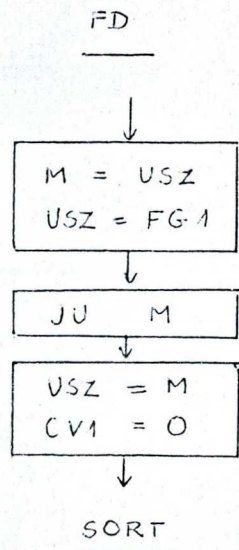
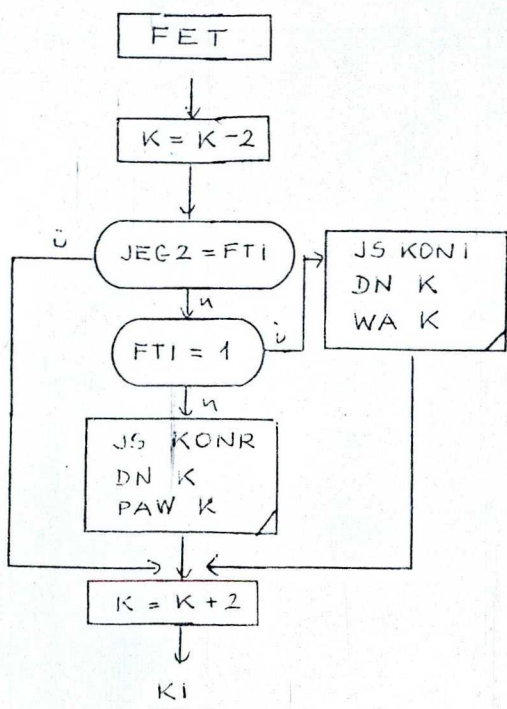


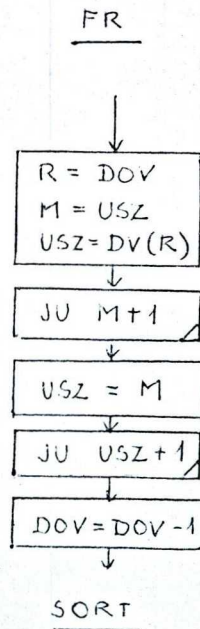
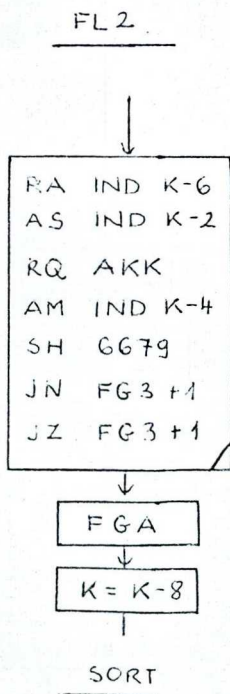
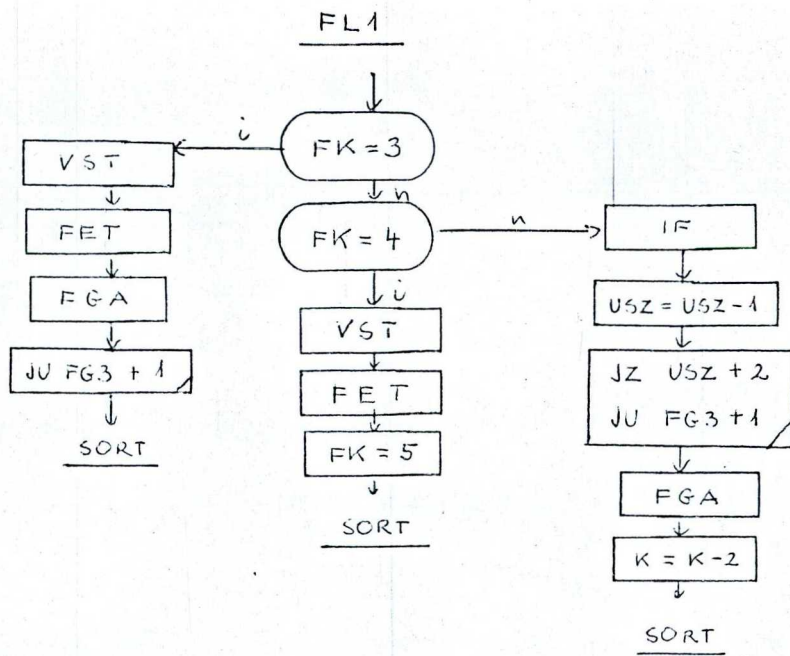
FS1

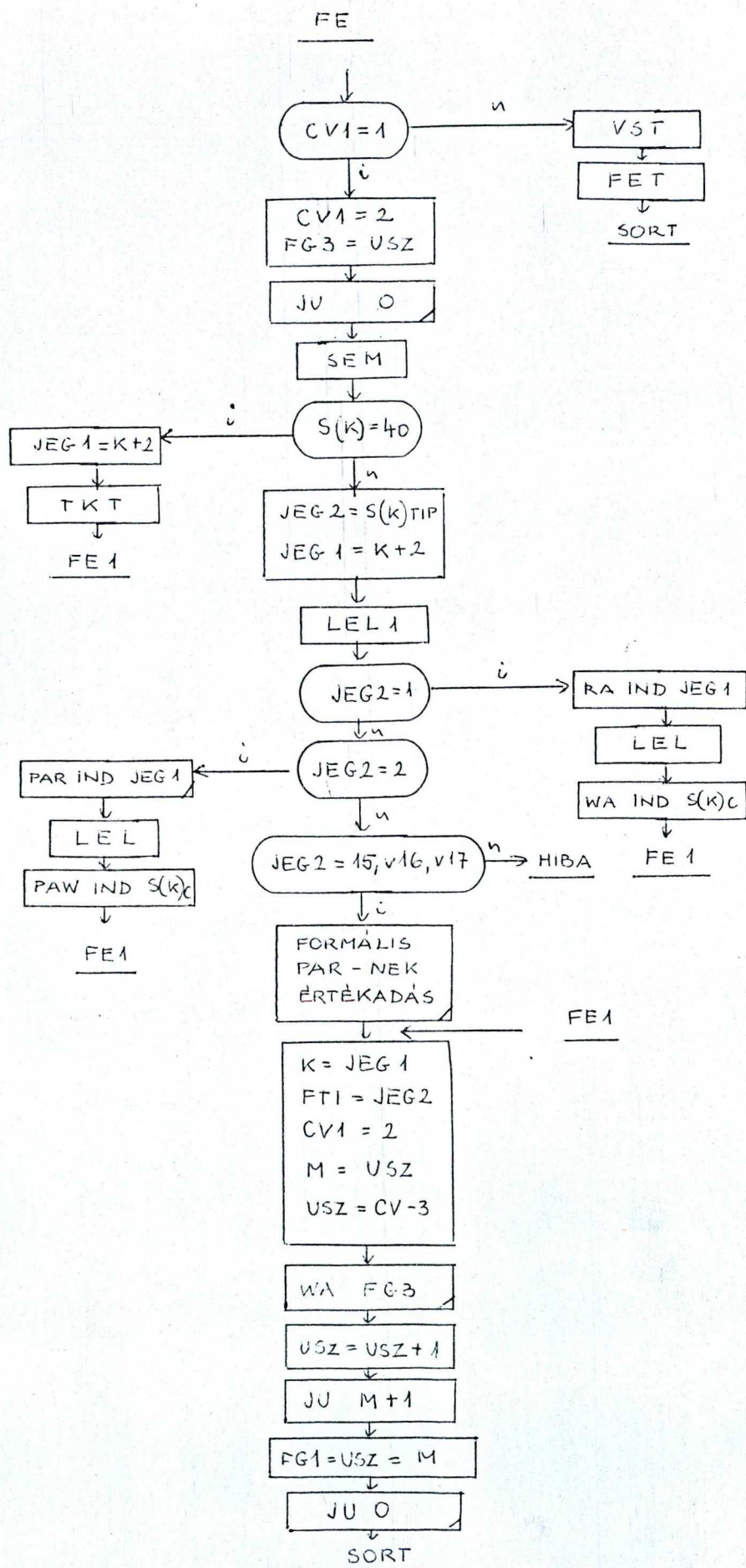


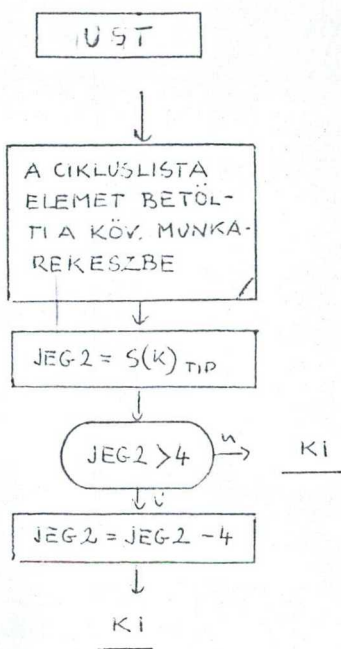
CE



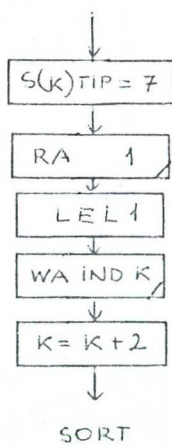




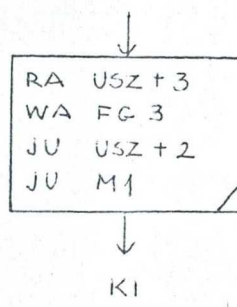




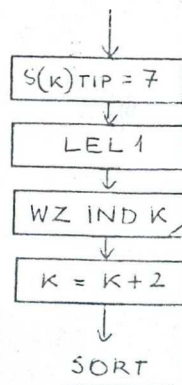
TRUE



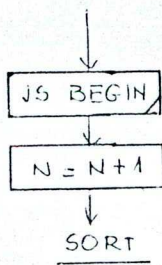
FGA



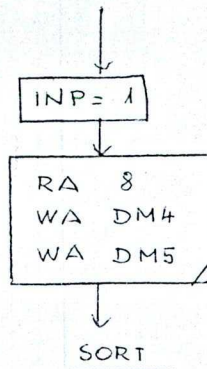
FALSE



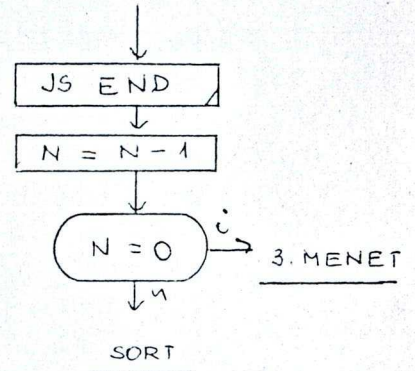
BEGIN



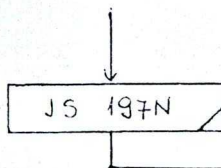
WIT



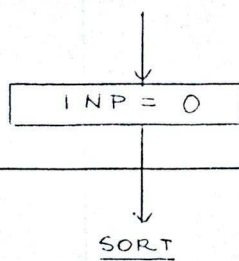
END



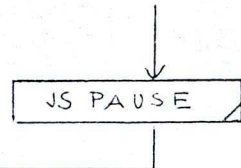
STOP



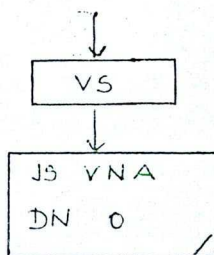
RED



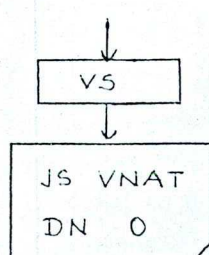
PAUSE



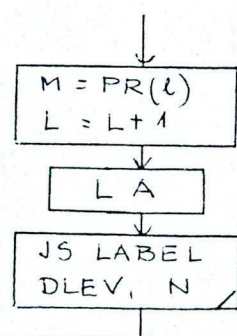
VNA



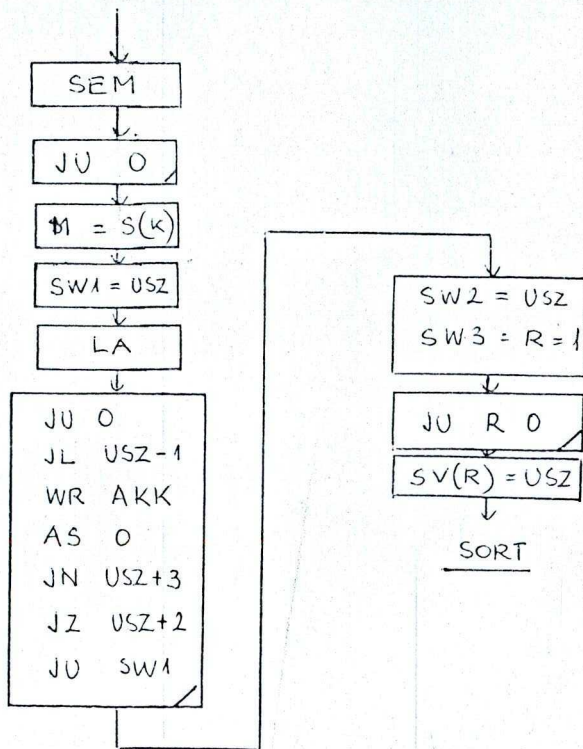
VNAT



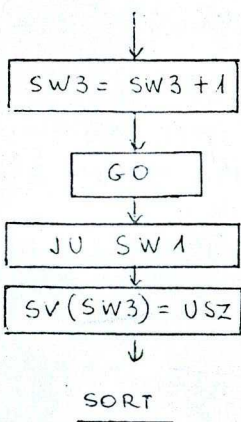
LABEL



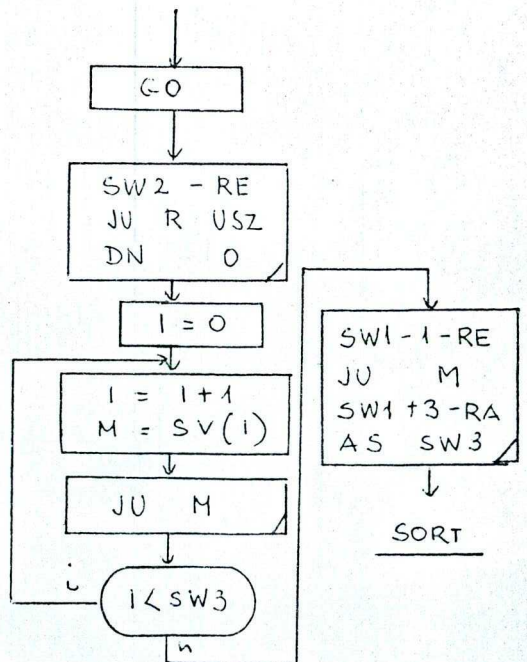
SW BEGIN

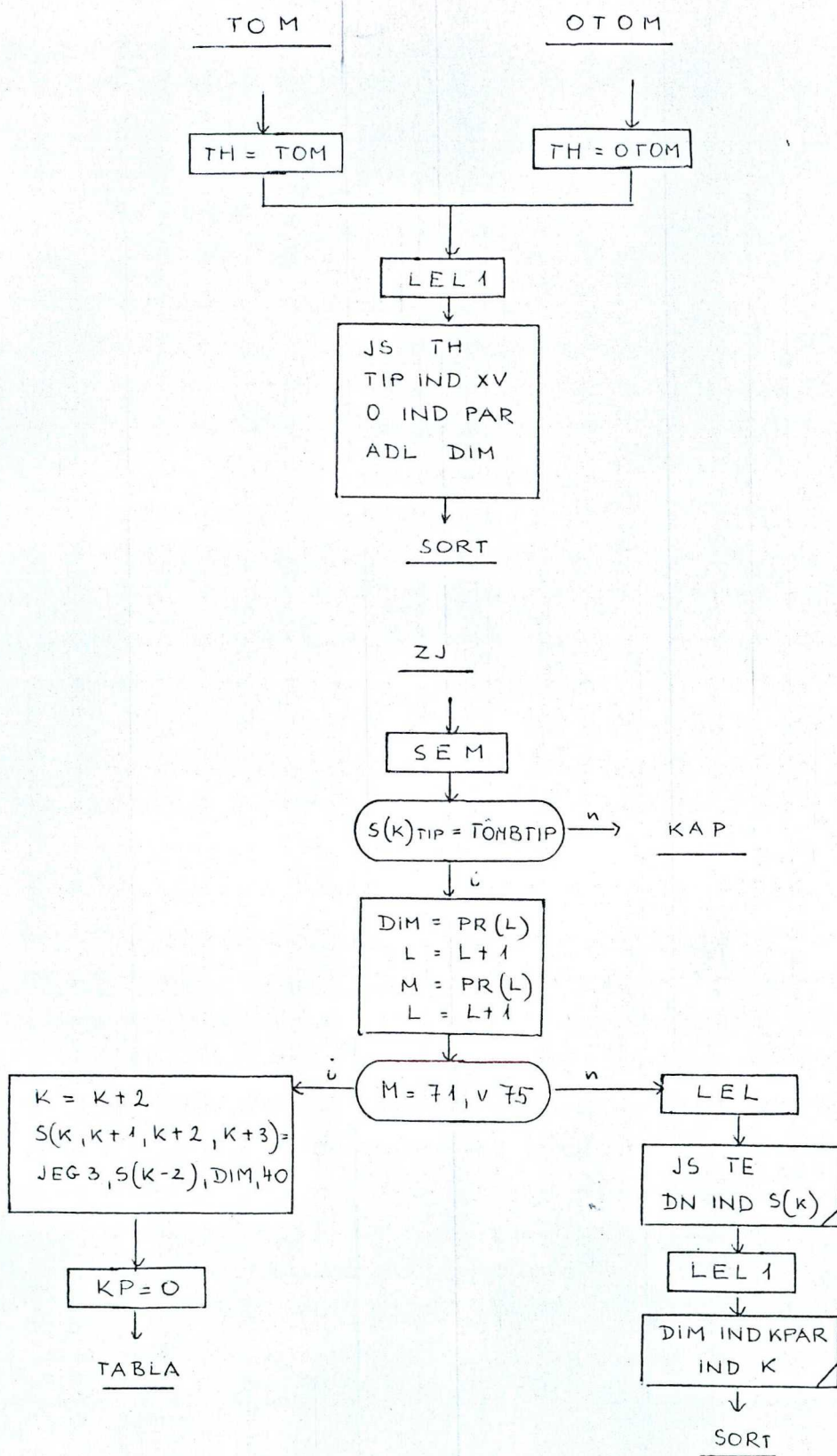


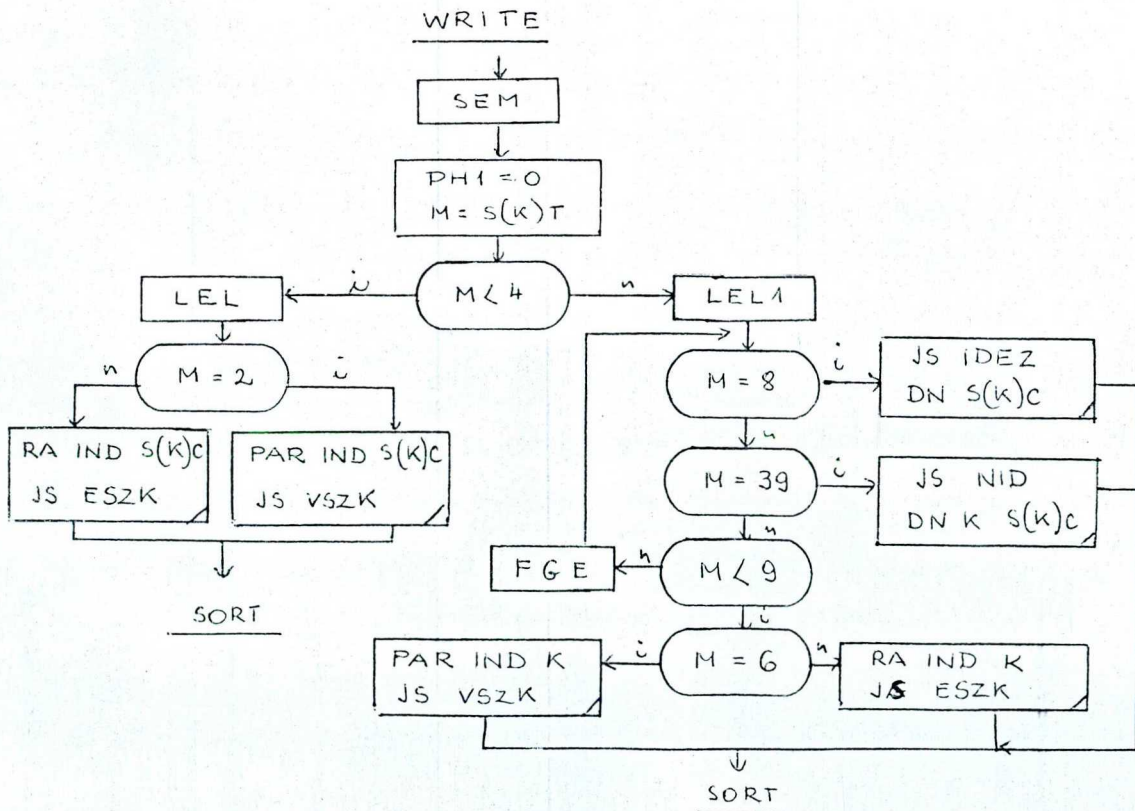
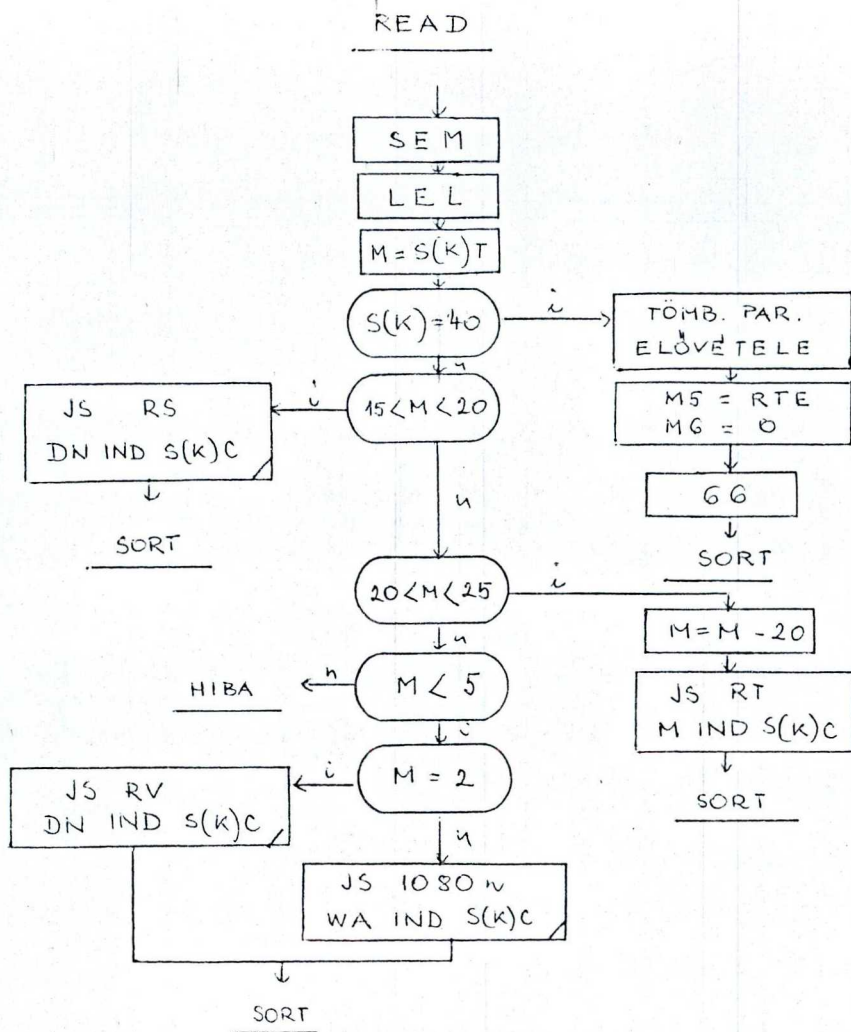
SWV

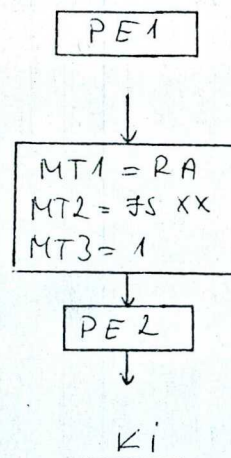
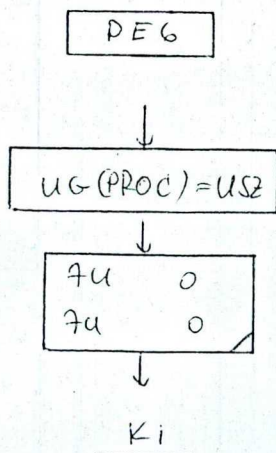
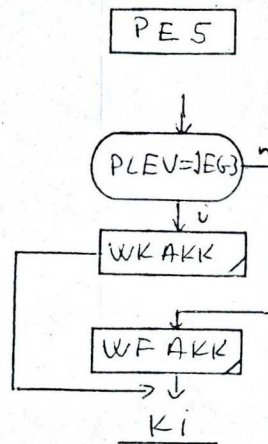
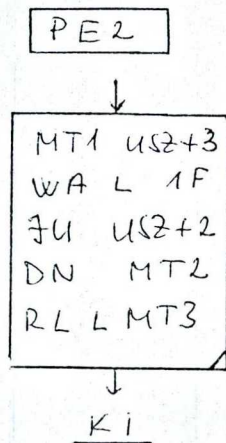
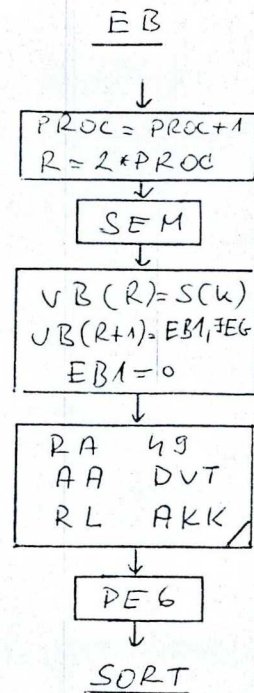
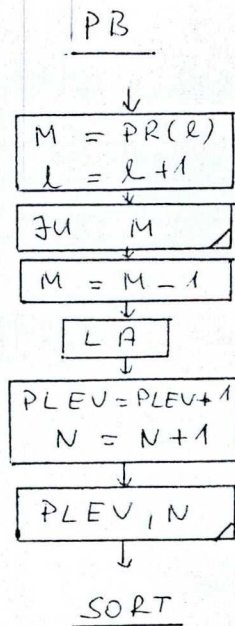


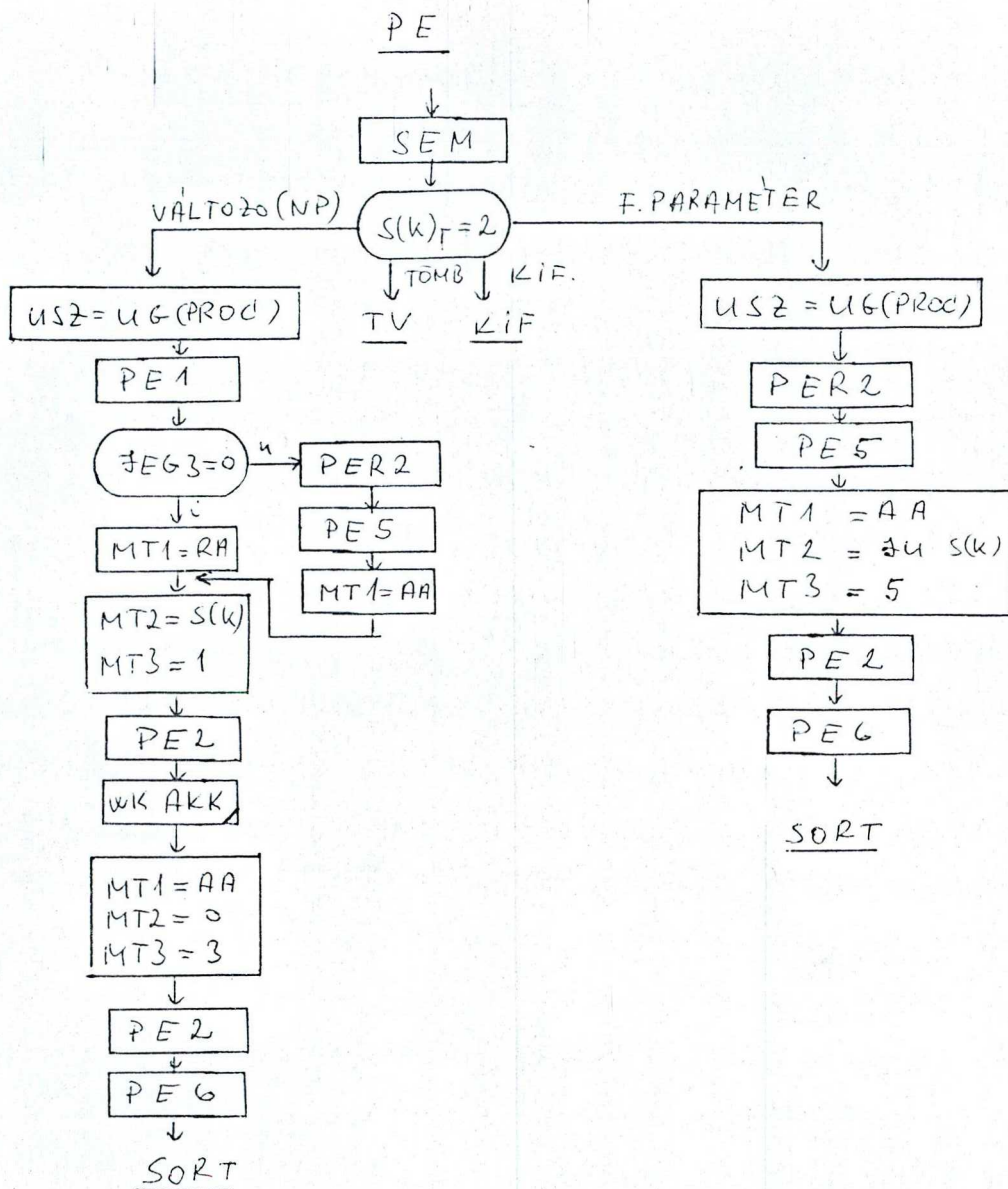
SWEND



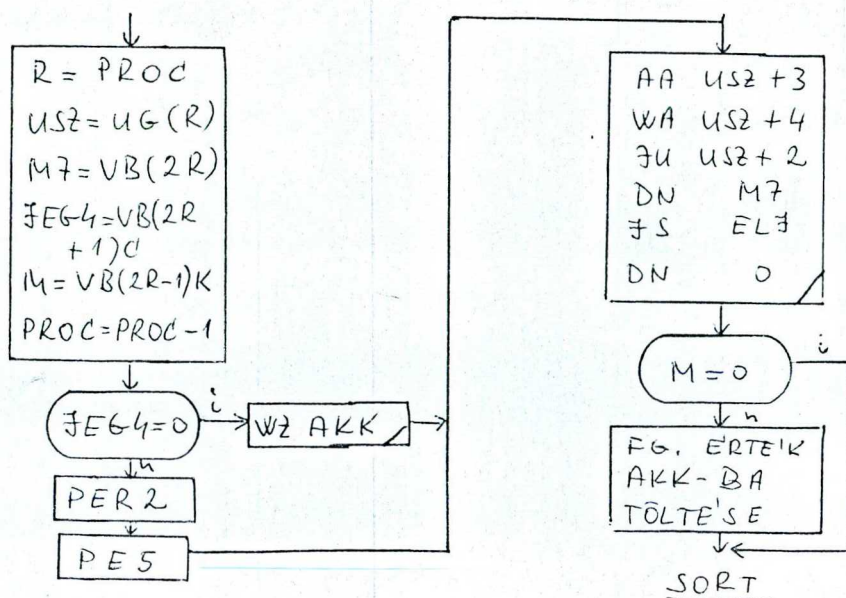


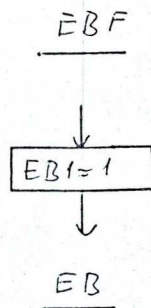
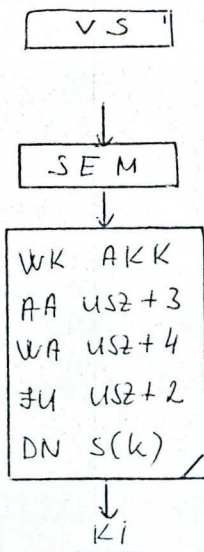
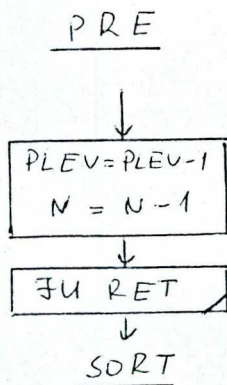
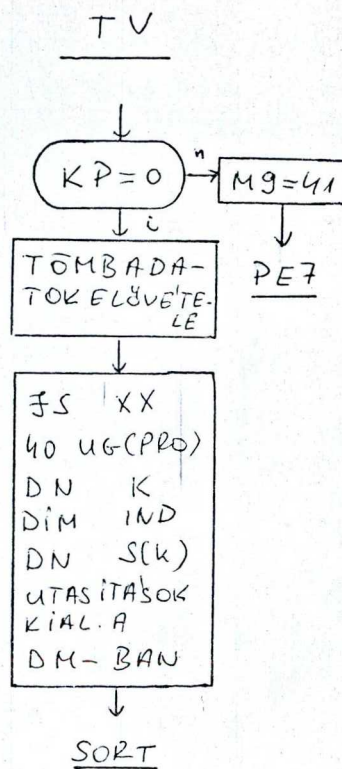
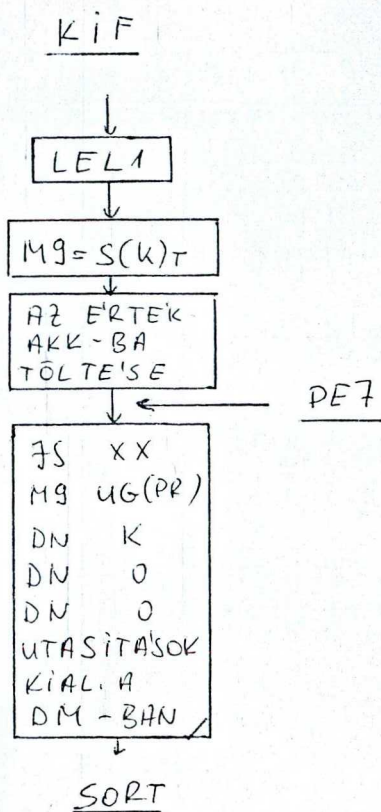




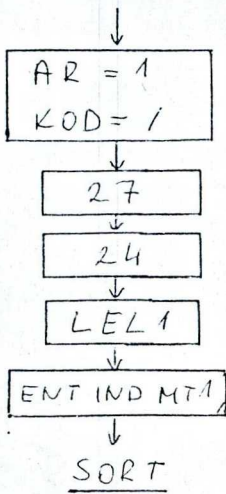


EBEND

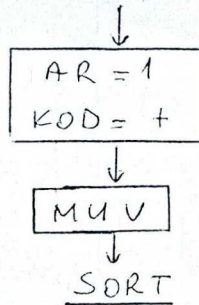




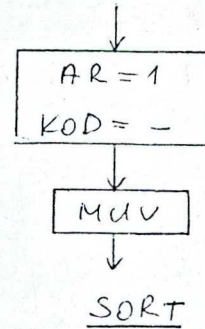
DIV



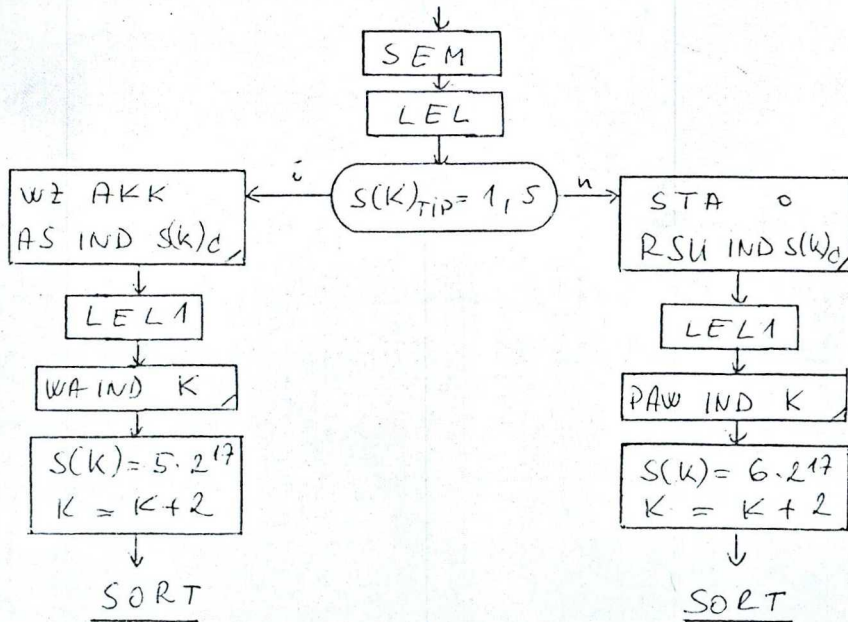
+



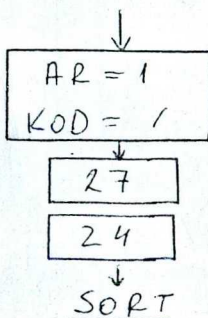
-



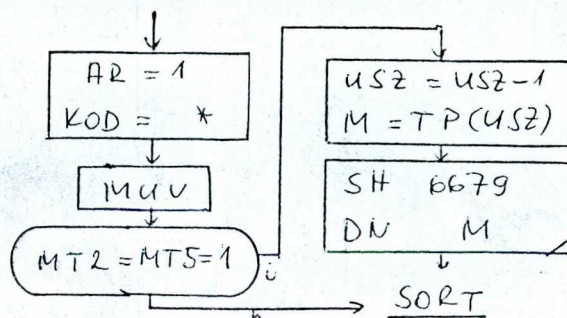
NEG

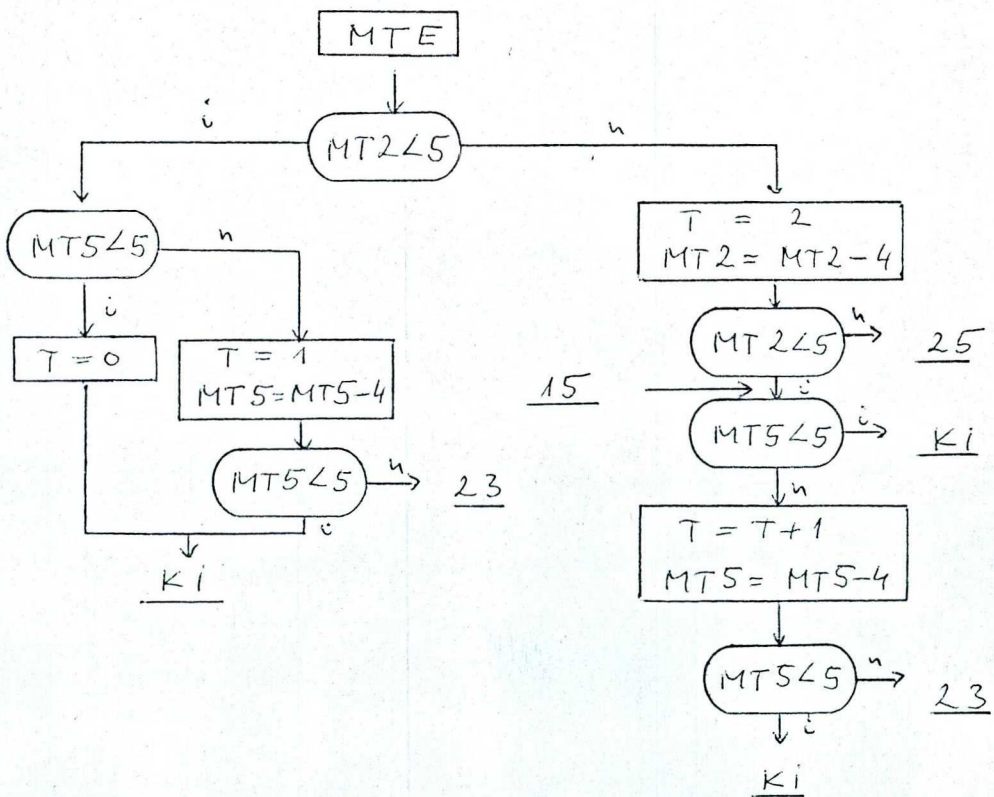
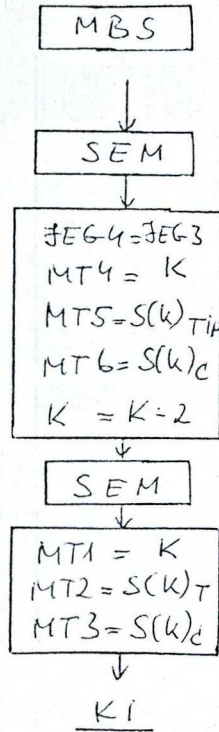
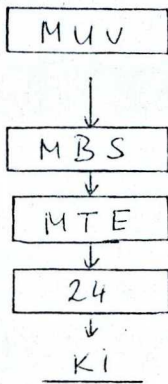


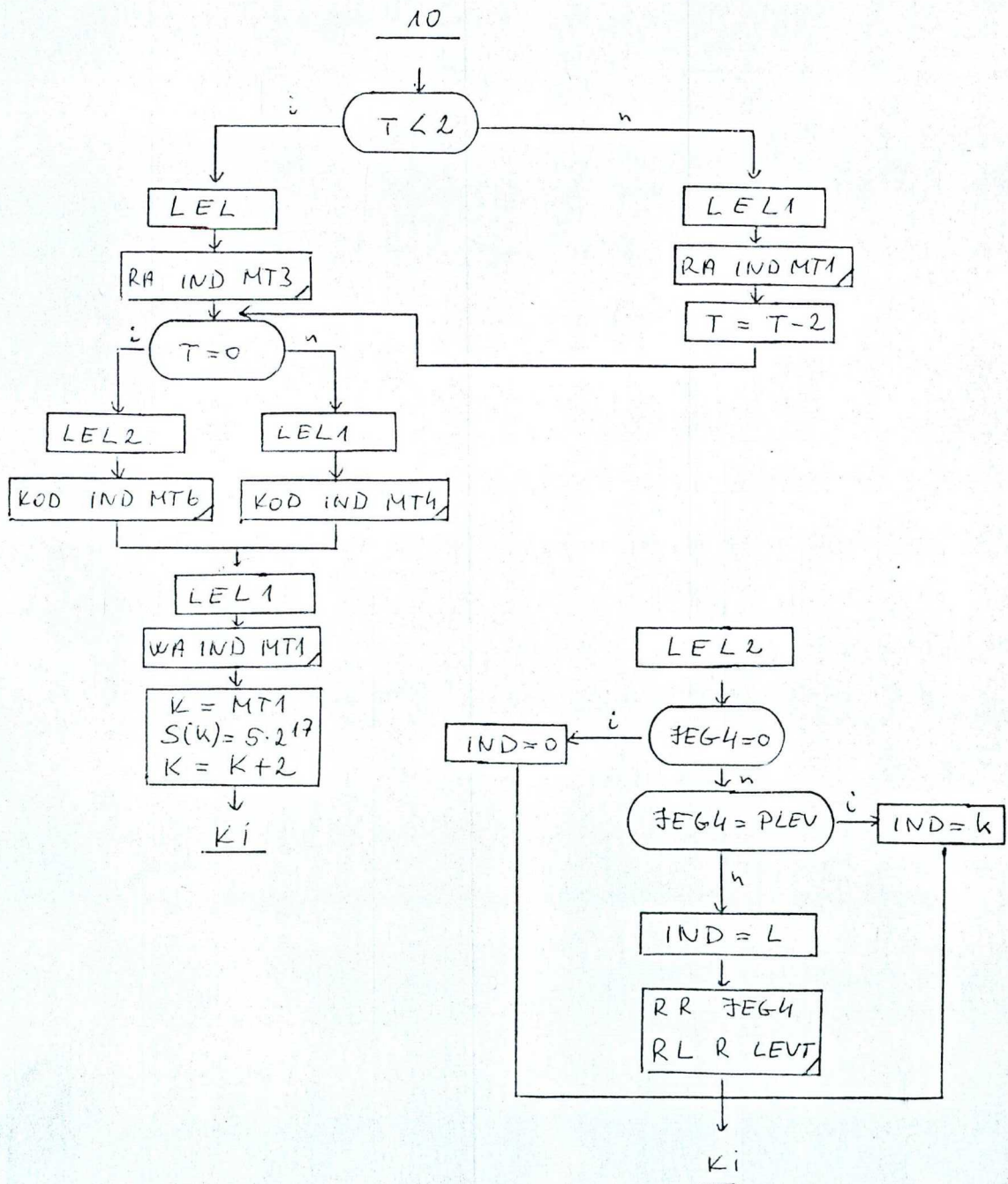
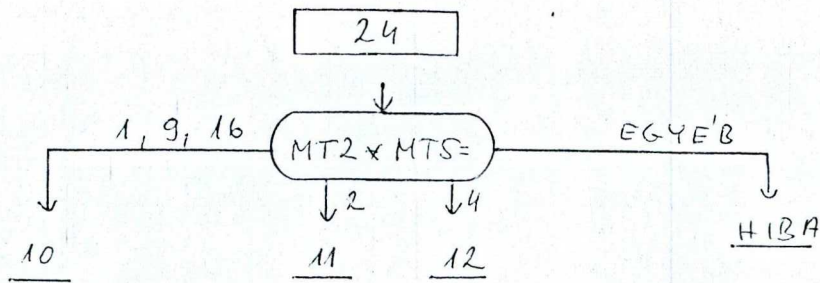
/

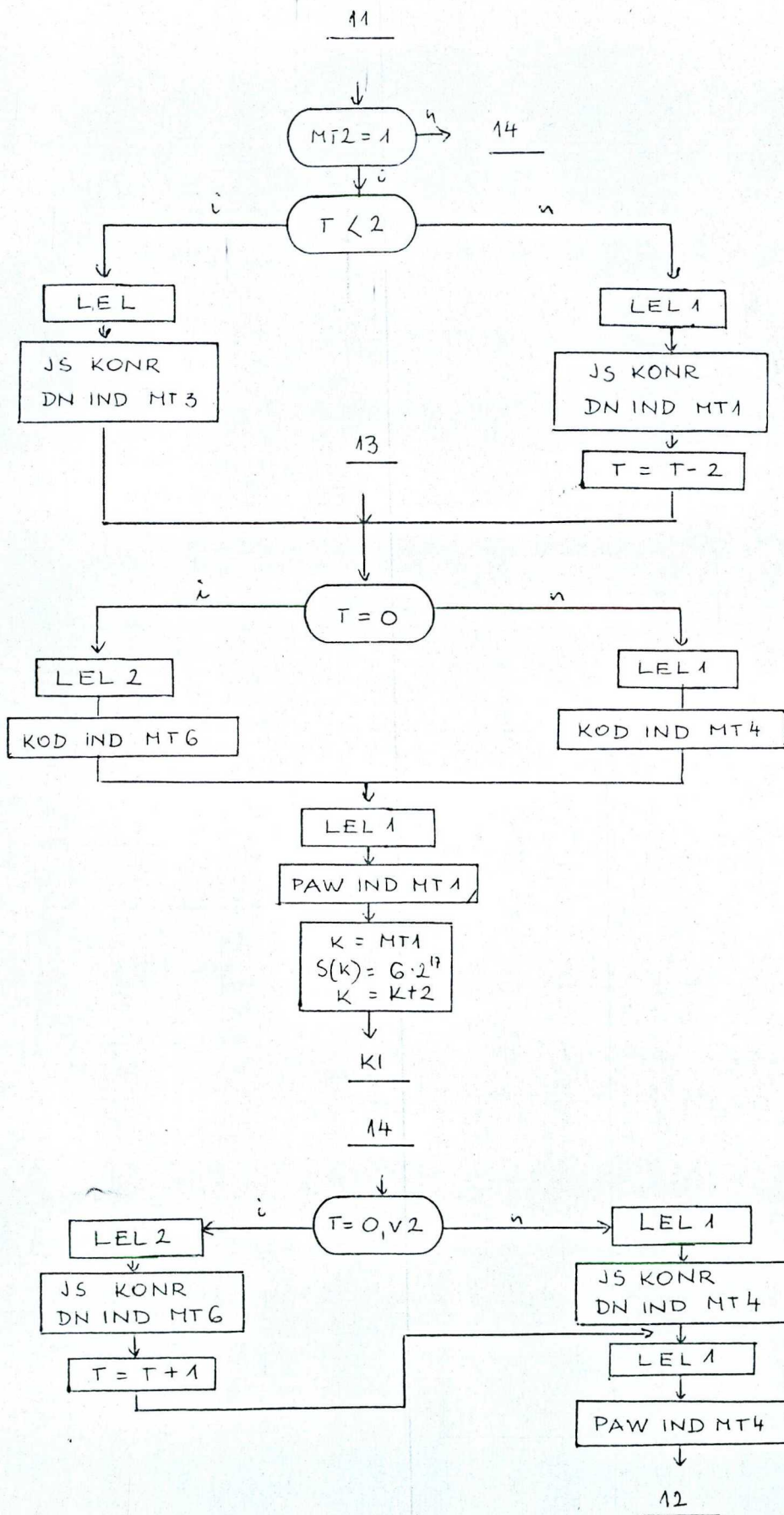


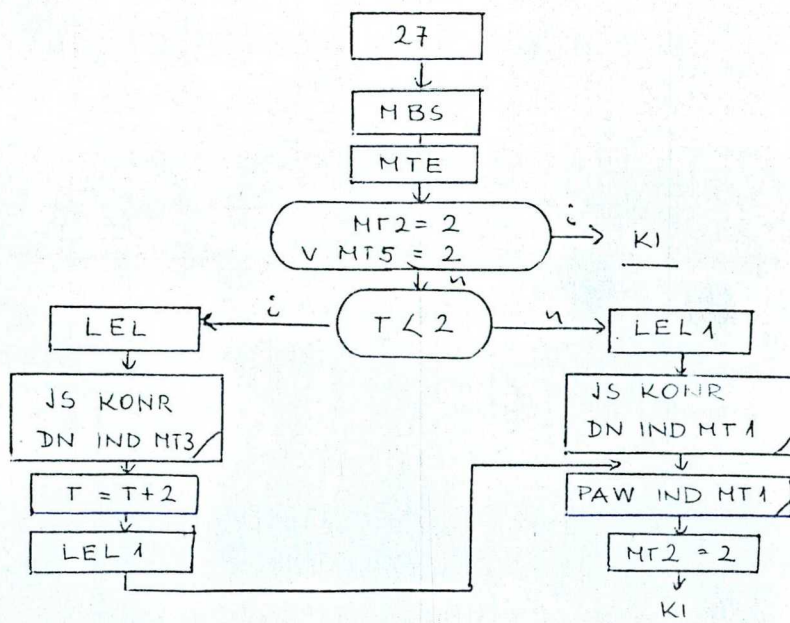
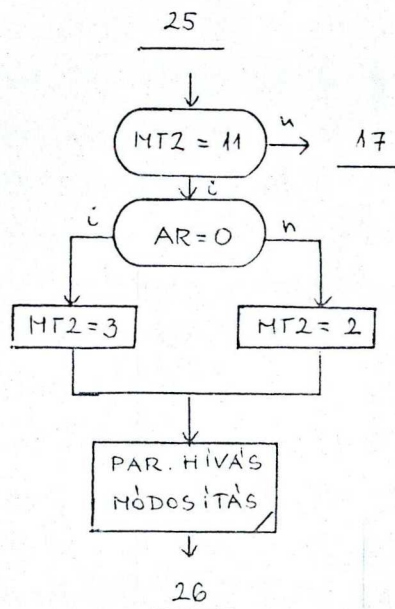
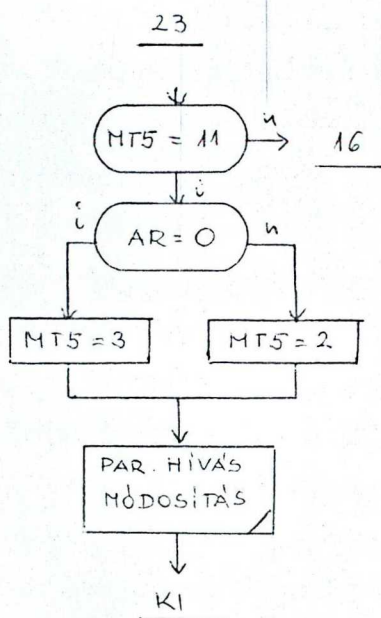
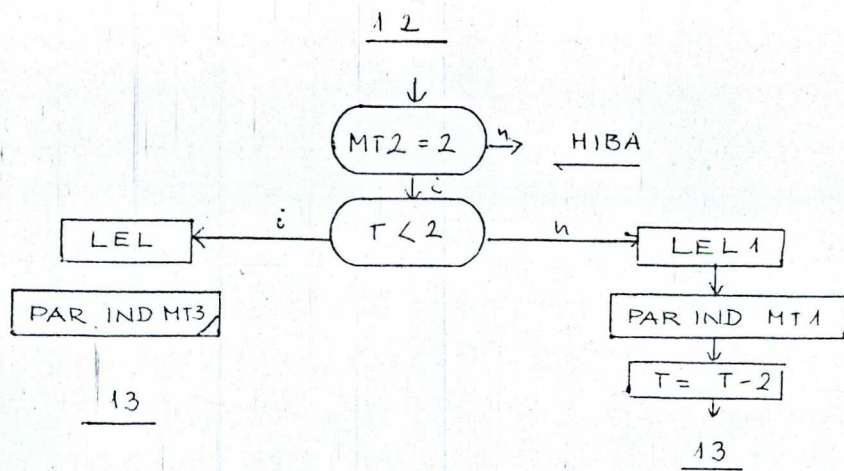
*

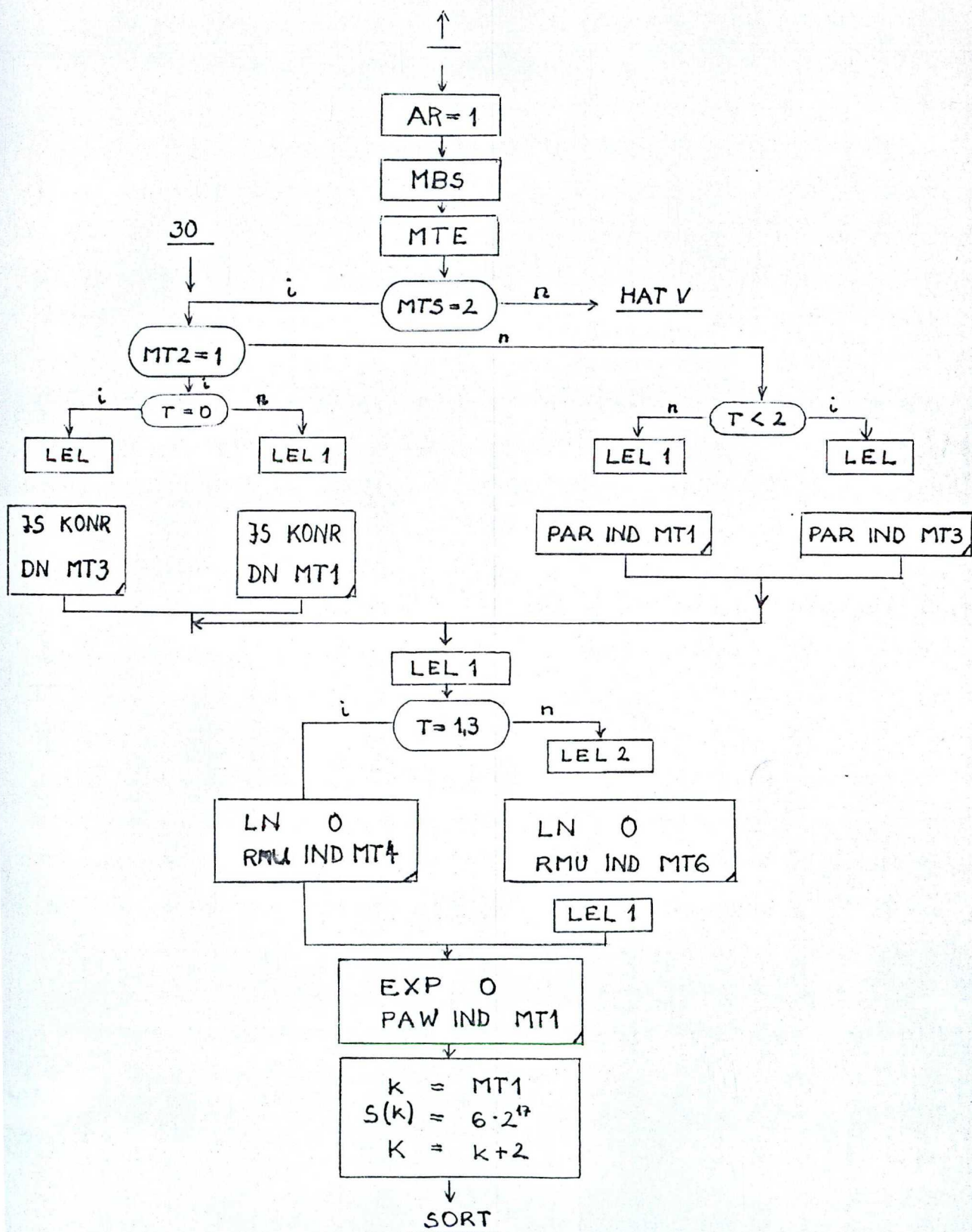


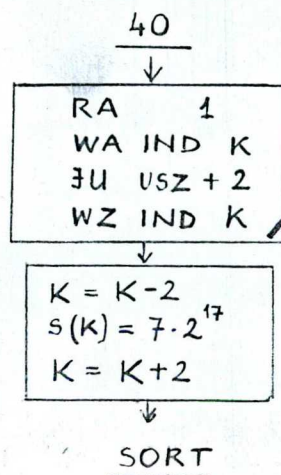
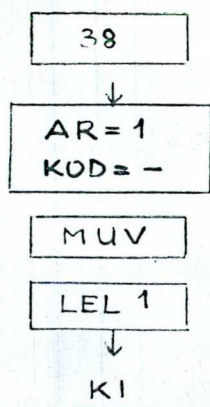
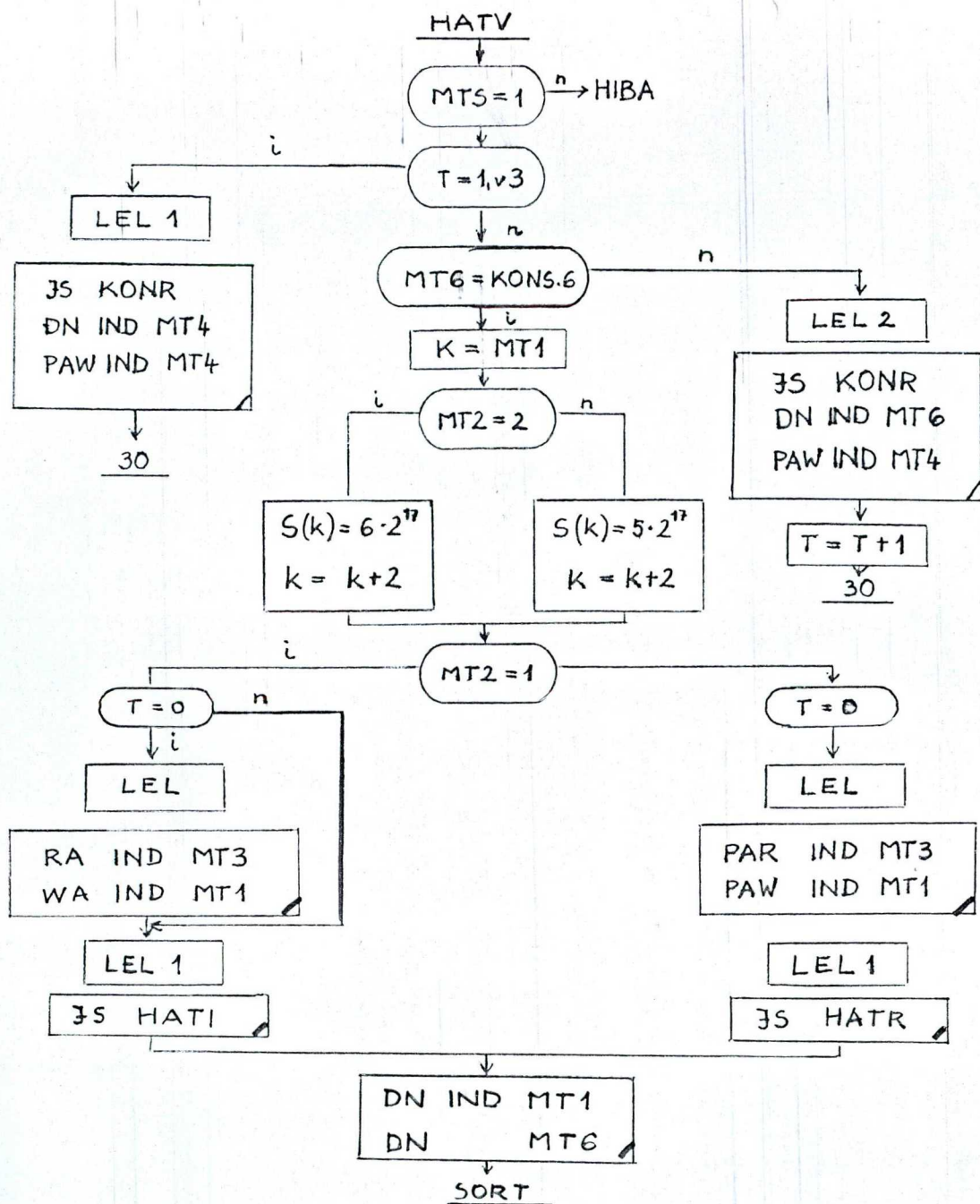




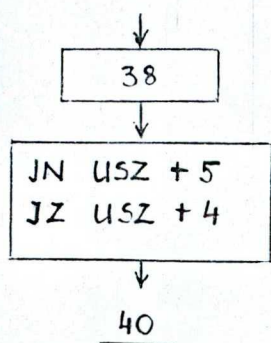




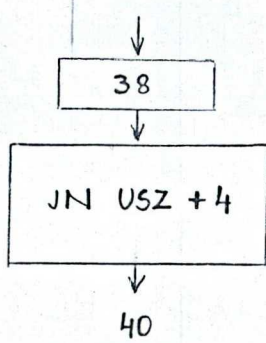




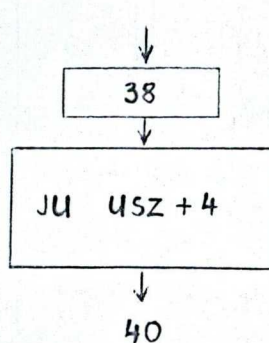
GT



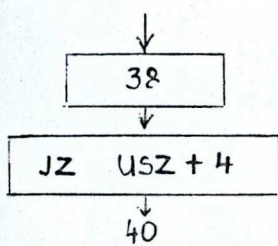
GE



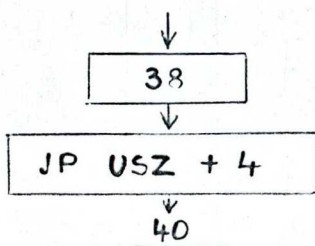
EQ



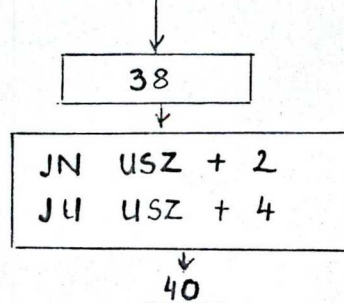
UEQ



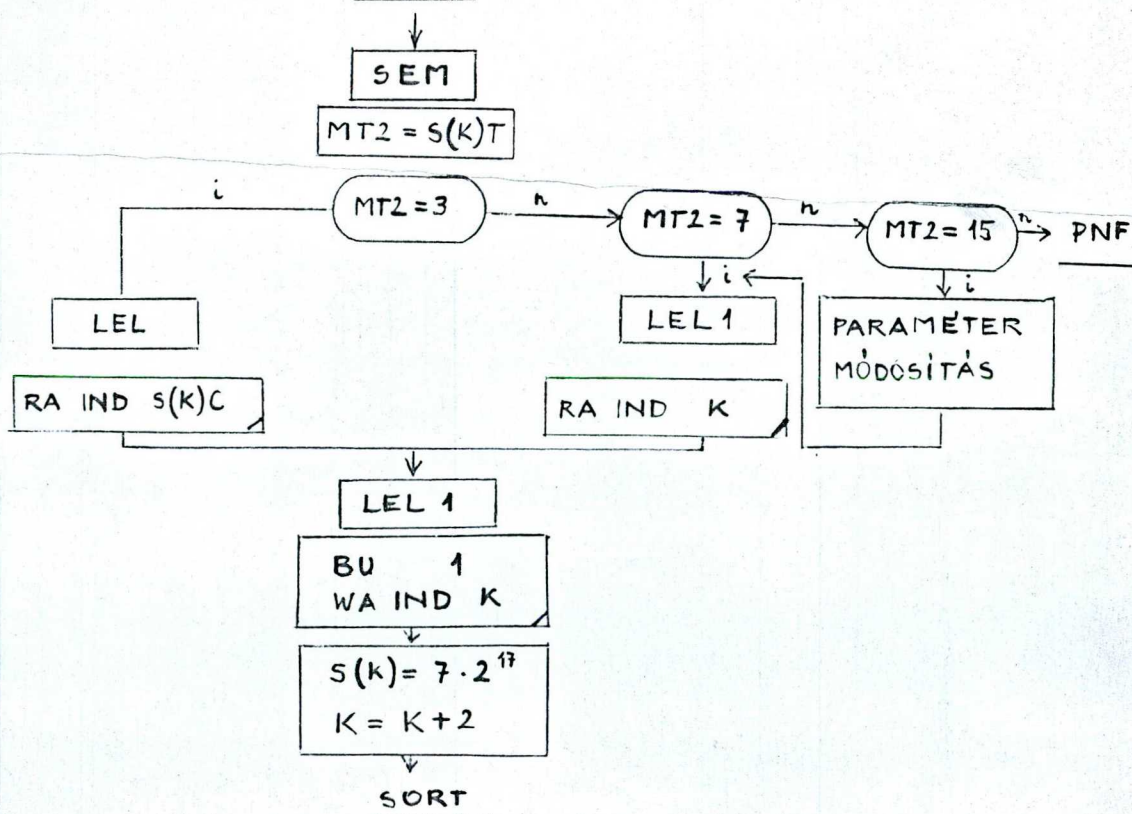
LE

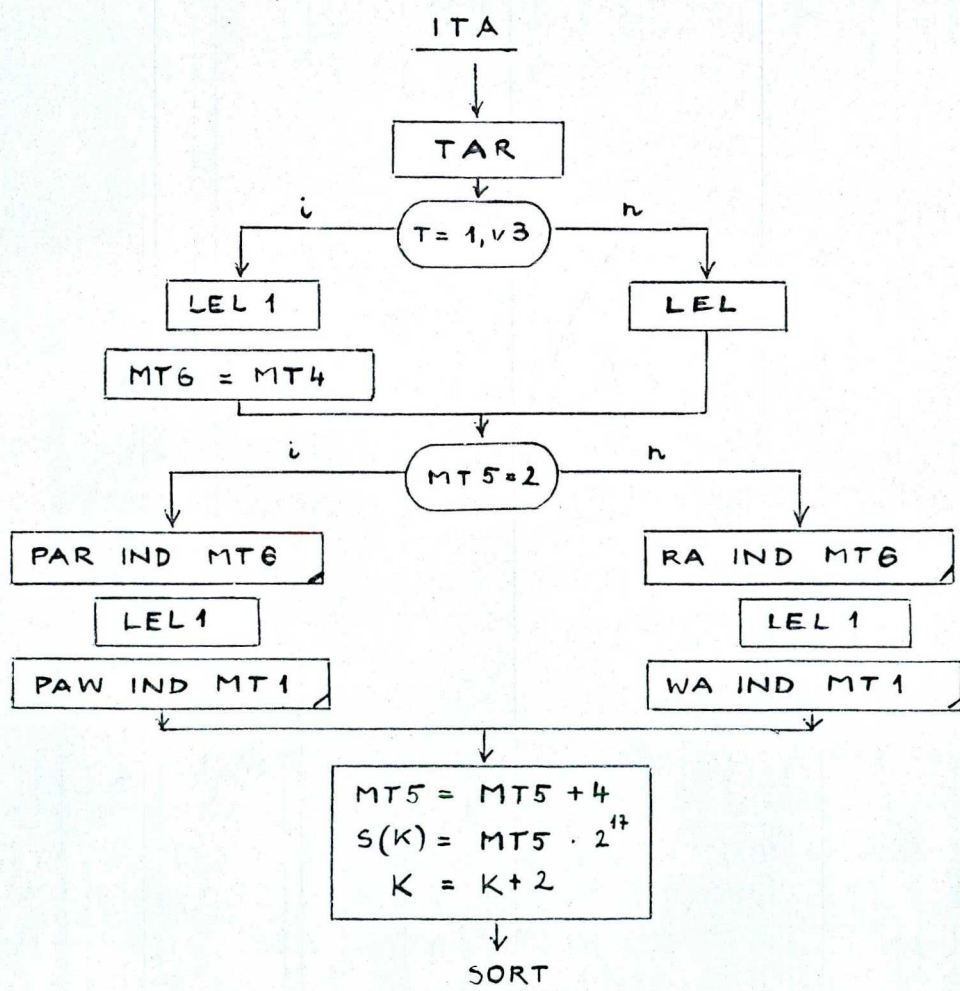
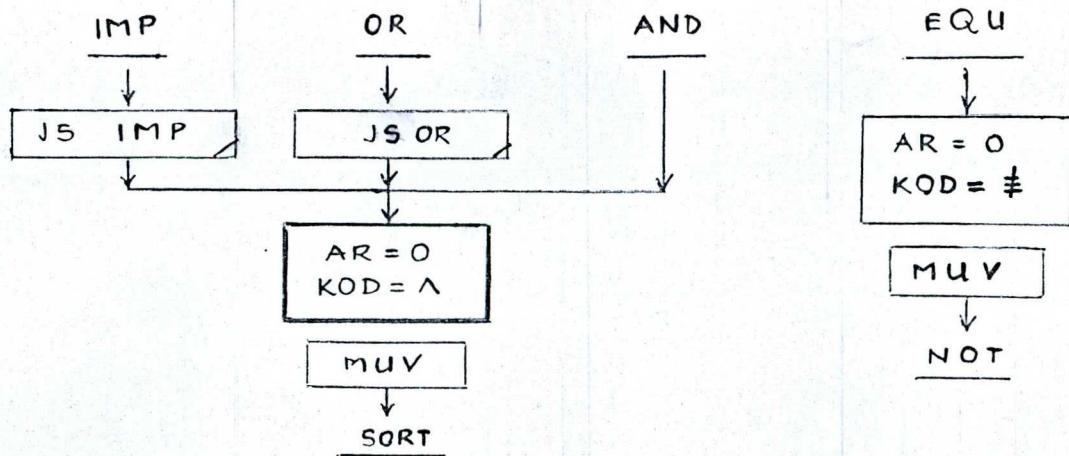


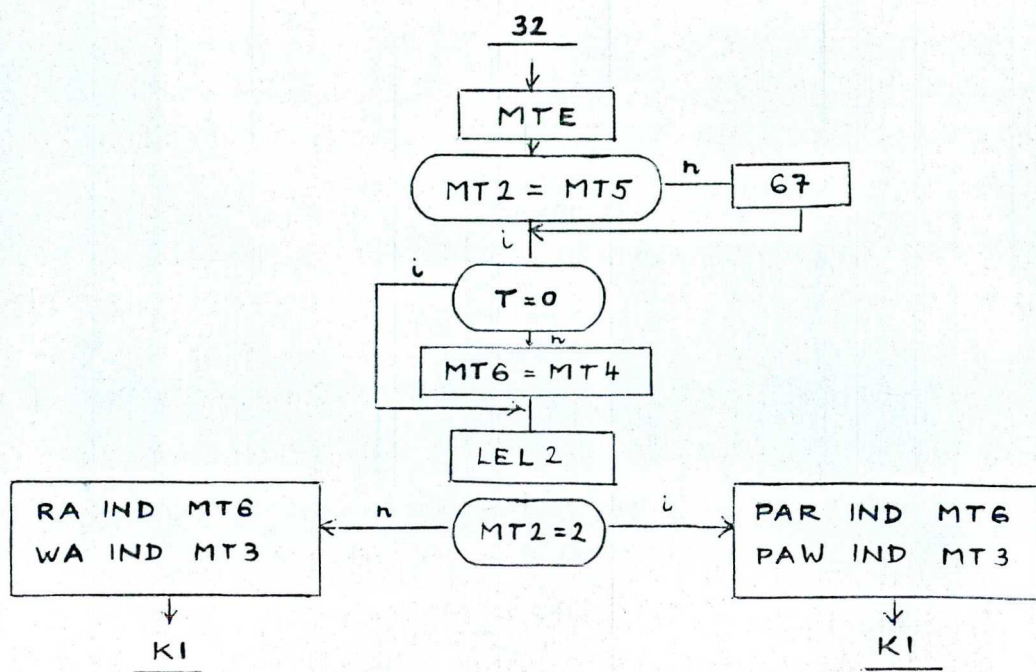
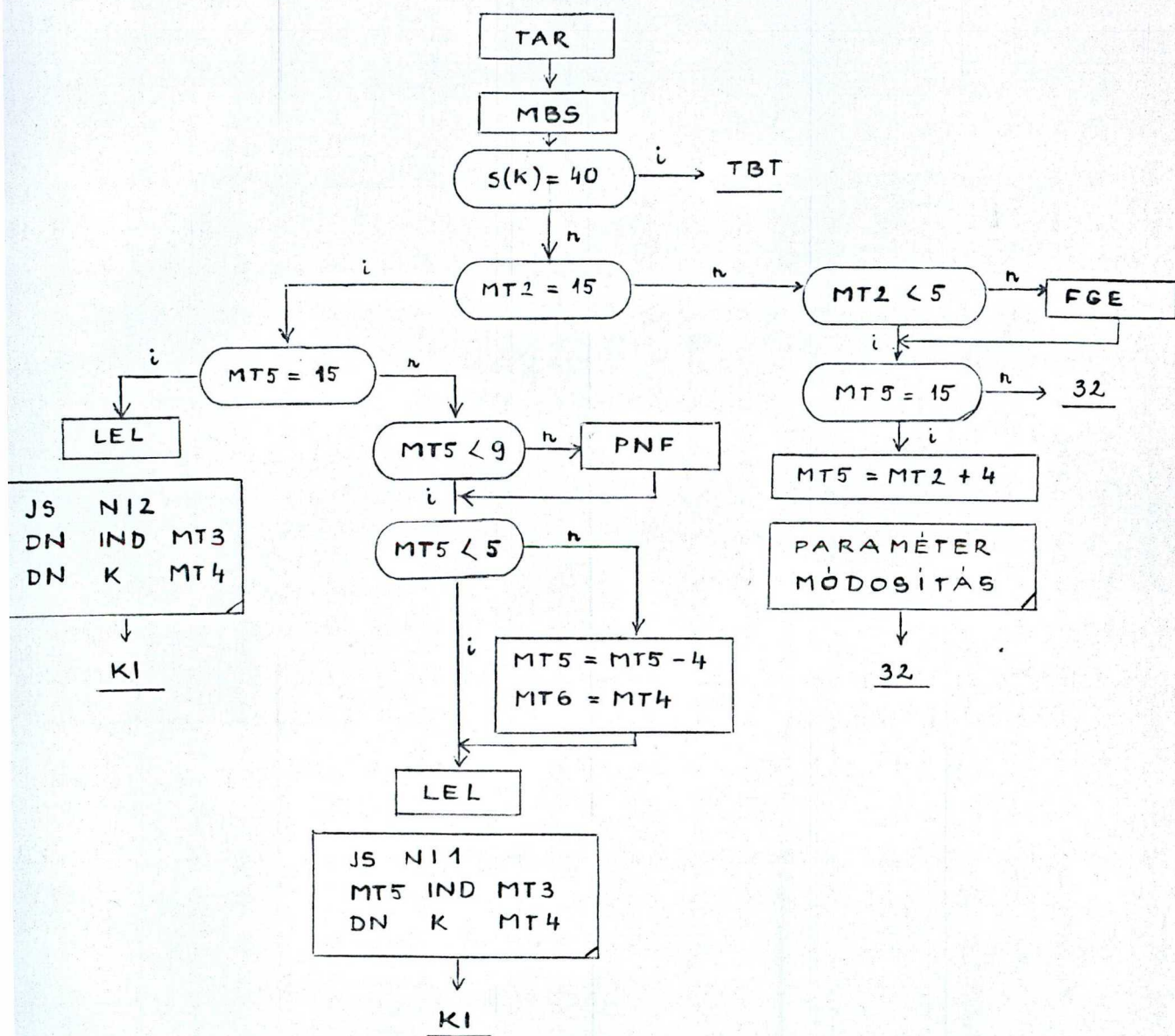
LT

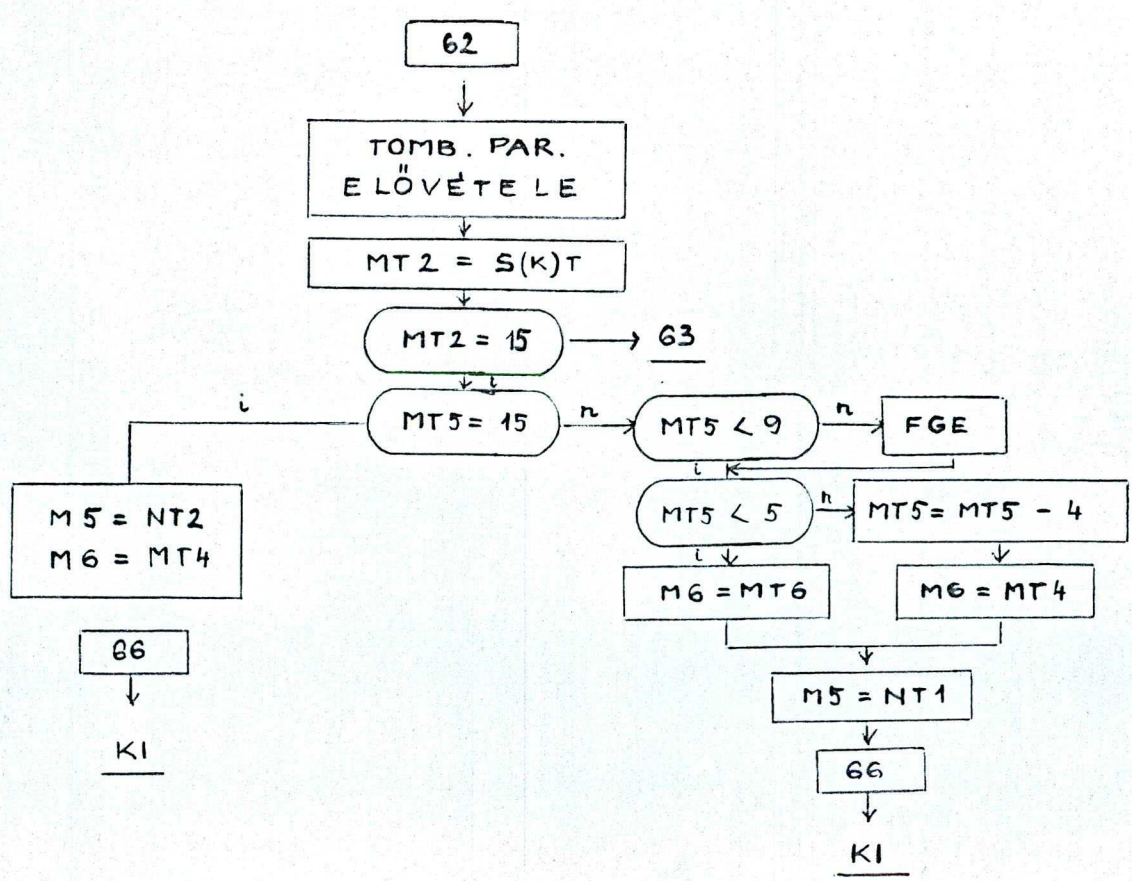
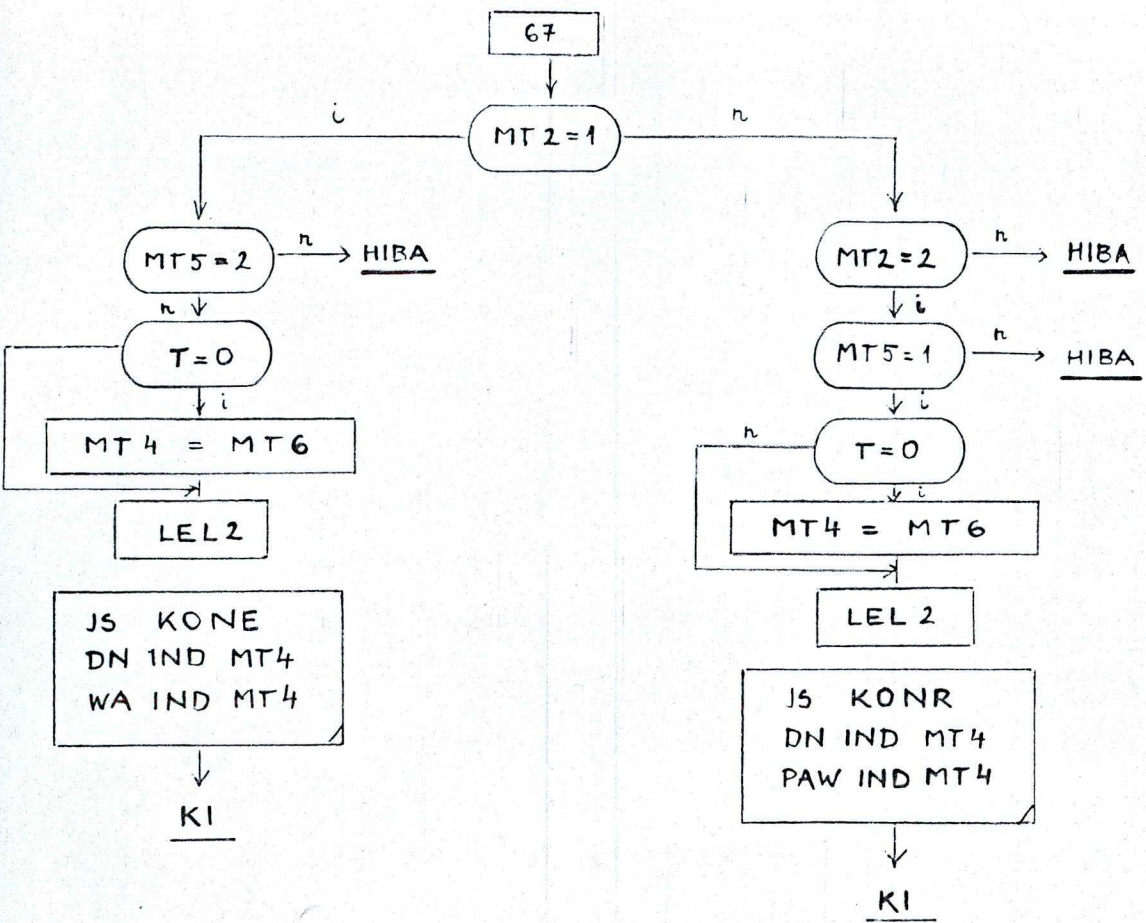


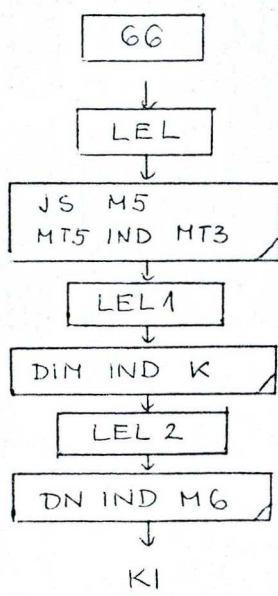
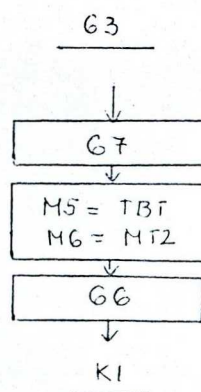
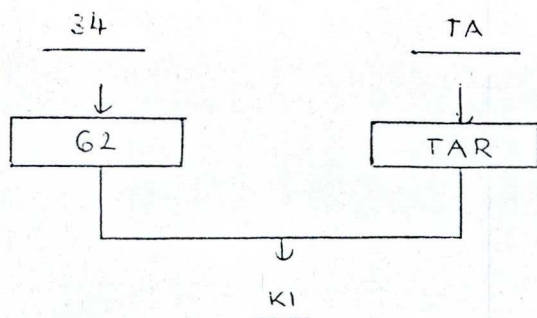
NOT



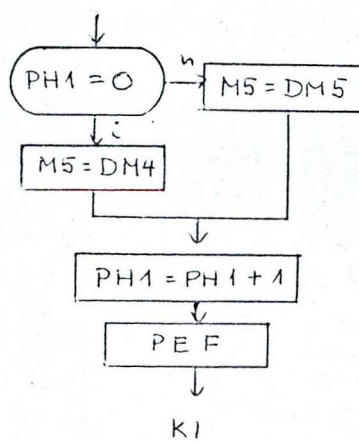




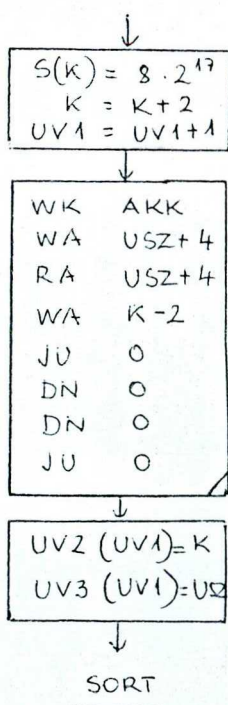




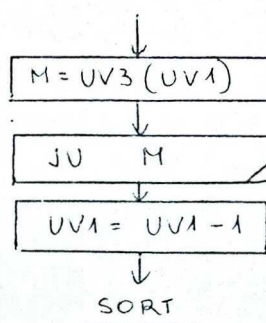
PERK



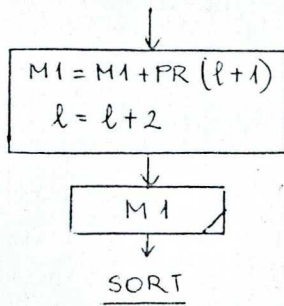
UV



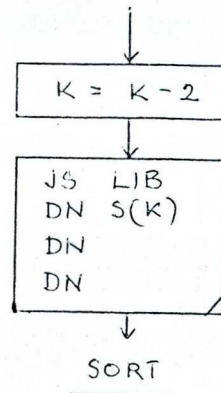
UV



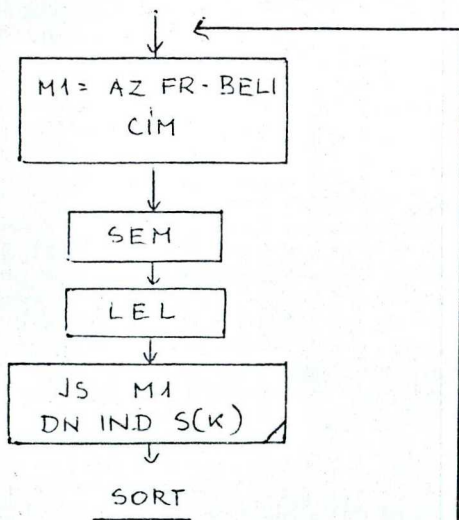
R 4 1



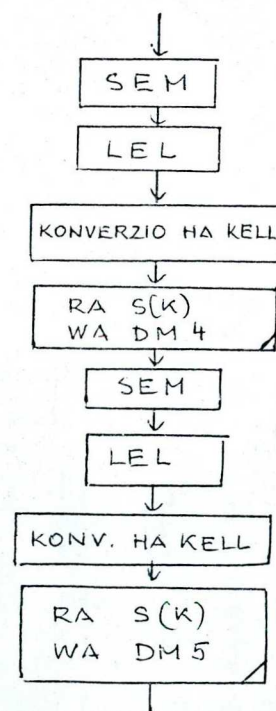
LW



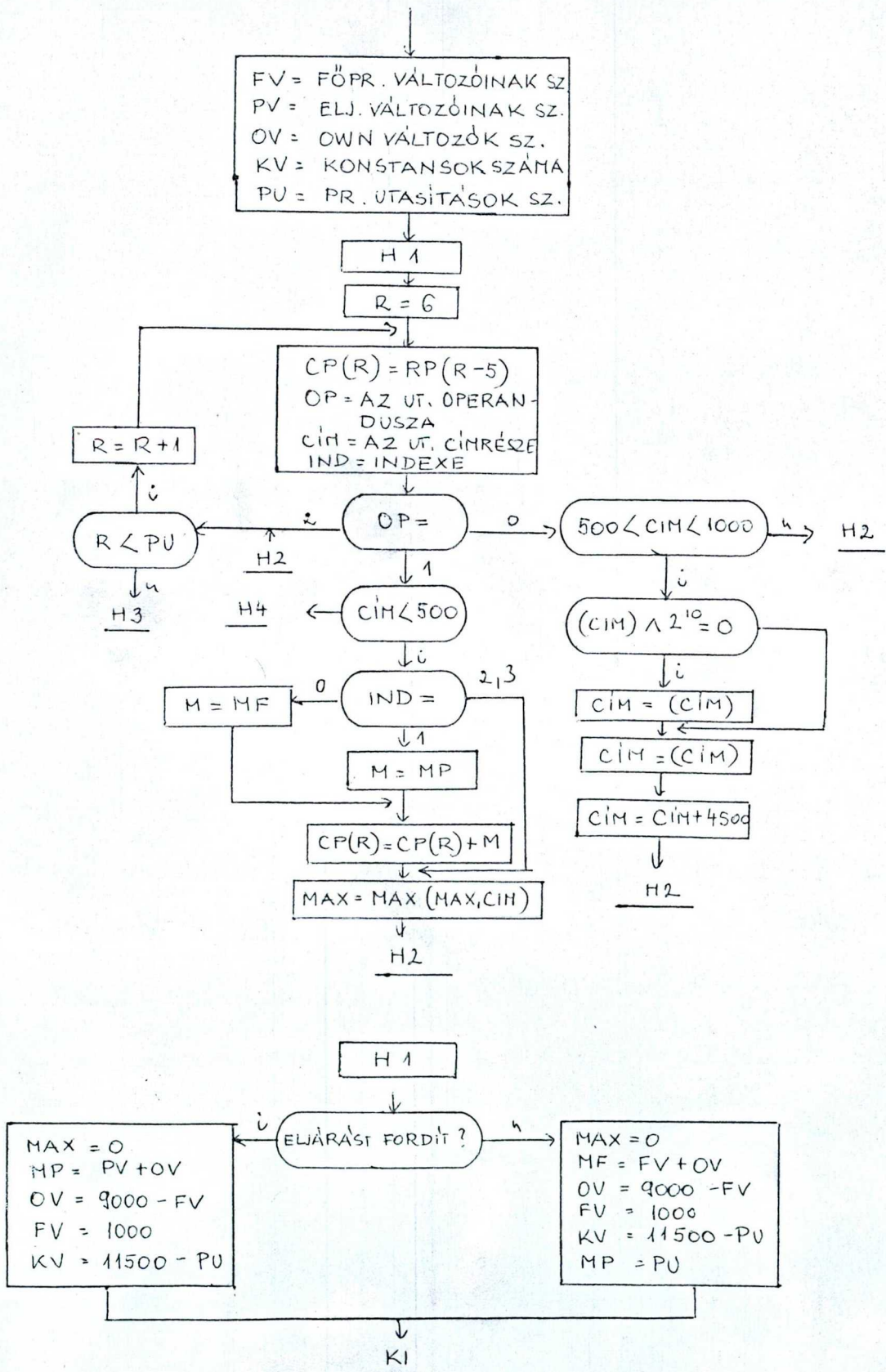
TU, DA INS, OUT

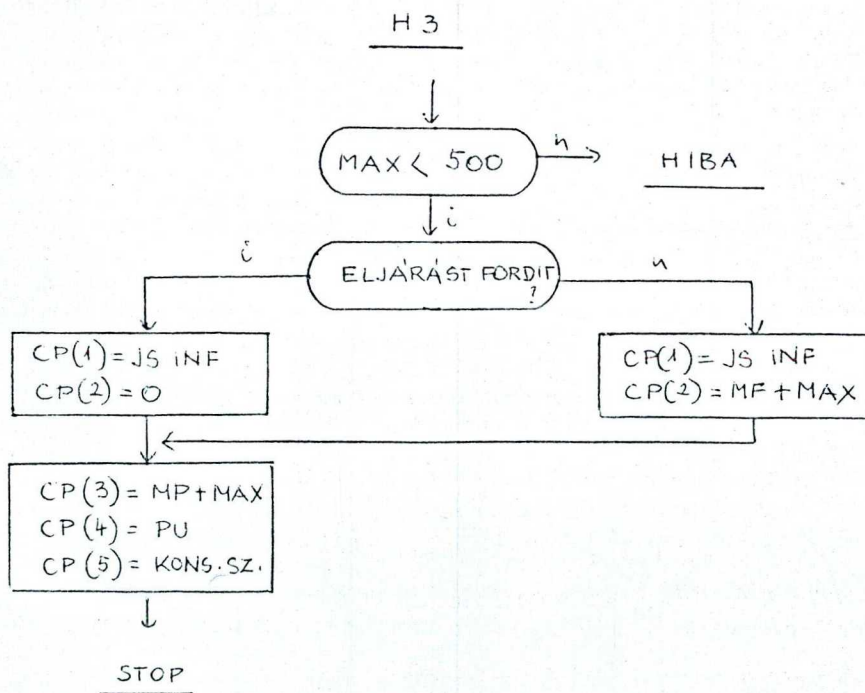
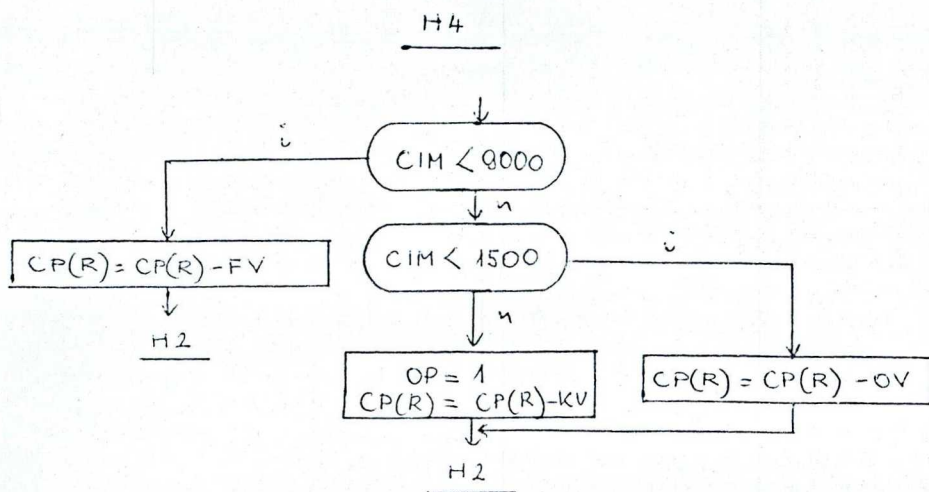


MT, MD



HARMADIK MENET





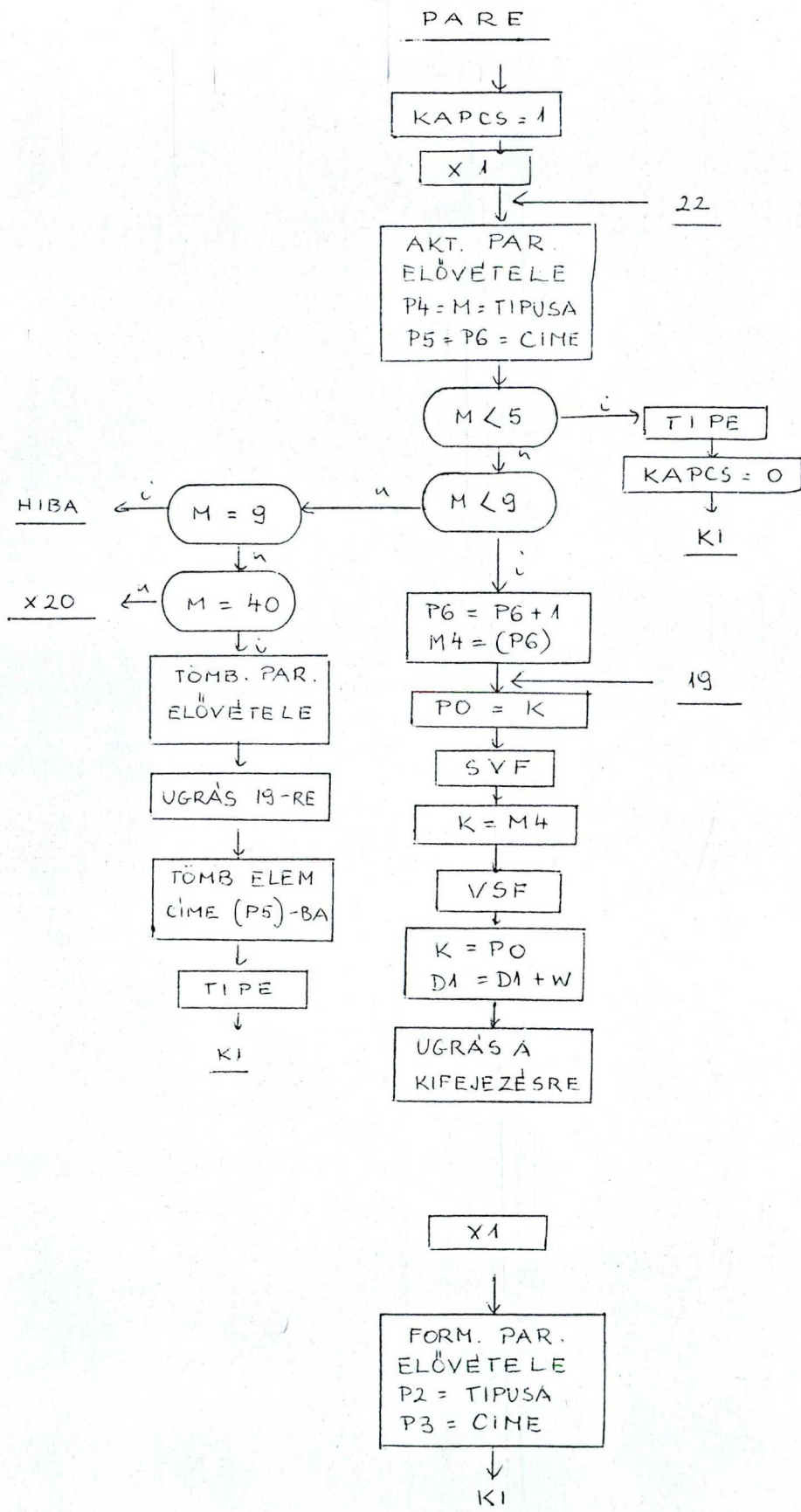
Futtatórendszer

Rutinok:

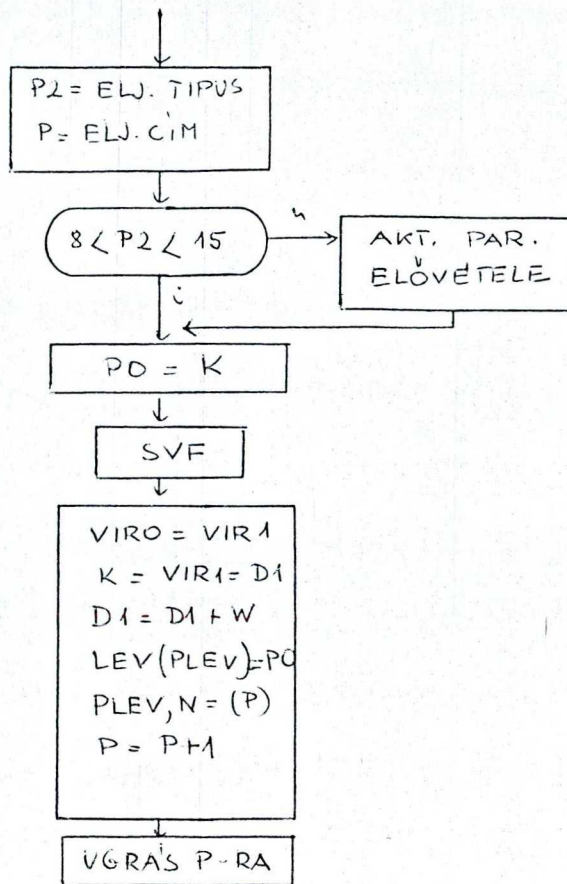
PARE	formális paraméternek értékadás
ELJ	eljárás aktivizálás
RETURN	eljárás vége
TE	tömbelem értékét tárolja.
BEGIN	blokkezdet tevékenysége.
END	blokkvége "-"
TOM	tömbdekларáció
OTO	Own tömbdekларáció.
TBT	tömbelemnek értékadás
NT1	specifikált tömbelemnek értékadás
NT2	specifikálatlan " "
RTE	tömbelembe olvasás
RT	tömb beolvasás
RTS	specifikált tömb beolvasása
RS	specifikált tömbelembe olvasás
KAP	switch-elemre ugrás
CIMKE	ugró utasítás kiértékelése
NI1, NI2	specifikálatlan paraméternek értékadás
OR, IMP	logikai műveletek végrehajtása
HATV	hatványozás
KE	konverzió egészre
VNA	value paraméternek értékadás
VNAT	value tömbelemeknek "
VSZK	valós szám kiírás
ESZK	egész szám kiírás
IDÉZ	szövegkiírás
NIDE'Z	string kiírás
LiB	könyvtári eljárás elővétele
INF	kezdőértékek beállítása
TU	mágnesszalag egység kijelölés
DA	lemez cím kijelölés
MT	mágnesszalag művelet
MD	mágneslemez művelet
LiNE	soremelések

Fontosabb változók:

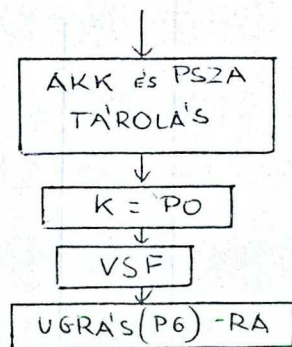
k	indexregiszter
AKK	akkumulátor
D ₁	din.memória első szabad címe.
D ₂	előző blokk D ₁ értéke
N	blokk szintszám
PLEV	eljárás szintszám
VIRO	előző eljárás adatainak kezdő címe
VIR ₁	az eljárás adatainak leendő "-"
LEV	indexregiszterek értékét tartalmazó vektor
PX	munka vektor
W	eljárások változóinak száma.
L L	érvényes könyvtári eljárások száma
LB	" " " nevét és kezdőcímét tartalmazó vektor.



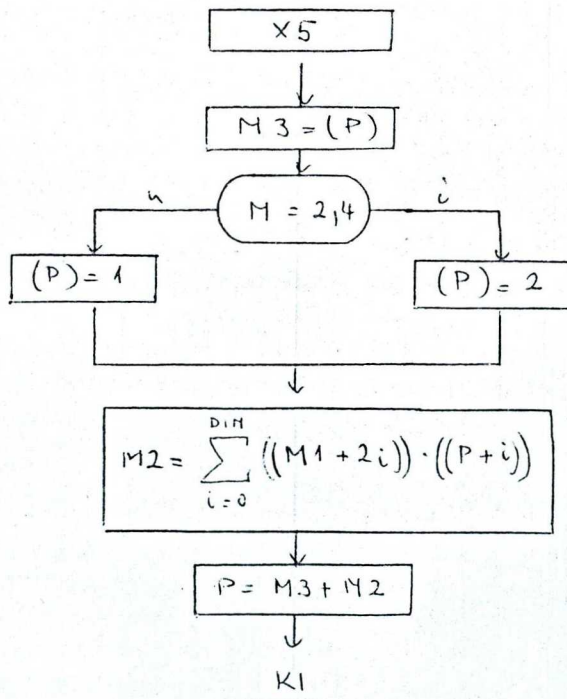
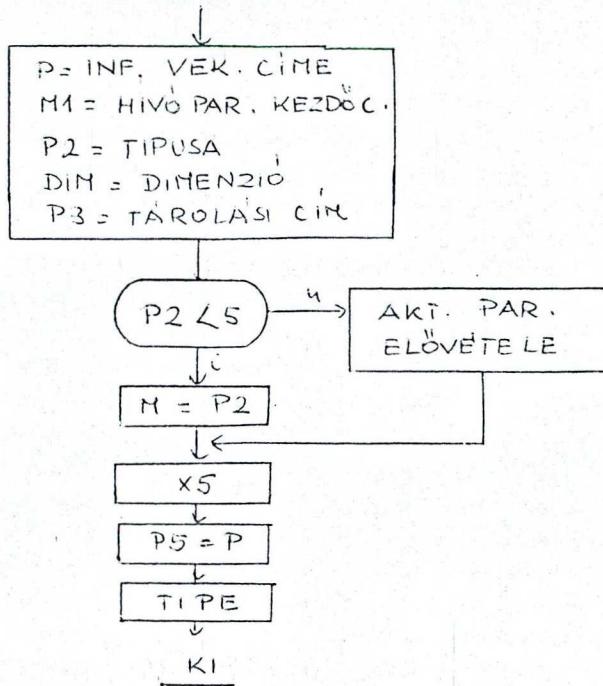
ELJ

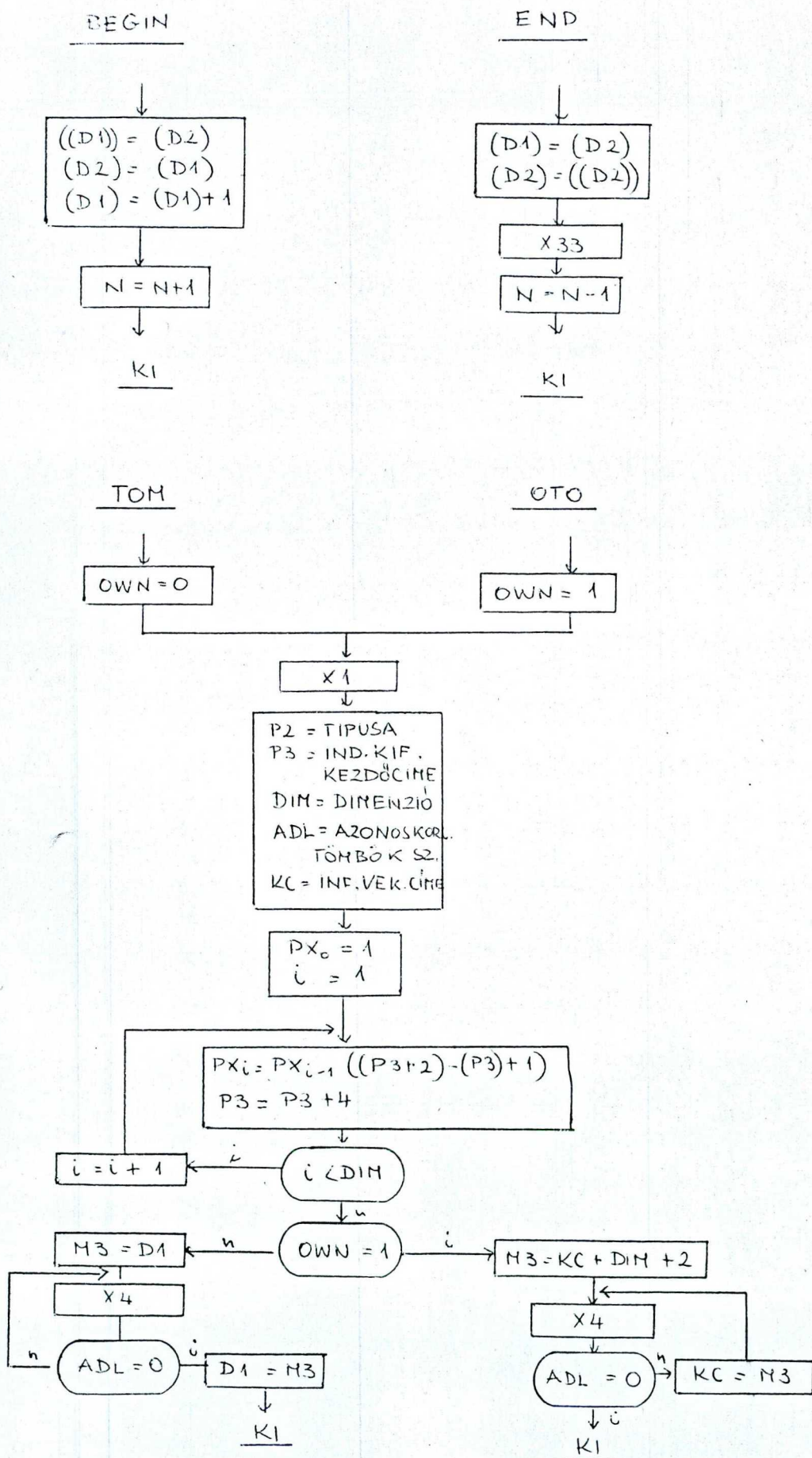


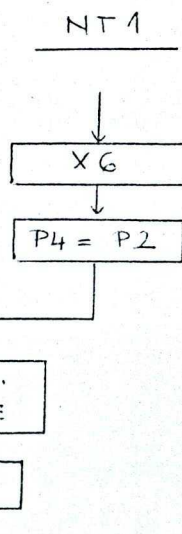
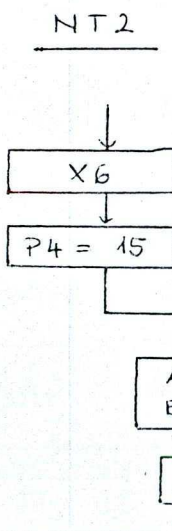
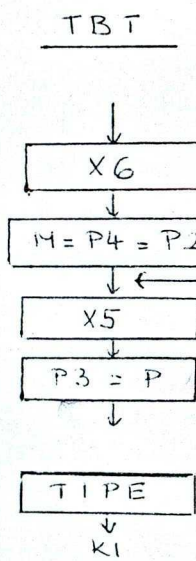
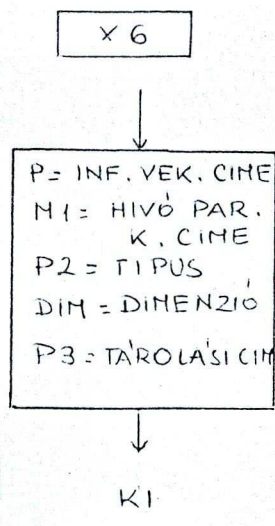
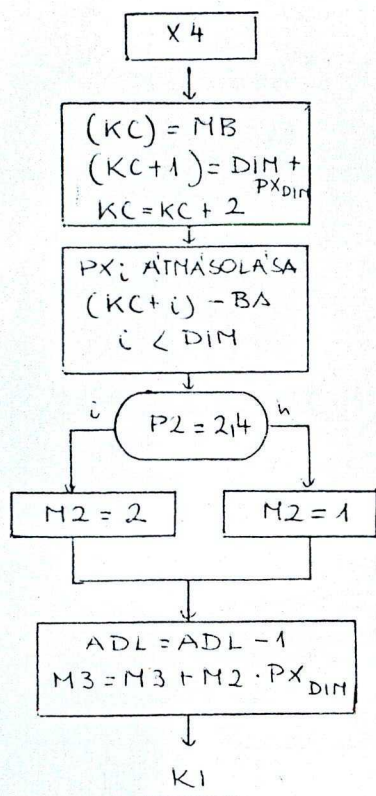
RETURN



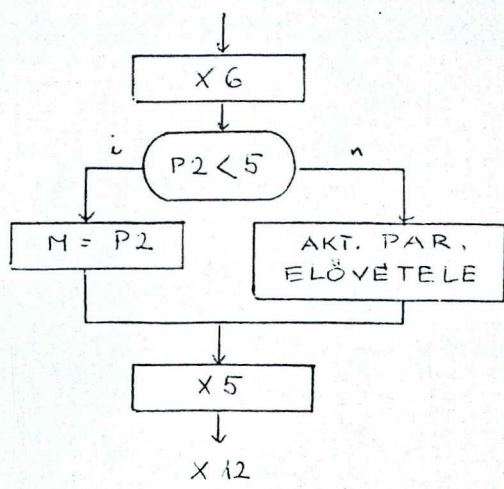
TE



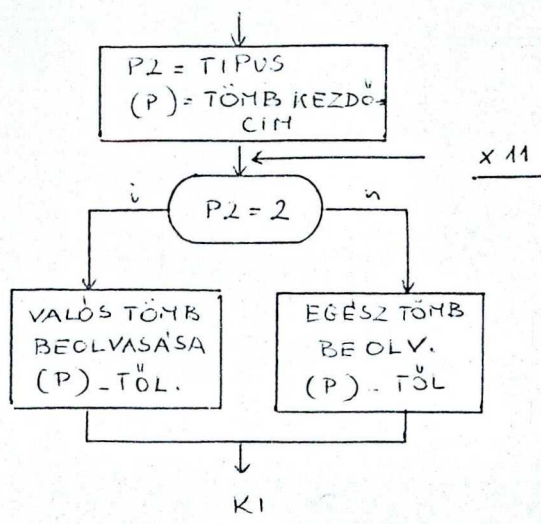




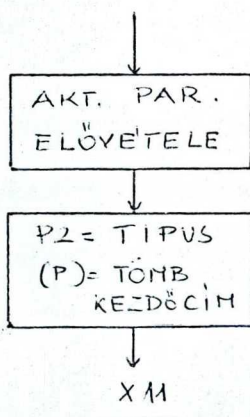
RTE



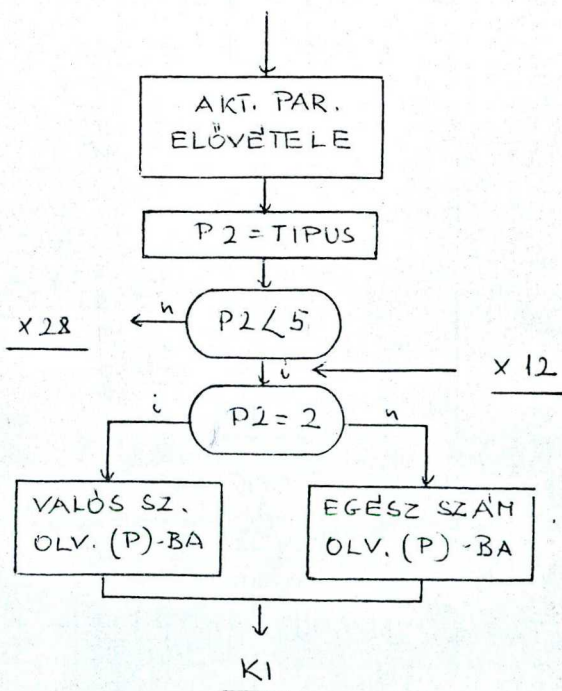
RT

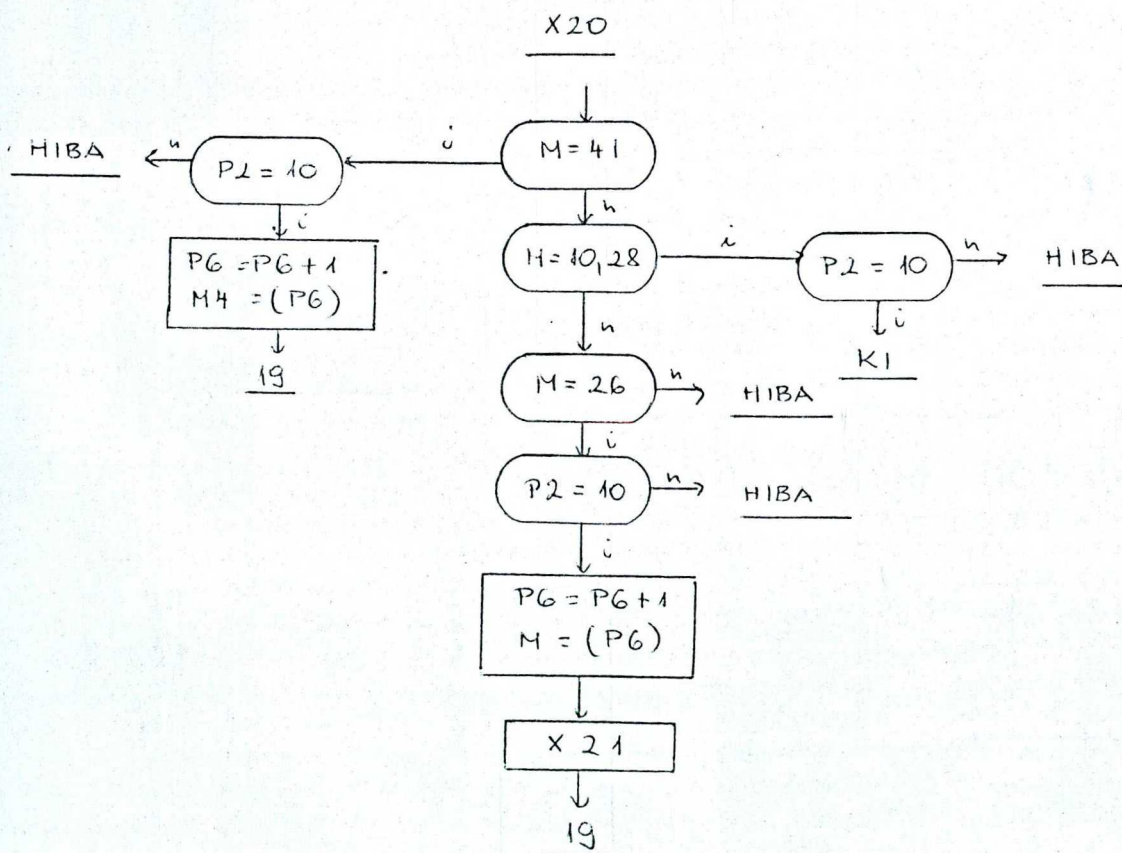
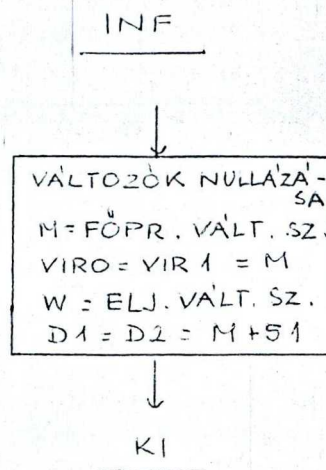
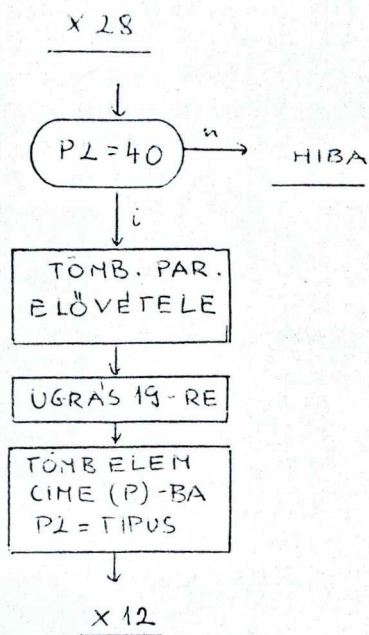


RT5

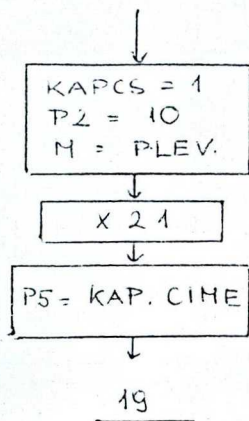


RS

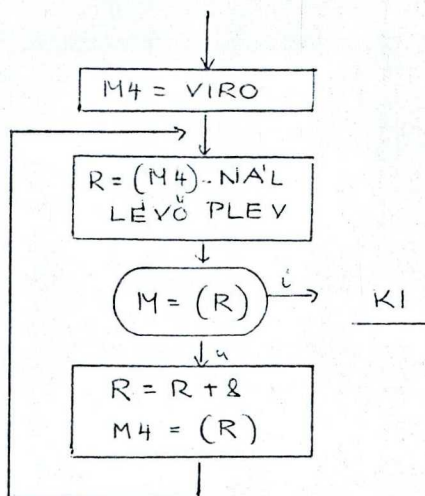




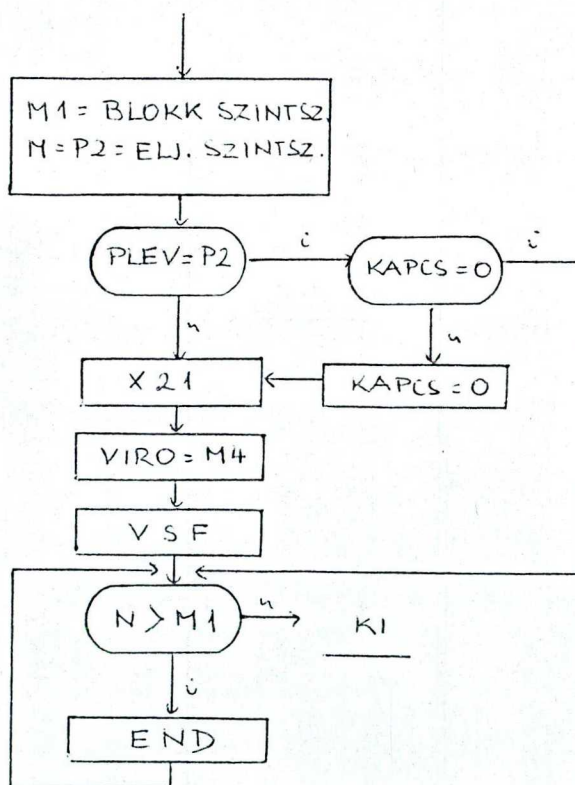
KAP

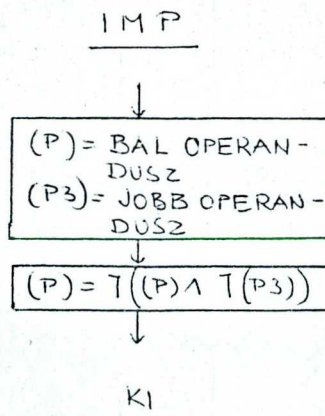
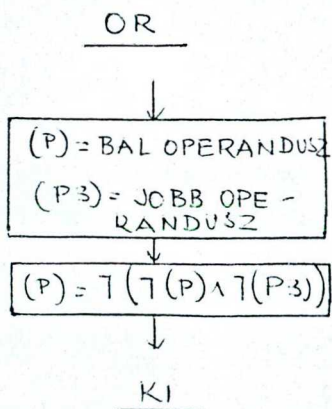
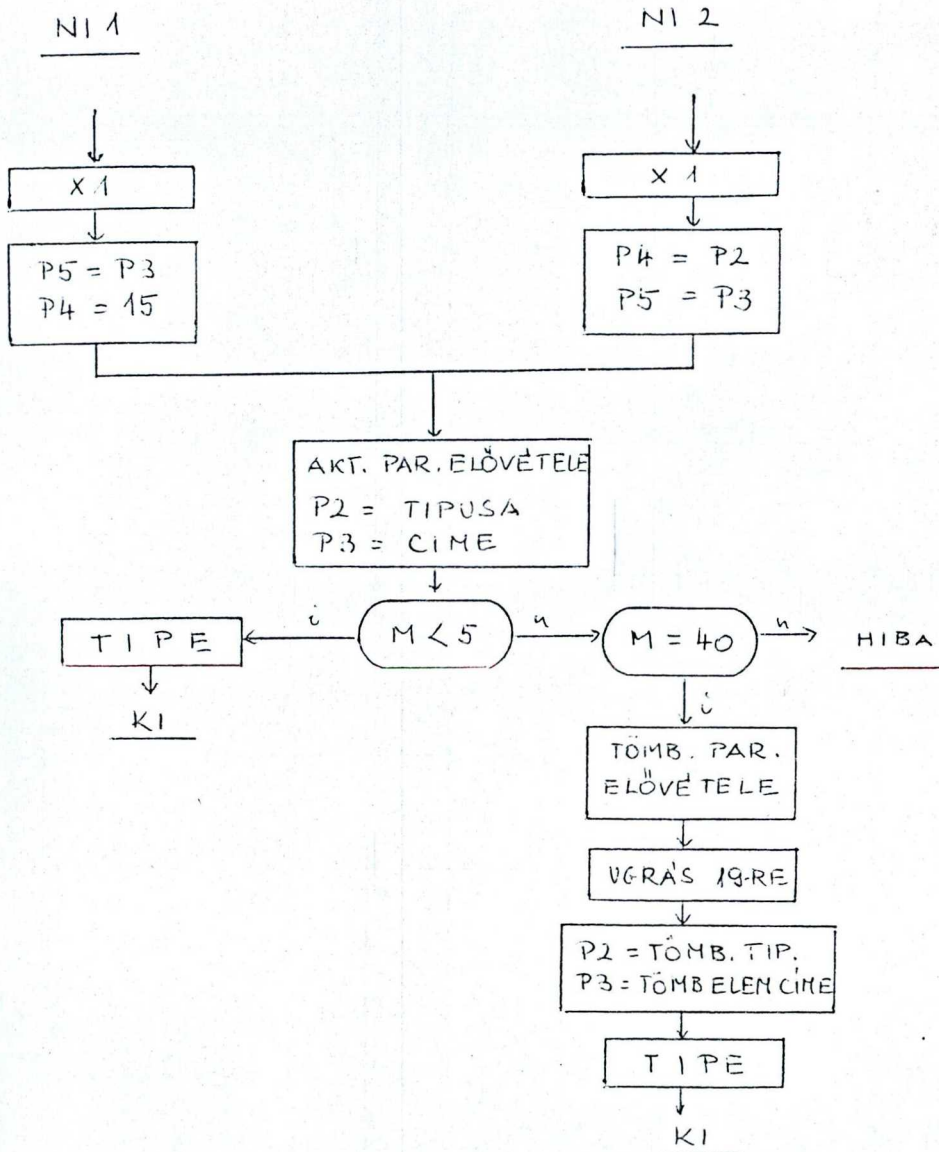


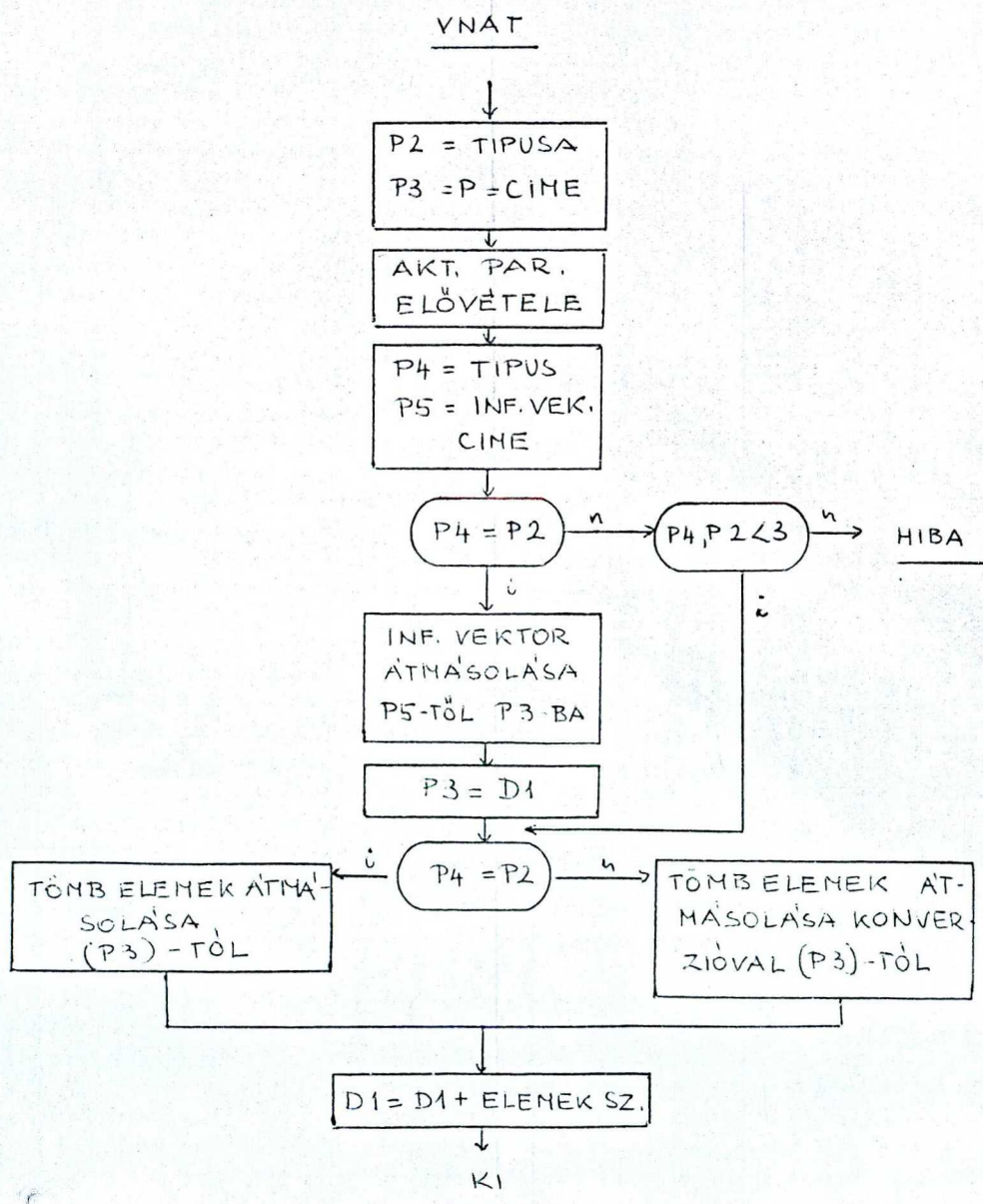
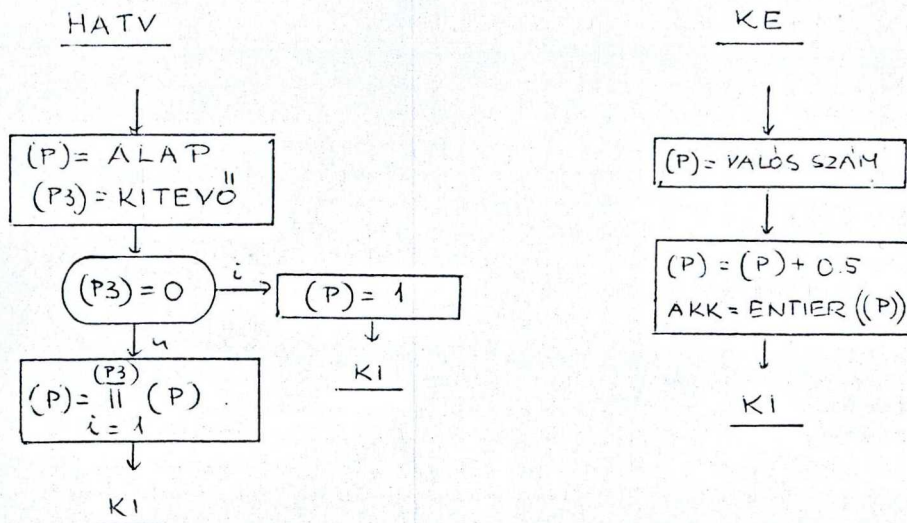
X 2 1



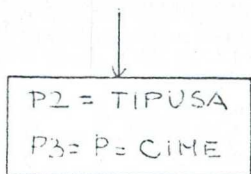
CIMKE







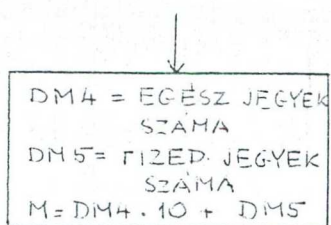
VNA



↓

22

VSZK



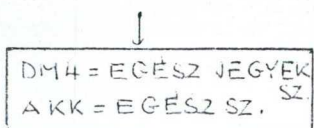
↓

KIÍRÁS (M)

↓

K1

ESZK



↓

KIÍRÁS (DM4)

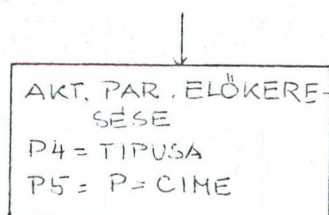
↓

K1

IDÉZ



NIDÉZ

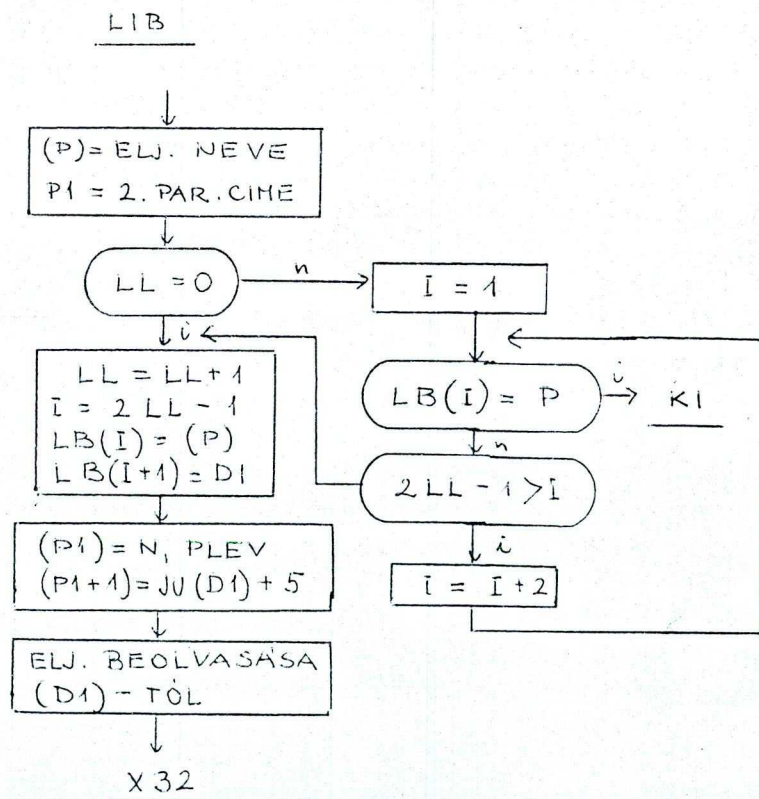
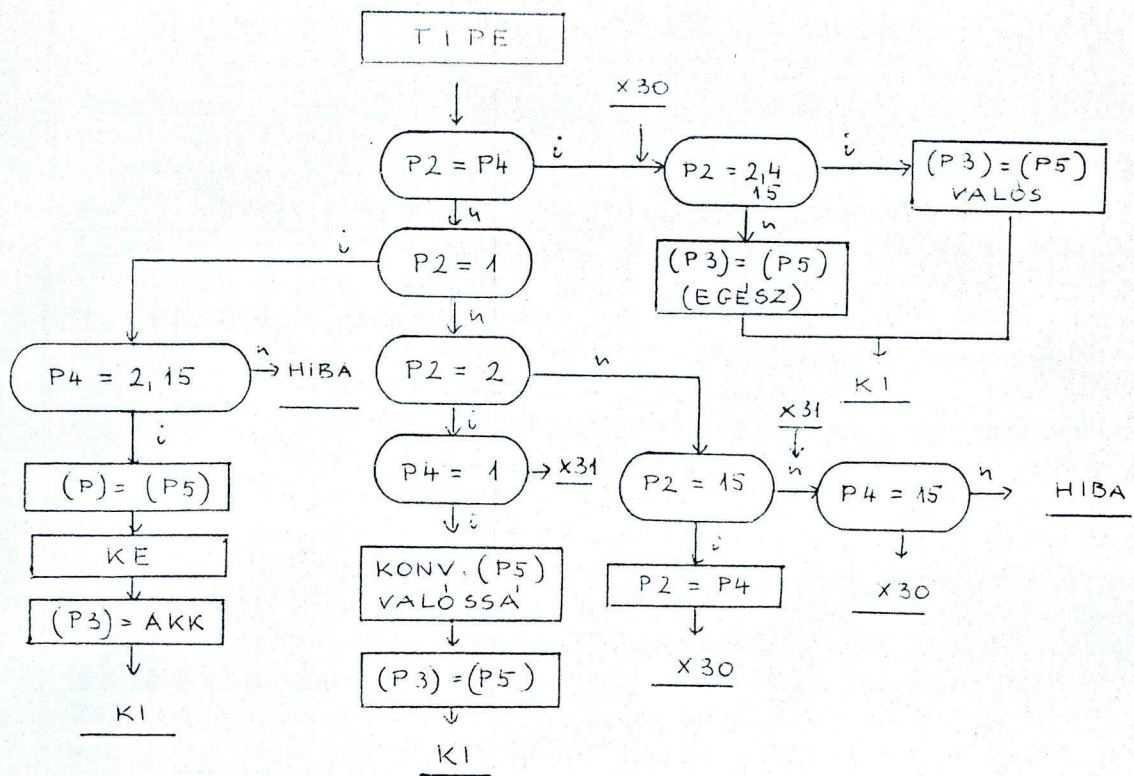


↓

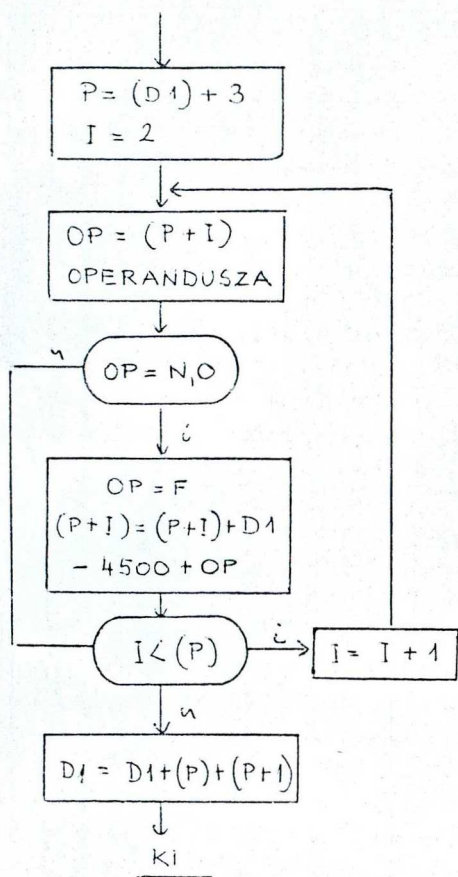
KIÍRÁS:
(P) DB SZÓT
(PH) - TÖL

↓

K1



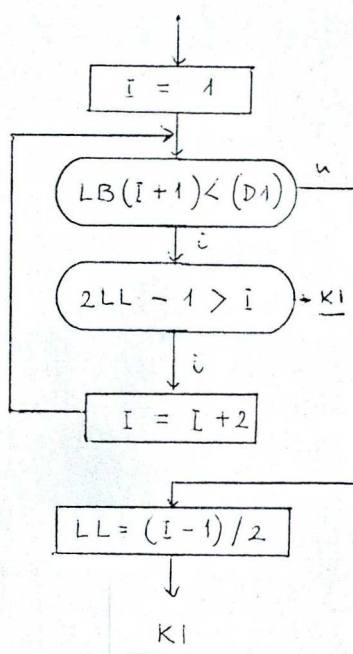
X 32



SVF

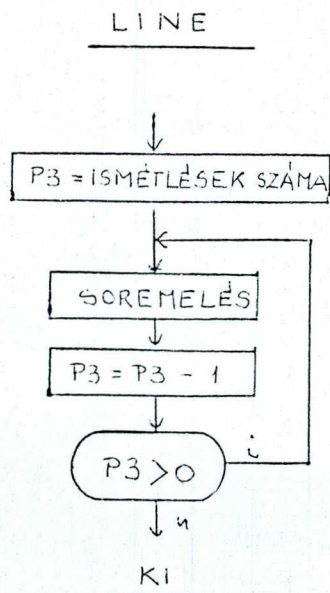
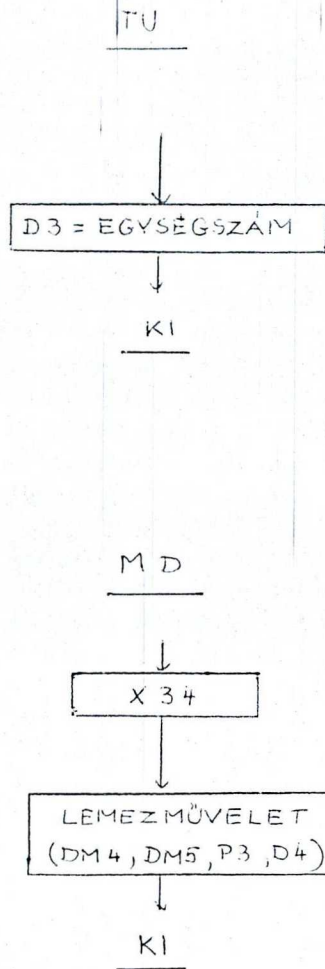
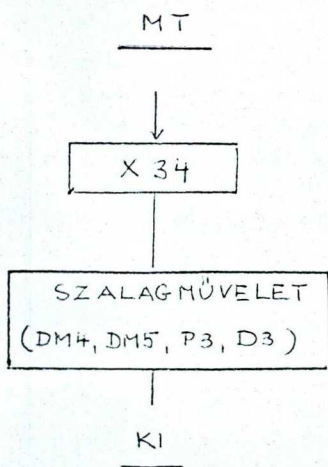
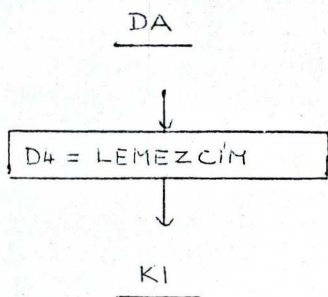
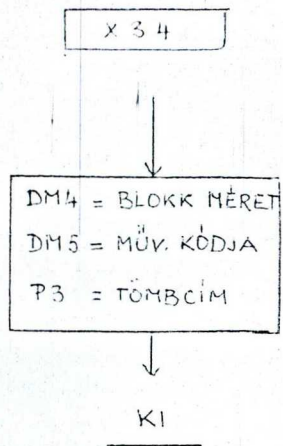


X 33



V SF





4. Melléklet

Mintapéldák

Az első mintapélda rekurzív eljárásbehívással kiszámítja 3 és 9 faktoriális értékét. A program a következő:

```
BEGIN      INTEGER A;  INTEGER  PROCEDURE  FAC/N/;
VALUE      N;  INTEGER  N;
FAC=IF      N EQ      0 THEN 1 ELSE  FAC/N-1/.N;
WRITE      /* FAKTORIÁLIS SZÁMITÁS */;
LINE /1/;   A=FAC /3/; WRITE /*3. !=', A/;
LINE /1/;   A=9;  WRITE  /A, ' != ', FAC /A/ /;
LINE /1/;   END ;
```

A program outputja :

FAKTORIÁLIS SZÁMITÁS

3 != 6

9 != 362880

/Mellékelve a gépkódu célprogram/

A második mintapélda egy sarkára állított kocka térbeli képét rajzolja ki különböző szemszögekből.

/Mellékelve a forrásprogram, valamint a program outputja./