tanulmányok 68/1977

MTA Számitástechnikai és Automatizálási Kutató Intézet Budapest

MAGYAR TUDOMÁNYOS AKADÉMIA
SZÁMITÁSTECHNIKAI ÉS AUTOMATIZÁLÁSI KUTATÓ INTÉZETE

A PL360 PROGRAMOZÁSI NYELV

Irta:

GYÁRFÁS ANDRÁS

Tanulmányok 68/1977.

A kiadásért felelős:

DR VÁMOS TIBOR

ISBN 963 311 051 3

ISSN 0324-2951

779280 MTA KESZ Sokszorosító. F. v.: Szabó Gyula

TARTALOMJEGYZÉK

Old.

1. BEVEZETÉS 1
2. A PL360 NYELV LEIRÁSA 7
   2.1 Néhány szó a nyelv leirása elé 7
   2.2 Alapfogalmak 7
   2.3 Deklarációk 10
   2.4 Utasitások 17
   2.5 A szinonim dekalráció 31
   2.6 Kezdeti értékek deklarációban 34

3. PROGRAM ÉS ADATSZEGMENTÁLÁS 38
   3.1 A szegmens fogalma 38
   3.2 Program szegmentálás 39
   3.3 Adatszegmentálás 41
   3.4 Külső rutin hivása PL360-ból 43
   3.5 Supervisor hivások 44
   3.6 PL360 rutin hivása külső rutinból 44

4. TUDNIVALÓK A FORDITÓPROGTAMRÓL 45
   4.1 A PL360 szimbólumának reprezentációja 45
   4.2 Standard azonositók 45
   4.3 Standard eljárások 46
   4.4 A forditóprogram inputja és listája 46
   4.5 A listázás vezérlése 47
   4.6 A COPY lehetőség 48
   4.7 Limitációk 48
   4.8 A forditóprogramok hivása és paraméterek átadása 48

5. FÜGGELÉKEK 49
   5.1 Makrólehetőség 49
   5.2 PL360 összefoglalás 53
   5.3 PL360 kódgenerálásról 56
   5.4 A PL360 forditó hibaüzenetei 60
   5.5 Két példa PL360 programra 62
   5.6 A PL360 szintaxisa 67

R E F E R E N C I Á K 70

1. BEVEZETÉS

A PL 360 programozási nyelv N. Wirth alkotása. / [1] , [2],[3] / A nyelv forditóprogramját N. Wirth, J.W. Wells Fr. és E Satterthwaite Fr. készitette el /[4] /, a forditóprogramon bizonyos bővitéseket az Oslói Számitóközpont végzett. /[5]/.

A PL360 a következő hármas céllal született:

/1/ Az IBM 360 /továbbiakban 360/ hardware által nyujtott lehetőségek kihasználása.

/2/ Kényelmes programirás és javitás.

/3/ "Tiszta" programozási stilus lehetővé tétele és támogatása.

Nyilvánvalóan az /1/ követelményt az assembler elégiti ki leginkább, mig a /2/ és /3/ feltételek magasszintü programozási nyelvek irányába mutatnak. A PL360 példázza azt, hogy ezeket az ellentmondó feltételeket ki lehet elégiteni. A konkluzió természetesen nem az, hogy a PL360 pótolja az assemblert, vagy a magasszintü nyelveket. A PL360 nyelvet a következő feltételek esetén érdemes alkalmazni:

1. Igen hatékony tárgykódra van szükség.
2. A feladat elég nagy ahhoz, hogy hagyományos /assembler/ módszerrel ne legyen kényelmes a programozás, javitás, dokumentáció.

Ilyen tipusu feladatok gyakran adódnak a software-ben, pl. forditóprogram irás, operációs rendszerek irása kapcsán. Azt mondhatjuk tehát, hogy a PL360 rendszerprogramozási nyelv.

Érdemes megjegyezni, hogy a PL360 forditóprogram az assemblernél jóval gyorsabb, és a tárgykód szinte ugyanolyan hatékony. Megemlitjük, hogy az IBM/SIMULA forditóprogram PL360-ban iródott és az IBM/SIMULA forditási sebessége igen jó.[6]

Magyarországon sok IBM és ESZR számitógép müködik, ezért ugy gondoljuk, hogy a PL360 terjesztése fontos feladat - ez a kiadvány is ezt a célt szolgálja. [1]

[1] Ez a nyelvi ismertetés [3], [4] és [5] információi alapján készült. A PL360 forditóprogramot az Oslói Számitóközpont bocsájtotta rendelkezésünkre - ez az OS operációs rendszer alá épült, a DOS operációs rendszerhez való illesztést Hermann Tamás végezte az MTA SZTAKI, Számitógép Főosztály Software Osztályán.

2. A PL360 NYELV LEIRÁSA

2.1. Néhány szó a nyelv leirása elé:

a 2. fejezetben a PL360 nyelv informális, de azért preciz leirását kisérelj̈ük meg. A leirás lineárisan olvasható, előrehivatkozás nincsen benne, legfeljebb célzások bizonyos később definiálandó fogalmakra. Néhány nehezebb, vagy tul aprólékos részt külön tárgyalunk. Ezek: a szinonim deklarációk /2.5/, kezdeti értékek beállitása /2.6/, valamint a program és adatszegmentáció, melyet külön fejezet /3.fejezet/ tárgyal. Ez utóbbi igen lényeges a PL360-ban, de jól különválasztható a nyelv többi elemétől.

A Függelék egy része /6.rész/ a PL360 szintaxisának BNF leirását adja, ez azonban referencia jellegü. A Függelék 5. része két teljes példaprogramot mutat be, ezt érdemes átfutni többször is. A nyelv gyakorlati használatát könnyiti meg a Függelék 2. része, mely gyors referencia a PL360 nyelvhez.

A Függelék 1. része a PL360 makro lehetőségeiről ad leirást, ezt csak a PL360 teljes ismeretében érdemes használni. Ez a rész valószinüleg csak annak érthető, aki ismeri a makró fogalmát. Megjegyezzük, hogy a leirás jó megértéséhez a következő ismeretek szükségesek:

A 360 utasitáskészlete [7], [8]
Algol 60-ról ismeretek [9]

2.2 Alapfogalmak

A PL360 nyelv nem független - a 360-as /vagy ezzel funkcionálisan ekvivalens/ gépcsaládhoz kötődik. A PL360 nyelv szemlélete szerint a 360 regiszterekből, cellákból és függvényekből áll.

2.2.1. Regiszterek és cellák

Alapvető tulajdonságuk, hogy képesek valamely é r t é k tárolására. A regisztereket és cellákat abból a szempontból osztályozzuk, hogy ez az érték nagyság és tipus szerint mi lehet. Ilymódon a regiszterek három tipusát különböztetjük meg:

- e g é s z r e g i s z t e r /32 bit hosszuság/
- v a l ó s r e g i s z t e r /32 bit hosszuság/
- d u p l a s z a v a s v a l ó s r e g i s z t e r /64 bit hosszuság/

A cellákat öt tipusba soroljuk:

- b y t e c e l l a /8 bit hosszuság/
- f é l s z a v a s e g é s z c e l l a /16 bit hosszuság/
- e g é s z c e l l a /32 bit hosszuság/
- v a l ó s c e l l a /32 bit hosszuság/
- d u p l a p o n t o s s á g u v a l ó s c e l l a /64 bit hosszuság/

2.2.2. Függvények

A PL360 a 360 utasitáskészletét függvényeknek tekinti, melyek regiszterek és cellák értékeit változtatják meg bizonyos szabályok szerint. /Esetleg a feltételkódot is beállitják/. Ezen függvényeket nem definiáljuk, [7], [8] -ban pl. megtalálhatók. Itt is kiemeljük, hogy hatékony PL360 programozáshoz a 360 utasitáskészletének ismerete szükséges. Előrebocsájtjuk, hogy ezen gépi utasitások függvényhivásként jelennek meg a PL360-ban.

2.2.3. A PL360 szimbólumai

Egy PL360 program szimbólumok sorozata. Ezeket a szimbólumokat a következőképpen osztályozzuk:

a l a p j e l, mely lehet betü, számjegy, speciális karakter és kulcs-szó

s t r i n g, melyen idézőjelek /"/ közé zárt karaktersorozatot értünk./Az "-karaktert szókásos módon "" jelöli egy string belsejében/.

c o m m e n t, ez a comment kulcsszótól a következő ; karakterig tart. Semmiféle hatása nincs a programra, pusztán a dokumentációt segiti.

2.2.4. Azonositók

Szokásos módon a z o n o s i t ó n betüvel kezdődő, betüket és számjegyeket tartalmazó karaktersorozatot értünk. Előrebocsájtjuk, hogy az azonositókat deklarálni kell - kivételt képeznek a standard szonositók.

2.2.5. Konstansok

A konstansokat a következőképpen csoportositjuk: egész, valós, hexadecimális és byte-konstansok. A konstansok definiciójához használjuk a következő jelölést:
DEC: decimális jegyek sorozata /0,1...,9 lehet és legalább egy számjegy legyen /
E D E C: D E C vagy_D E C ahol az aláhuzás jel negativ előjelet fog jelenteni

HEX: # jel után dexadecimális jegyek sorozata /0,1, ...,9,A,B,C,D,E,F / lehet és /legalább egy jegy legyen a sorozatban/.

A. E g é s z-konstansok: EDEC, HEX alakuak
B. F é l s z a v a s-e g é s z konstansok: EDEC S, HEX S, alakuak /S jelzi, hogy félszavas egész/
C. V a l ó s konstansok: EDEC . DEC ,EDEC R ,EDEC. DEC 'EDEC ,EDEC'EDEC ,HEXR

Itt a . tizedespontot jelöl, az ' után 10-es alapu skálafaktort kell irni.

D. Duplaszavas valós konstansok:
EDEC . DEC L , EDEC L , EDEC . DEC' EDEC L ,
EDEC' EDEC L , HEXL

E. Hexadecimális konstans: HEX

F. Byte-konstans: "karakter" /egykarakteres string, a stringet 2.2.3-ban definiáltuk/

HEX X /két hexadecimális jegy lehet/

Példák: A-ra: 0, 1063, -1 , 1F2E3D4C
B-re: 10 S , #FF00 S , = 13S
C-re: 1.0, - 13R, -3.1416, 2.7'8, 3.52' 3, 46000001R
D-re: 3.14159265359L #4E0000000000001L
E-re: # AB13
F-re: "B" , "?", 1FX, """" /Az utolsó a " karaktert jelöli/

Megjegyzendő: Negativ előjel: - pozitiv előjel : nincs
skála-faktor elé: ' félszavas egész végére: S
tizedespont. valós végére: R /ha. vagy' nem jelzi a valós számot/
duplaszavas valós végére: L, byte-konstans: "ek közé, vagy HEX X .

2.3. Deklarációk

2.3.1. Bevezetés

A PL360-ban az ALGOL-60-hoz hasonlóan minden programban használt azonositót deklarálni kell. Ez alól kivételt az u n. standard azonositók jelentenek csak.A deklarációk négy fő tipusba sorolhatók: regiszter-deklará-

ciók, cella-deklarációk, függvény-deklarációk és procedure-deklarációk. Megjegyezzük, hogy egy deklaráció érvényességi köre /scope/ a b l o k k később definiálandó fogalmával függ össze.

2.3.2. R e g i s z t e r - deklaráció

A regiszterek azonositóit nem kell deklarálni, ezek standard azonositók a PL360-ban:

R0,R1,R2,..., R14,R15	az egész regiszterek azonositói
F0,F2,F4,F6	a valós regiszterek azonositói
F01,F23,F45,F67	a duplaszavas valós regiszterek azonositói

Ezek az azonositók a gép fizikai regisztereit azonositják, tehát pl. F01 és F0 fizikailag nem különbözők, hanem F01-nek része az F0 - lásd [7], [8]. A regiszterek azonositói helyett egyszerüség kedvéért legtöbbször regiszterekről fogunk beszélni későbbiekben. /2.5.1-ben lesz még szó bizonyos regiszterdeklarációról/.

2.3.3. C e l l a- d e k l a r á c i ó és c e l l a - h i v a t k o z á s

A cella-deklaráció célja, hogy azonositókat rendeljen cellákhoz. Láttuk /2.2.1./ hogy a cellák ötfélék lehetnek, ennek megfelelően ötféle deklaráció van cellákra. Ezeket a byte short integer integer real és long real kulcsszavakkal különböztetjük meg. A kulcsszó leirása után azonositókat lehet felsorolni vesszővel elválasztva.

Pl.
```
byte flag
short integer   i, j, k
long real       x, y, Z1
```

Megjegyzés: byte és character, valamint integer és logical ekvivalensek. Lehetőség van még egydimenziós tömb deklarálására,

ez a PL360 legmagasabb szintü adatstrukturája. Ez formailag ugy történik, hogy az array kulcsszó után irjuk a tömb méretét /nem negativ, egész szám lehet csak/, majd a már megismert módon deklaráljuk a tömb egy elemét.

Pl.: array 3 integer t1 itt t1 három-elemü tömböt azonosit. Az egyes elemek egész tipusuak. /t1 tömb 12 byte-ot foglal a memóriából/. Előrebocsájtjuk, hogy a hozzáférés a tömbhöz index segitségével történik és t1/0/, t1/4/, t1/8/-al lehet hivatkozni az egyes elemekre. /Assembler-szemlélet!/

További példák:

```
array 132 byte line          /132x1 byte/
array 1000 real quant,price
                    /1000x4byte + 1000x 4 byte/
```

Megjegyezzük, hogy a cella-deklarációk lehetőséget adnak arra is, hogy k e z d e t i é r t é k e t adjunk a deklarált változóknak. Erre azonban csak a 2.6 fejezetben térünk ki.

A cella-deklarációval deklarált azonositó használatát a programban c e l l a - h i v a t k o z á s-nak nevezzük.

```
Pl.:    integer i - ez cella-deklaráció
        i:=12     - itt i cella-hivatkozás /értékadásban/
  vagy  array 5 real r - ez cella-deklaráció /tömb/
        r/4/:=3.14 - ez cella hivatkozás /értékadás a tömb
                     második elemére/
```

C e l l a - h i v a t k o z á s formája a következő lehet:

1. Index nélküli cella-hivatkozás: a deklarációban szereplő azonositó használata. Ha az azonositó tömb-deklarációban szerepel, akkor ez a forma nem megengedett.

2. Indexes cella-hivatkozás: a deklarációban szereplő azonositó

használata zárójelek közé tett index-el. Az index lehet:

egész konstans
egész regiszter /RO használata indexként tilos/
egész regiszter ± egész konstans

Az utóbbi két esetben az index értékét a regiszter aktuális tartalma határozza meg. Az indexes cella-hivatkozást akkor használjuk, ha a tömb valamelyik elemére kívánunk hivatkozni. A tömb i elemét az /i-1/ h indexszel tudjuk elérni, ahol h az egy tömbelem által elfoglalt byte-ok számát jelenti. Nem tilos, de nem is tanácsos indexes cella-hivatkozást alkalmazni abban az esetben, ha a megfelelő azonositó nem tömb.

2.3.4. Függvénydeklaráció

A PL360-ban függvényen nem a magasszintü nyelvekből ismerős függvényfogalmat értjük. Itt a függvény arra szolgál, hogy gépi kód is használható legyen egy PL360 programban. Ezt a használatot a függvény-hivás jelenti majd.

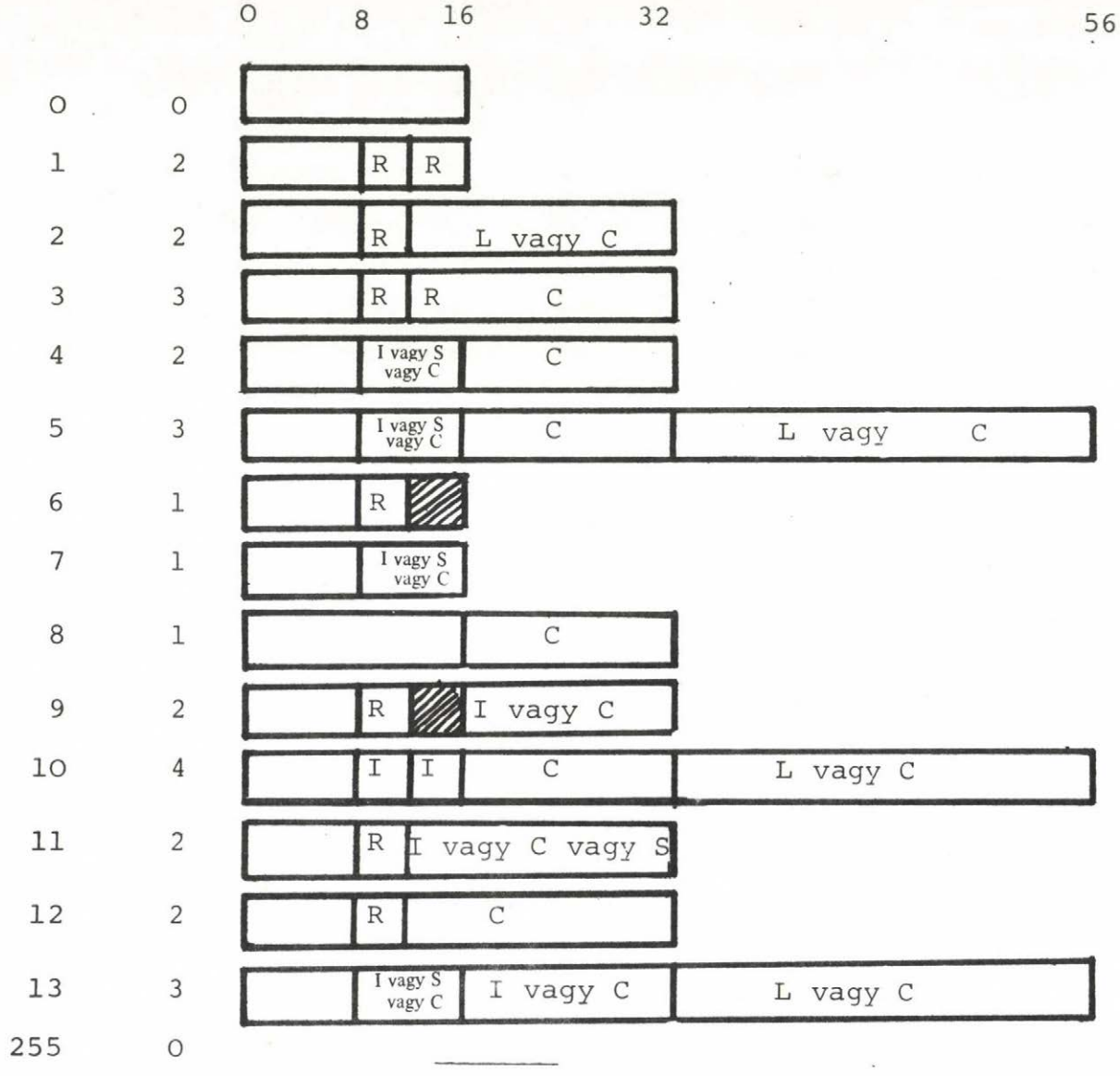
A függvénydeklaráció formája a következő:
function azonositó /formátumszám, utasitáskód/, ... ahol

azonositó: ezzel a névvel lehet majd meghivni a függvényt

formátumszám: 0,1,2, ..., 13, vagy 255 lehet. Ez definiálja a gépi utasitás formáját /RR,RX,stb./ továbbá az aktuális paraméterek számát és tipusát.

utasitáskód: konstans a gépi utasitás első két byte-ja /Ha csak az első byte-nak van jelentősége, akkor is két byte hosszon kell megadni. Célszerü hexadecimális konstanst irni/. A második byte-nak csak bizonyos függvényeknél van jelentősége.

A formátumszámokhoz tartozó gépi utasitások formáját és paraméterezését a következő táblázat definiálja:



R: regiszter azonositó I: egész konstans C: cella-hivatkozás S:string

L: konstans, melynek cime lesz az aktuális paraméter /literál/

Megjegyzések:
1. Ha C cella-hivatkozás 16 biten van elhelyezve, akkor nem indexelhető.
2. Ha C cella-hivatkozás 8 biten van elheylezve, akkor ez csak "displacement" lehet. (Lásd még: szinonim deklaráció, 2.5.)

Példaként felsoroljuk a standard függvények deklarációját /ezeket nem kell deklarálni PL360 programban/.

Function MVI (4,#9200), SET (8,#92FF), RESET (8,#9200)

Ez a három standard függvény ugyanahhoz a gépi utasításhoz tartozik, az utasításkód #92 /MVI utasitás/. A SET és RESET egy paraméteres függvény, mely a paraméter által hivatkozott byte tartalmát #FF-re, ill. #00-ra állitja.

Példa: A függvények hivása:
```
MVI ("0",ALFA)
MVI (5, BETA)
SET (ALFA)
RESET (ALFA)
```
ahol feltételezzük, hogy byte ALFA, BETA deklaráció előzte meg a függvényhivásokat.

További standard függvények: /tipusok szerint csoportositva/

function: MVC (5,#D200)
function: CLI (4,#9500), TEST (8,#95FF), CLC (13,#D500)

Megjegyzés: a CLI és TEST standard függvény ugyanazon gépi utasitást generálja híváskor /#95 utasitáskód: compere logical immediate/, azonban a TEST egy paraméteres. Mindhárom a feltételkódot állitja, mely PL 360 programban vizsgálható. /if, while utasitás, lásd később/.

function: LM (3,#9800), STM (3,#9000)
function: SRDL (9,#8000), SLDL (9,#8D00)
function: IC (2,#4300), STC (12,#4200)

Megjegyzés: az IC formátumszáma azért különbözik az STC formátumszámától, mert az IC-nél literál is megengedett aktuális paraméterként, pl. IC (R3, "A") függvényhivás lehetséges.

function: LA (11,#4100)
function: UNPK (10,#F300), CVD (12,#4E00), EX (2,#4400),
TR (5,#DC00), ED (5,#DE00)

Megjegyzés: Az EX /execute/ utasitás használatáról később külön szó lesz. /2.5./

Amennyiben a felsoroltakon kivül bármilyen más függvényre van szükségünk, deklarálni kell azt. Külön figyelmet érdemel a 255 formátumszám, ekkor ugyanis nem generál kódot a függvényhivás. Ennek egy alkalmazása lehet a függvény, ill. procedure-hivások egyszerü törlése.

```
Példák: function LTR (1,#1200)      deklaráció
                LTR (R3,R4)         függvényhivás
                if<then ...         feltételkód vizsgálat
```

2.3.5. Procedure-deklaráció

A PL360-ban a procedure nem az ALGOL-ból ismerős fogalmat jelöl, csupán annak egyetlen vonását tartalmazza. A p r o c e d u r e d e k l a r á c i ó egyetlen célja, hogy egy utasitást /mely lehet rendkivül bonyolult utasitássorozat: blokk is/ névvel lássuk el, majd ezzel a névvel hivhassuk végrehajtásra. A PL360 procedure-nak formális paraméterei nincsenek, továbbá procedure-ban deklaráció sem lehetséges. Mindez világos lesz a procedure-deklaráció formájából is, amely

procedure azonositó (egészregiszter); utasitás

Az azonositó lesz a procedure neve, a zárójelbe tett regiszter nem paraméter, hanem a visszatérési cim kerül a regiszterbe a procedure hivásakor. Ebből az következik, hogy az utasitásnak nem szabad elrontani ezen regiszter tartalmát. Ezt kétféleképpen lehet biztositani: a regisztert mentjük, majd visszaállitjuk a procedure vége előtt, vagy a regiszter használatát elkerüljük a procedure utasitásában. Procedure hivása egyszerüen nevének leirásával történik. /Procedure hivásról később lesz szó. /2.4.3./

2.4. Utasitások

A következő alpontok a PL360 utasitásait definiálják:

2.4.1. A PL360 blokk utasitása

A PL360 utasitásai közül a b l o k k alapvető fontosságu. A blokk fogalma közelitőleg megfelel az ALGOL blokk és compound fogalmának. [9]

A blokk egy olyan PL360 utasitás, mely a következő alaku:

begin D;D; ... D;S; ... S; end

D betük deklarációkat jelentenek, az S betük pedig PL360 utasitásokat. A blokk definiciója tehát rekurziv és pillanatnyilag semmitmondó, hiszen még egyetlen egyéb PL360 utasitást sem tárgyaltunk. Figyeljük meg, hogy minden deklarációt, ill. utasitást ; zár le, még az end előtti utasitást is.

Elképzelhető, hogy a blokkban egyetlen deklaráció sincs, legalább egy utasitás zonban kötelező.

A deklarációk csupán erre a blokkra és ezen blokk belső blokkjaira érvényesek és bővebb, vagy parallel blokkok deklarációit hatálytalanitják. /Ha ugyanazok az azonositók/. A blokkban előforduló összes cella-hivatkozás, function és procedure deklarálva kell hogy legyen - kivéve a standard azonositó esetét, valamint azokat melyek valamely, ezt a blokkot tartalmazó blokkban vannak deklarálva.

A blokk végrehajtásán a benne szereplő utasitások szekvenciális végrehajtását értjük.

A blokk bármely utasitásának lehet egy, vagy több c i m k é j e. A cimke azonositó, melyet : választ el a cimkézett utasitásttól, ill. többszörös cimkék esetén a cimkék egymástól is :-tal vannak elválasztva.

A cimkék a blokkban olyan pontokat jelölnek, melyekre go to utasitással lehet hivatkozni. Ugyanaz a cimke nem lehet többször egy blokkon belül. Figyeljük meg, hogy magának a blokknak sem lehet cimkéje /kivéve, ha a blokk egy másik blokkon belül utasitást képez/.

2.4.2. A go to utasitás.

A go to utasitás formája:

go to azonositó

Az utasitás végrehajtása a következő algoritmus szerint történik:

a/ Tekintsük a go to utasitást tartalmazó legszükebb blokkot.

b/ Ha ezen blokkban a go to azonositója cimke, akkor a a program ezen cimkéhez tartozó utasitással folytatódik. Ellenkező esetben a blokk végrehajtása befejezettnek tekintődik és az őt tartalmazó legszükebb blokkal indul ujra a b/ bont.

A go to utasitást PL360 programban viszonylag ritkán kell alkalmazni, mert egyéb utasitások /if, while, case/ áttekinthetőbbek és természetesebbek sok helyzetben.

2.4.3. Függvény és procedure hivás.

Függvény hivás a függvény nevének leirásával és zárójelek között az aktuális paraméterek vesszővel elválasztott felsorolásából áll. /Ha nincs paraméter, akkor csak a nevet kell leirni/. /2.4.1. szerint függvényhivás csak akkor lehetséges, ha a hivás olyan blokk belsejében van, amelyben a függvény deklarálva volt/.

Pl.: COPY /deklaráció: function COPY (0,#1812), ez a 2-es regiszter tartalmát másolja az 1-esbe.

LR utasitásnak felel meg./

CMPR 34 /deklaráció: function CMPR 34 (0,#1934), ez a CR utasitáson alapszik/.

X /R2, ALFA/ /deklaráció: function X (12,#5700), ez az X kizáró- vagy utasitás. Feltételezzük, hogy integer ALFA deklaráció is volt a programban/.

A függvényhivás eredményeként a paraméterek szerint összeállitott gépi kód kerül fordításkoz generálásra, ezért a függvényhivás /az assemblerhez hasonlóan/ gépi kód hatékonyságu. Nem jelenti ez azonban azt, hogy a PL360-ban érdemes tulságosan erőltetni a függvényhivásokkal való programozást. A PL360 ugy van megtervezve, hogy az ö s s z e s utasitása igen hatékony kódot generál. Gyakorlatilag azt lehet mondani, hogy a veszteség az assembler programozáshoz képest elhanyagolható.A másik végletes elképzelés sem helyes, mely szerint PL36o programban függvényhivásokat szinte soha nem kell alkalmazni. Bizonyos függvényekre igen gyakran szükség van, ezek azonban általában a standard-függvények között vannak. Ez óriási nyereség, hiszen 20 standard függvény van összesen és ezek közül is leginkább a következők használatosak:

MVI, MVC, SET, RESET, CLI, TEST, CLC, LM, STM,IC, STC, LA

Nem nagyon durva torzitása a helyzetnek, ha azt mondjuk, hogy a PL360 ezen 12 gépi utasitás /valójában ez csak 9/ használatát követeli meg leggyakrabban.

A procedure hivás egyszerüen a procedure nevének leirásával történik.
A 2.4.1. pontban mondottak itt is érvényesek - procedure hivás csak akkor értelmes, ha van olyan blokk, mely a hivást tartalmazza /a hivás ezen blokk begin-je és end-je között van/ és ebben deklarálva van a procedure. Rekurziv

procedure hivás tehát megengedett - ez azonban a procedure-ről mondottak alapján összehasonlithatatlanul egyszerübb, mint az ALGOL rekurziv procedure-hivás problémája. /Itt nincs aktuális-formális paraméter helyettesités és mivel PL360-ban nincs meg a procedure lokális deklarációinak fogalma, deklaráció azért a procedure lokális változóinak több példányban való tárolása sem merül fel/. Az egyetlen probléma a visszatérési cim megfelelő kezelésével van.

Példa rekurziv hivásra: /A példában használnak néhány utasitást, melyet csak később definiálnak, de ugy gondoljuk, hogy ezek nem zavarják a megértést/.

```
1 begin
2 procedure p (R9) ;
3     begin if R5=0 then  R8:=R9; visszatérési cim mentése;
4           if R5<10 then begin  R5:=R5+1 ;
5                                P ;
6                         end ;
7           R9:=R8; comment  a külső hivás visszatérési címének
                             visszaállitása;
8     end ;
9  R5:=0 ;
10 P ;
11 end
```

Megjegyzés: A program futása a következő lesz:

R5 tartalma 0 lesz /a legkülső blokk, melynek begin-je 1-es, end-je 11-es sorban van, első utasitása/

P procedure hivása /10. sorból/

R8:=R9 /R5=0 teljesül, a 10.sorbeli hivás utáni cim, mely az R9-ben van, R8-ba kerül./

R5:=R5+1 /R5<10 teljesül/

P procedure hivása /5. sorból/ ez R9-et megváltoztatja!/

Az előző két lépés ismétlődik meg kilencszer, az utolsó P hivásnál már R5=10 fennáll.

R9:=R8 visszatérési cim visszaállitása

A 8. sorbeli end a P procedure visszatérését eredményezi.

A 11 sor end . a legkülső blokkot terminálja.

2.4.4. A null utasitás.

Erre az utasitásra pusztán azért van szükség, mert mint láttuk minden blokknak legalább egy utasitást tartalmaznia kell. Lehet olyan helyzet, amikor nincs szükségünk utasitásra a blokkban, ekkor a null utasitást alkalmazzuk.

2.4.5. A case utasitás.

A case utasitás formája a következő:

```
case egész regiszter of  begin
                         utasitás ;
                         utasitás ;
                            .
                            .
                            .
                         utasitás ;
                         end
```

A megadott egész regiszter nem lehet R0. Az utasitás végrehajtása: a megadott regiszter tartalma határozza meg, hogy a begin és end közötti utasitások közül hányadikat kell végrehajtani. Ha a regiszter tartalma n, akkor az n. utasitás hajtódik végre, és a case utasitás befejeződik. Mellékhatásként a megadott egész regiszter értéke 4-gyel szorzódik.

Figyeljük meg, hogy /a blokk utasitáshoz hasonlóan/ a case utasitás is rekurziv definicióval van megadva.

2.4.6. Cella-értékadás.

A cella-értékadás utasitás formája a következő:

cella-hivatkozás: = regiszter

Az utasitás hatására a baloldalt hivatkozott cellába bemásolódik a regiszter tartalma. Problémát jelent, hogy különböző tipusu regiszterek, ill. cella-hivatkozások esetén ez a másolás mit jelent. A baloldal és jobboldal csak a következő párositásokban szerepelhet:

baloldal tipusa	jobboldal tipusa
egész	egész
félszavas egész	egész
valós	valós
valós	duplaszavas valós
duplaszavas valós	duplaszavas valós

Példák:
```
i : = R3          /deklaráció: integer i /
j( 8 ) : = R5     /deklaráció: array 3 integer j, a cella-hivatkozás a tömb 3.elemére történik/
j(4 ) : = R5      /deklaráció: array 3 short integer j, itt is a 3. tömbelemre van hivatkozás/
price ( R1 ) : = F0 /deklaráció: array 1000 real price/
```

Megjegyzések:

1. Érdemes ehhez a részhez a kódgenerálásról szóló fejezet megfelelő részét is átnézni. /5.3./
2. Első pillanatban megdöbbentő a cella-értékadás primitivsége, hiszen az i:=1, i=j értékadások nincsenek megengedve, nem beszélve az i:=i+1 stb. tipusokról, ezeket magasabb szintü nyelvekben megszoktuk. Ez a probléma azonban a PL360 szelle-

mében a 360 utasitás készlete miatt nagyon bonyolultan lenne csak megoldható. /Az i:=i+1 végrehajtásához regisztert kellene felvenni, mely a programozó számára rejtve marad/. Itt annyit jegyzünk csak meg, hogy a fenti értékadások standard függvények /MVI, MVC/ és / vagy a következő pontban definiálandó regiszter-értékadás segitségével oldhatók meg. /Lásd 2.4.7 B. pont 2.megjegyzés/

2.4.7. Regiszter-értékadás.

A regiszter-értékadás a PL360 elég erős eszköze. Megkülönböztetünk egyszerü regiszter-értékadást, ennek a fogalomnak azonban csak ebben a fejezetben van jelentősége, a regiszter-értékadás könnyebb megértését szolgálja.

A. Egyszerü regiszter-értékadás:

A1. regiszter : = konstans
A2. regiszter : = regiszter
A3. regiszter : = cella-hivatkozás
A4. Ide soroljuk az A1.-A3. eseteket, annyi variációval, hogy a jobboldalon a következő operátorok valamelyike állhat a := jel után: abs, neg, neg abs

Megjegyzés:

Az utasitások definiciója magától értetődik. Az előző szakaszhoz hasonlóan itt is csak öt tipus-pár van megengedve a baloldal és a jobboldal számára. Ezek:

baloldal	jobboldal
egész	egész
egész	félszavas egész
valós	valós
duplaszavas valós	valós
duplaszavas valós	duplaszavas valós

A5. egész regiszter : = string /ez A1-ben nem szerepel, mert a konstansok közé nem vettük be a string-et/ A string legfeljebb 4 karakterből állhat, ha rövidebb akkor balra 0 karakterekkel törlődik a regiszter.

A6. egész regiszter := @ cella-hivatkozás. Ezen konstrukció hatására a cella-hivatkozás abszolut cime kerül a regiszterbe. Ez lehetővé teszi pointerek használatát. Megjegyezzük, hogy LA /load address/ utasitásra fordul le ez az értékadás.

Példa egyszerü értékadásra:

R5:=i, R6:=j (R8), R7:=k (R5+3), F23:=X, F0:=Y (R1)

Ezek A3-ba tartoznak, integer vagy short integer deklarációban kell szerepelnie i,j,k-nak , X-nek real vagy long real deklarációban, y-nak pedig real deklarációban.

R5:=5, R6:=13 S, F0:=3.14, F23:=3.14159265359L

Ezek A.1. tipusu értékadások.

R3:=R5, F0:=F2, F23:=F0, F23:=F67

Ezek A2. tipusba tartoznak. Végül

R2:="ABCD", R4:="A", R9:=@i, R8:=@j (R3)

A5., ill. A6-ra adnak példákat.

B. Regiszter-értékadás általában

B1. Regiszter-értékadás jobboldalához aritmetikai operátort irunk, majd ez után egy konstansot, regisztert, vagy cella-hivatkozást, továbbra is regiszter-értékadást kapunk.

A r i t m e t i k a i o p e r á t o r o k:
+, -, *, /, ++, -- és = :

Az utolsó három igényel magyarázatot: a ++ és -- valós operandusok esetén nem-normalizált összeadás és kivonás, egész operandusok esetén pedig logikai összeadás és kivonás. Az = : operátor a baloldalán

lévő regiszter operandus tartalmát átviszi a jobboldalon lévő regiszter, vagy cella-hivatkozás tartalmába.

B2. Egész-regiszter értékadás jobboldalához logikai operátort, majd egész konstansot, egész regisztert, vagy egész tipusu cella-hivatkozást irva, továbbra is egész regiszter értékadáshoz jutunk.

L o g i k a i o p e r á t o r o k:

and, or, és xor /kizáró vagy/

B3. Egész-regiszter értékadás jobboldalához shift operátort, majd egész konstansot, vagy egész regisztert irva továbbra is egész-regiszter értékadáshoz jutunk.

S h i f t - o p e r á t o r o k:

shll, shla, shrl, shra

A B1 - B3-ban definiált regiszter értékadások végrehajtása balról jobbra haladva történik, a rész-eredmények /az=: kivételével/ mindig a kiindulási regiszterben vannak.

Pl.: R 10:=i+j-R3+k/8/ regiszter-értékadás végrehajtása

az R10 ← i
R10 ← R10+j
R10 ← R10-R3
R10 ← R10+k(8) lépések végrehajtásával ekvivalens. Ennek következményeképpen például az R1:=R2+R1 és R1:=R1+R2 értékadások nem egyenértékűek, mert az

első R1 ← R2, R1 ← R1+R1 , a második R1 ← R1, R1 ← R1+R2 lépésben

hajtódik végre

Megjegyzések:

1. Az A. pontban tett megjegyzés a tipusok vegyitéséről a B1-B3

pontokra is vonatkozik. Illegálisak például az R3:=5+3.14, R3:=i+3.14, FO:=j+3.14 értékadások, az utolsó csak akkor helyes, ha j valós-nak van deklarálva. /Ha j long real, akkor is helytelen/

2. Érdemes külön kiemelni az =: operátor használatát.
   Pl.: R3:=i+1=:i+6 értékadás lépései:

   R3 ← i
   R3 ← R3+1
   R3 → i
   R3 ← R3+6 tehát suba alatt /az önmagában tilos/

   i:=i+1 utasitás is végrehajtódott.

3. Szorzás és osztás egész operandusokkal ugy hajtandó végre, hogy az egész regiszter páratlan legyen. A szorzásnál az eredmény, az osztásnál a maradék az eggyel kisebb számu regisztert használja.

Példa: R3:=X*y+z értékadás után R2 elromlik.

Példák regiszter értékadásra:

```
FO:=quant ( R1 ) * price ( R1 )
R9:=R8 and R7 shll 8 or R6
R3:=2+j-k=:1
```

2.4.8. Egyszerü utasitások

A 2.4.1. - 2.4.8. szekcióban definiált utasitásokat e g y s z e r ü u t a s i t á s-nak nevezzük. A következő egyszerü utasitásokat definiáltuk tehát: blokk, go to, függvényhivás, procedure-hivás, null, case, cella-értékadás, regiszter-értékadás.

A PL360-ban további három utasitás van: for utasitás, if utasitás és while utasitás. Ezek nem egyszerü utasitások. Ezt a három utasitást definiáljuk a következő szekciókban.

2.4.9. A for utasitás.

Ez az utasitás ciklus szervezésére szolgál. Alakja a következő:

for egész-regiszter értékadás step növekmény until határ do utasitás

ahol a határ egész tipusu konstans, regiszter vagy cella-hivatkozás, illetőleg félszavas egész tipusu konstans, vagy cella-hivatkozás.
A növekmény egész konstans lehet csak.

A for utasitás végrehajtása a következő lépésekben történik:

1. A regiszter értékadás végrehajtódik. /A szóbanforgó regisztert kontrol-regiszternek nevezzük/.

2. Ha a növekmény nem negativ /negativ/, akkor a kontrol regiszter tartalma és a határ kerül összehasonlitásra. Ha a kontrol regiszter tartalma nem nagyobb /nem kisebb/, mint a határ, akkor a 3. lépés következik. Ellenkező esetben a for utasitás terminál.

3. A do utáni utasitás végrehajtódik.

4. A kontrol-regiszter tartalmához a növekmény hozzáadódik és a 2. lépés következik.

Példák:
```
for R1:=0 step 1 until n do  STC (R0,line(R1))
for R2:=R1 step 4 until R0 do
    begin  F23:=quant (R2)* price (R2) ;
           F01:=F01+F23 ;
    end
```

2.4.10. Az if utasitás

Az if-utasitásnak két alakja van. Az egyikben nincs else-ág, a másikban van.

Ennek megfelelően a két utasításforma:

if feltétel then utasitás és
if feltétel then egyszerü utasitás else utasitás.

Az utasitások végrehajtása nem igényel magyarázatot. Az, hogy a második formában a then ágon csak egyszerü utasitás lehet azt jelenti, hogy a for, if és while utasitások ki vannak tiltva innen. Természetesen begin és end közé téve, blokk formájában már nincs akadály, hiszen a blokk egyszerü utasitás.

A f e l t é t e l az if utasitás mindkét formájában a következő négy eset valamelyike:

1. Feltételkód-vizsgálat.

Bizonyos függvények a feltételkódot beállitják. A standard függvények közül a CLI, TEST, CLC és ED utasitások ilyenek és természetesen sok további függvény definiálható, mely a feltételkódot állitja. Állitják továbbá a feltételkódot a következő operátorok:

+, ++, -, --, and, or, xor, shla és shra

/Ezek az operátorok kivétel nélkül regiszter-értékadás jobboldalán fordulhatnak elő/.

A feltételkód vizsgálata ugy történhet, hogy a feltétel helyére az if utasitásban a következő szimbólumok valamelyikét irjuk:

=, ¬=, <, <=, >=, >, overflow, ¬overflow ¬mixed, mixed

A következő táblázat mutatja, hogy ezen szimbólumok használatával milyen feltételkód esetén lesz a feltétel igaz az if utasitásban.

A 0,1,2,3 a feltételkód értékeit jelzi. A mixed és ¬mixed a TM /test under mask/ függvény esetén célszerü használni.

szimbólum	feltétel
=	0
¬ =	1 vagy 2
<	1
> =	0 vagy 1
> =	0 vagy 2
	2
overflow	3
¬ overflow	0,1 vagy 2
mixed	1
¬ mixed	0 vagy 3

Példák: CLI (5, number);

if< then SET (flag (0)) else if = then SET (flag(1)) else SET (flag(2));

R2:=i+j

if overflow then

Az első példában a CLI standard függvény összehasonlitotta az 5-öt és a number értékét. Ha 5 number akkor flog (0) , ha 5=number akkor flag /1/ különben flog /2/ lesz beállitva. A táblázat első oszlopában álló szimbólum legtöbb esetben /összehasonlitás, aritmetikai müveletek/ tükrözi azt, amit vizsgálni akarunk.

2. Reláció

Ennek formája: regiszter után relációjel, majd konstans, regiszter, vagy cella-hivatkozás. A tipusok egyeztetése kötelező, ugyanugy, mint a regiszterértékadásban. /lásd 2.4.7. A4.pont utáni megjegyzés/.

R e l á c i ó j e l e k: =, ¬ =, < , < =, > =, >

A feltétel igaz, ha a reláció igaz.

Példák: if R1<10 then. . .
if R3> =i/8/ then . . .
if F2> =3.75 then

3. Byte-test.

A feltétel lehet egy byte tipusu cella-hivatkozás, vagy annak negáltja. A feltétel akkor teljesül, ha a hivatkozott byte minden bytje 1-es, /azaz tartalma # FF/.

Példa: if flag then . . .
if i (2) then . . .
/Flag és i byte tipusunak van deklarálva/.

4. Összetett feltétel:

Az 1-3 pontokban felsorolt feltételeket and operátorokkal, vagy or operátorokkal össze lehet kapcsolni. Az or és and vegyitése azonban nem megengedett.

Példák: if F2 > -3.1 and F2< 3.1 then . . .
if = or i> = 1 then . . .
if flag1 and flag2 then . . .

2.4.11 A while utasitás.

A while utasitás formája a következő:
while feltétel do utasitás
A feltétel ugyanugy van definiálva, mint az if utasitásban. /2.4.10./ A do után következő utasitás mindaddig végrehajtódik, amig a feltétel igaz.

2.4.12. A PL360 program felépitése.

Egy program PL360-ban a következőképpen néz ki:

utasitás .

Ez az utasitás általában blokk, kivéve, ha a program egyetlen utasitásból áll, és nincs deklaráció. /Lásd még 3.2/

Érdemes kiemelni, hogy a ; mindig egy blokk deklarációi és utasitásai után áll. Egyetlen kivétel a procedure deklaráció, ahol a procedure /egész regiszter/ ; -t kell irni és ezt követi egy utasitás.

2.5. Szinonim deklaráció

A szinonim deklarációnak két fajtáját különböztetjük meg:

2.5.1. Szinonim regiszter-deklaráció:

Ez arra szolgál, hogy valamely regisztert más néven is lehessen használni, mint a standard neve. Formája:

tipus register azonositó syn regiszter, azonositó
syn regiszter, . . .

A tipus az integer register és long real kulcsszavak valamelyike.

Például: integer register i syn R1, j syn R2
integer register j syn F0 hibás !
real register realnum syn F2

2.5.2. Szinonim cella-deklaráció.

Ez lényegesen bonyolultabb, mint a szinonim regiszter-deklaráció, de több felhasználása van. A szinonim cella-deklarációnak több tipusát különböztetjük meg:

A. tipus azonositó syn egész konstans. A tipus byte, integer, short integer, real, long real, vagy akár array

tipus is lehet. Az azonositó olyan cimre fog mutatni, melynek displecement-jét az egész konstans határozza meg. A konstans 0 és 4095 közé kell, hogy essen. A cimhez 0 bázisregiszter fog tartozni, azaz nincs bázisolva.

1. Alkalmazás: Tegyük fel, hogy 0 és 4095 közötti abszolut cimről kivánunk valamilyen adatot elolvasni /supervisor terület/.
   Az integer timer syn #50 szinonim deklarációval a timer cellahivatkozás a memória /hexadecimálisan/ 50-ik byte-jára történik.

2. Alkalmazás: Ha az A. tipusu cella-hivatkozásban indexet használunk, akkor az indexként használt regiszter bázisolni fogja a cimet, az indexben használt konstans pedig a displacement-hez adódik hozzá. Ilymódon táblázatok különböző elemeihez való relativ hozzáférést lehet megvalósitani.

```
Integer X syn 4, y syn 8, z syn 12
array 10 integer a,b,c

        R1 : = @a;
        R2 : = @b;
        R3 : = @c;
```

X/R1/ és a/4/, y/R2/ és b/8/, z/R3/ és c/12/ cella-hivatkozások egyenértéküek. Ezzel relativ cimek szimbólikus kezelése válik lehetővé.

3. Alkalmazás. 4,5,7 és 13 formátumszámu függvénydeklaráció esetén 1 byte-os cella-hivatkozás is lehet paraméter.

```
integer const syn 13
byte alfa
MVI /const, alfa) : ekkor az MVI függvény a
```

13 konstansot mozgatja az alfa byte-ba.

B. tipus azonositó syn indexelt cella-hivatkozás, ahol a tipus tetszőleges, a cella-hivatkozás pedig egy előzőleg A. tipusu szinonim deklaráció baloldalán szerepelt.

Példa:

```
integer i syn 24
integer j syn i (R2), ebben
```

az esetben a j cella-hivatkozáshoz 24 displacement, és 2-es bázisregiszter tartozik.

A s t a n d a r d c e l l a-d e k l a r á c i ó k A. és B. tipusuak. Ezek a következő módon vannak deklarálva:

```
integer MEM syn 0                    A. tipus
integer B1  syn MEM (R1),            B. tipus
        B2  syn MEM (R2),
             .
             .
             .
             .
        B12 syn MEM (R12)
```

Ilymódon tehát a B10 (5) cella-hivatkozás a 10-es regisztert bázisként használja a displacement 5. B10 (R5) cella-hivatkozás a 10-es regisztert bázisként és 5-ös regisztert indexként használja

C. tipus azonositó syn cella-hivatkozás, ahol a tipus tetszőleges, a cella-hivatkozás pedig előzőleg nem szinonim deklarációban volt deklarálva.

Példa:

```
array 5 integer  i
integer j syn i (4)
```

A következő példa azt mutatja, hogyan lehet egy egész számot, mely R1-ben van, duplaszavas valóssá konvertálni az F01-be.

```
long real x=(#4E0000000000000OL);
                    /Kezdeti értékadásról a következő
                     fejezet szól/
```

```
integer xlow syn x ( 4 );
```

```
A konverzió :  Xlow : = Rl; FOl:=X;
Forditott konverzió : FOl : = FOl ++ #4E0000000000000L;
                      X ; = FOl ; Rl : = Xlow;
```

Megjegyzés:

Mindhárom eddigi tipus bonyolitható szinonim deklarációk láncolatával. Könnyen látható azonban, hogy ez nem változtat a három alapvető tipuson. Érdemes megjegyezni, hogy A. esetben a regiszterrel indexelt cella-hivatkozás regisztere bázisként, B. és C. esetben indexként kerül forditásra.

2.6. Kezdeti értékek deklarációban

2.6.1. Kezdeti értékek egyszerü tipusu cella esetén.

A kezdeti érték beállitása a deklarált azonositó után = karakter és egy konstans, string, vagy cim leirásával történik.

a/ Konstans kezdeti érték adásnál a következő táblázat mutatja a megengedett tipusokat:

cella tipusa	konstans tipusa
byte	egész, félszavas egész, byte konstans
félszavas egész	egész, félszavas egész
egész	egész, félszavas egész
valós	valós, duplaszavas valós
duplaszavas valós	valós, duplaszavas valós

Ha az elhelyezendő konstans tul hosszu, akkor balról levágásra kerül.

Példák: byte b="A", c=#A3X, d=5, e=1005S, f=3000

/e és f kezdeti értéke nem fér el egy byte-ban, ezért a megfelelő szám utolsó byte-je lesz ábrázolva/

short integer i=105, j = 35

real X=3.14

b/ String kezdeti érték tetszőleges tipusu cella esetén megengedett. Vigyázni kell azonban, mert stringet a forditó soha nem vág /ha hosszabb, mint a deklarációban szereplő tipus hossza/ és nem egészit ki /ha rövidebb, mint a deklarációban szereplő tipus hossza/. A kezdeti értékadás tehát akkor müködik csak jól stringek esetén, ha a string hossza megegyezik a megfelelő tipus hosszával.

Példák helyes értékadásra stringgel:

byte a="A" - megjegyezzük, hogy itt "A" string, - de ugyanakkor byte-konstansnak is felfogható.

```
short integer  i="AB", j="CD"
integer        k="ABCD"
real           X="ABCD"
long real string= "ABCDEFGH"
```

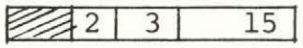
c/ Cim kezdeti érték egész, vagy félszavas egész tipusu cellának adható.
Ekkor az egyenlőségjel jobboldalára a

(a) cella-hivatkozás kerül. Ennek hatására a tárolt kezdeti érték a bázis-displacement pár /ha félszavas egésznek adunk kezdeti értéket/, bázis-displacement és esetleges index regiszter /ha egésznek adunk kezdeti értéket/. Az utóbbi esetben csak az alsó 20 byt kerül kitöltésre.

Példák: byte i ; /tegyük fel, hogy d=16 b=3/
short integer addr=(a) i

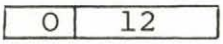
addr | B | 16 |

```
integer addr1 =  i /R2/
```

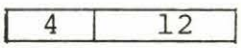


Érdekes a szinonim deklarációval való kapcsolat:

```
integer j syn  12
short integer addr 2 = @j
```

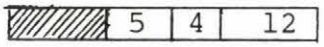



```
short integer addr3 = @j  (R4 )
```



Következő példa:

```
integer k syn 12;
integer l syn k (R4 );
integer addr 4 syn @l (R5 )
```



A következő példa illegális:

```
integer i ;
short integer  j= @i (R5 )  /j-ben nem fér el
                             az 5-ös index re-
                             giszter/
```

Alkalmazásként bemutatjuk, hogyan lehet adatot utasitásként használni. Ez a lehetőség ugyan ellentétes a PL360 szellemével /adat és program a deklarációk miatt elkülönül/, de bizonyos speciális szituációkban mégis megengedett. A módszer szerencsére elég bonyolult ahhoz, hogy nagyon megfontolja az ember, mikor nyul hozzá.

Tegyük fel, hogy egy CLC /compare logical characters/ utasitást akarunk alkalmazni. Ez megoldható lenne a CLC függvény deklarációjával, azonban a program végrehajtása során különböző hosszuságu mezők összehasonlitására van szükség és a hosszuságokat nem tudjuk előre. Ráadásul a második cim is változik a CLC-ben. /D2, B2/

```
array  40 integer cim 1 /tegyük fel, hogy ciml bázisa:
                          9-es regiszter, displacement=100/
short integer clcl= D500, clc2=@ciml, clc3=@B3
                          /B3 standard azonositó lásd 2.5.2./
```

ekkor a kezdeti értékek beállitása után

clcl		clc2		clc3	
D5	00	9	100	3	0

A programban az EX /execute/ standard függvénnyel lehet a vezérlést átadni.

Például:

```
R3:=@cim2; Rll=23;
EX (Rll, clc 1);
```

ekkor 23 hosszuságban az összehasonlitás ciml és cim2 között.

```
R3:=@cim3;  Rl2:=length;
EX (Rl2, clc 1);
```

ekkor a length cella-hivatkozás adja meg az összehasonlitás hosszuságát cim 1 és cim3 között.

2.6.2. Kezdeti érték beállitása array esetén:

Tömbök esetében a kezdeti értékek beállitása zárójelek közé tett listával történhet. A lista elemei vesszővel vannak elválasztva.A nyitó-zárójel előtt ismétlési tényezőt lehet alkalmazni. Lista-elem ismét lehet zárójelek közé tett lista. Ismétlési tényező itt is megengedett. Egy tömbelemre vonatkozó értékadás szabálya megegyezik a 2.6.1.-ben leirtakkal. Legjobb példán keresztül megmutatni a tömb-értékadás lehetőségeit:

```
array 3 integer a=(5,6,13)
array 132 byte line = 132 ( " " )
array 132 byte buffer = 33 (" " ,2 ("x"),"  ")
```

ekkor buffer tartalma: blank xx blank xx blank stb.

```
array 15 integer x=3 ( line,"ANCD",3(5) )
```

Ha tömb eredeti értékadásban a tömb méreténél kevesebb elemet sorolunk fel /ismétlési tényezők figyelembevételével természetesen/, akkor a magasabb indexü tömbelemek definiálatlanok maradnak. Van egy olyan lehetőség is, melyben a tömb méretét éppen kezdeti értékadással definiáljuk:

array datadef tipus = (. . . .)

Példa: array datadef integer (3,12,25,7 (4), 19 (3))-ez egy 29 elemü tömböt definiál.

3. PROGRAM ÉS ADATSZEGMENTÁCIÓ

A 2. fejezetben a PL360 minden lehetőségét megismertünk. A 360 konstrukciója folytán azonban csak "kicsi" programokat tudunk irni "kevés" adattal. A következő két példa illusztrálja, mire gondolunk:

```
go to X ;                                begin
     .   }                               array  4096 byte a ;
     .   } 4095-nél                      array  100  byte b ;
     .   }                                                    }
     .   }                                    R12:=b (2 );    }
X ;  .   } programrészlet                                     }
     .   }                               end                  }
```

Az első példánál a go to x olyan cimre hivatkozik, mely tulságosan távol van, a második példánál a b tömb hivatkozás az a tömb deklarációja miatt nem elérhető. Mindezek a problémák a program és az adatok "bázisolásával" függenek össze.

3.1. A Szegmens fogalma.

S z e g m e n s n e k nevezünk egy olyan program, vagy adatrészletet, melynek cimzéséhez ugyanazt a bázisregisztert

használjuk. A PL360-ban a programra R15, az adatokra R13 bázisregiszter van kijelölve automatikusan. Ezen regiszterek használatát ezért kerüljük !

3.2. Program szegmentálás.

Lehetőség van arra /és ez nagyobb program esetén elkerülhetetlen/, hogy PL360 programot szegmensekre bontsunk. Ez PL360-ban teljesen a programozó feladata. /Automatikus programszegmentáció is elképzelhető lenne/.
Szegmenst procedure-ból képezhetünk ugy, hogy a procedure deklarációjában segment procedure-t irunk. Egy programrészletből szegmenteket kell képezni akkor ha hossza /hosszon a bennük lévő utasitások összhosszát értjük, mégpedig gépikódban/ 4095 byte-nál nagyobb. Ennek becslését elég könnyü elsajátitani. Ha egy programrészlet 4095-nél hosszabb és nincs szegmentálva, akkor a fordító hibaüzenetet ad. A főprogram /a legküldő blokk utasitása/ automatikusan szegmensnek számit.

Példa:
```
begin
   procedure p1 (R1); begin . . . . . end;
   segment procedure p2 (R1); begin . . . . end;
   comment főprogram indul;
   p1;
   p2;
    :
    :
   end
```

A példán egy teljes PL360 program vázát látjuk. Két szegmens van, egyik a főprogram és a p1 procedure együttesen, a másik a p2 procedure. A szegmentáció akkor jó, ha p1 és a főprogram együttes hossza, valamint p2 hossza 4096-nál kisebb.

A segment procedure deklaráció hatására a procedure utasitásaiból egy kontrol szekció /CSECT/ generálódik. Ennek

neve is lesz, melyet a forditóprogram ad. /A név SEG nn alaku/. Elérhető az is, hogy a generált CSECT-név a procedure neve legyen. Ekkor a global procedure deklarációt kell alkalmazni. További fontos lehetőség az external procedure deklaráció. Ebben az esetben a fordító nem generál tárgykódot a procedure-ről. Ha futtatni akarjuk a programot, akkor az external procedure nevével egy CSECT-nek rendelkezésre kell állnia. /A szerkesztést a Linkage-editor végzi/. Az external procedure tehát külön forditott /PL360, vagy egyéb nyelven irt/ programok beszerkesztését teszi lehetővé.

Példa:

```
begin
external procedure p1 (R1); null;
external procedure p2 (R1); null;
p1;
p2;
end
```

Ez a program müködik, ha végrehajtás előtt összeszerkesztjük a már leforditott

```
global procedure p1 (R1); begin ..... end.
```

és

```
global procedure p2 (R1); begin ..... end.
```

programokkal.

Amint a példákból kiderül, az external és global procedure deklarációk segitségével lehet függetlenül forditott PL360 összeszerkeszteni. Itt egy kiegészitést kell tennünk a PL360 program definiciójához: /Lásd 2.4.12./

Egy PL360 program kétféleképpen nézhet ki:

A: utasitás
B: global procedure

Az A. esetében a program végrehajtható, a B. esetben azon-

ban nem, ekkor a felhasználása egy másik PL360 programból történhet, ha ott external-ként deklaráljuk. Ebben a programban null utasitást kell irni az external procedure-be, hogy ne sértsük meg a PL360 szintaxist.

Az entry procedure a lokális és globális procedure előnyeit egyesitik. Ha ugyanabból a programból hivjuk,ahol deklarálva van, akkor ugy viselkedik, mint egy közönséges procedure, de külső szegmensből is meghivható./Neve bekerül az ESD táblába /10/ /.

Példa:

```
global procedure G (R3) ;
begin
        entry procedure E (R11); begin . . . . end;
        .
        .
        .
        E;
end;
E;
```

Az első E hivásnál lokális procedure-ként viselkedik E, azaz R15 nem kerül ujratöltésre. A második hivás sem illegális, ekkor azonban R15 ujratöltődik.

3.3. Adatszegmentálás.

Ha egy PL360 programon utasitás . alaku, akkor az összes blokk összes adata az R13 bázisregiszterrel cimződik. Ha PL360 program global procedure, akkor nincs automatikus bázisregiszter-hozzárendelés az adatokhoz. A programozónak lehetősége van arra, hogy megszabja, milyen bázisregiszterrel cimezze az adatait. /Megjegyezzük, hogy program esetén csak a szegmentálást tudja befolyásolni a programozó - ez azonban elég is/. Ez b á z i s - d e k l a r á c i ó v a l történik, melynek formája:

segment
global data azonositó
external data azonositó
common data azonositó
common
dummy

} valamelyike, utána base egész regiszter

Ezen deklarációk hatását megszünteti a blokk vége, vagy explicite megszüntethető a close base deklarációval. A bázis deklaráció hatására a blokkba való belépéskor a megadott bázisregiszter értéke automatikusan betöltődik. /Kivétel: dummy bázisdeklaráció./

Példa:
```
begin array 100 integer i;
      segment base R3;
      array 1000 integer j;
      close base ;
      array 1000 integer k;
            .
            .
            .
end
```

Ebben a programban i és k tömböket R13 cimzi,/ implicit hozzárendelés/ a j tömböt R3 . /A j tömb egy CSECT lesz, melynek nevét a fordító generálja/.

Nézzük meg, mi a különbség a kétféle bázisdeklaráció között:

1. Segment base esetén a fordító által generált névvel /SEG nn/ az ide tartozó adatok egy CSECT-et fognak alkotni.

2. Global data azonositó base esetén az ide tartozó adatok CSECT-et fognak alkotni, a CSECT neve az azonositó.

3. External data azonositó base esetén nem készül CSECT az adatokból, ez a CSECT /az adott névvel/ kivülről kerül beszerkesztésre. Megjegyezzük, hogy az 1, 2, 3 eset a programszegmentálással analógiát mutat.

4. Common data azonositó base deklaráció esetén u.n. cimkézett common CSECT keletkezik az ide tartozó adatokból. Ezt fel lehet használni például Fortran-ban irt programmal való adatcserére. /Lásd részletesebben [10] /

5. Common base deklaráció esetén az ide tartozó adatok u.n. blank common CSECT-et alkotnak. /Lásd részletezve [10] /

6. Dummy base deklaráció dummy CSECT-et generál a hozzá tartozó adatokból. Az ide tartozó adatok nem érhetők el, csak a cimeik léteznek. /Lásd részletesebben [10] /
   Az ilyen blokkba való belépésnél a bázis-regiszter nem töltődik fel.

Megjegyzés:
Párhuzamos blokkban nyugodtan használható ugyanaz a regiszter bázis-deklarációban. Ugyanabban a blokkban a bázis-deklarációknak különböző regisztereket kell használni. Egymásba skatulyázott blokkok esetén célszerü különböző regisztereket megadni a bázis-deklarációkban - ellenkező esetben a programozó dolga a regiszter megfelelő betöltése.

3.4. Külső rutin hivása PL360-ból.

Ha pl PL360 procedure-ból meghivjuk p2 external procedure-t, akkor a következő a hivási konvenció: /assembler-ben leirva/

```
      USING  P1,15
P1    DS     OD
      :
      :
      L      15,=V(P2)
      DROP   15
      BALR   n,15
      USING  *,n
      L      15,=A(P1)
      USING  P1, 15
      DROP   n
```

Ez a konvenció a hivott rutintól a következők tudomásulvételét kivánja:

/1/ Belépéskor R15 az entry-pont cimét tartalmazza,
external procedure p2 /Rn/-ben szereplő Rn regiszter tartalmazza a visszatérési cimet,

/2/ Visszatérés előtt a visszatérési cimet Rn-be vissza kell tenni. /Vagy nem szabad elrontani/

/3/ R15-ben nem szabad információt visszaadni a pl-nek /mivel R15-öt a visszatérés után rögtön elrontja a hivó/

Vegyük észre, hogy ha p2 is PL360-ban /jól/ van megirva, akkor /1/, /2/, /3/ teljesül.

3.5. Supervisor hivások.

SVC utasitásokkal /függvényként deklarálva őket/ kilépünk a supervisorba. Vigyázni kell arra, hogy az SVC hivás elronthatja az R15-ös regisztert, ezért mentését programozni kell. /Egyéb regiszter-rontások is okozhatnak bajt, pl. R13-é !/

3.6. PL360 rutin hivása külső rutinból.

Ha pl külső rutin meghivja a /helyesen megirt/ p2 PL360 procedure-t, akkor a hivó tartsa be a következőket:

/1/ R15 a híváskor tartalmazza p2 belépési programját.

/2/ A visszatérési cimet tegye be p2 deklarációjában szereplő n-es regiszterbe. A visszatérés automatikus, és a visszatérési cim visszatéréskor is benne van az n-es regiszterben.

Abban az esetben, ha p2 utasitás . alaku volt /nem global procedure/ a következőt kell tudni:

- a belépési pont neve SEG 01
- a visszatérési cimet R14-be kell tenni
- R13 egy 18-szavas mentési terület cimére mutasson, mert p2 ezt használja

- visszatéréskor a regisztereket p2 visszaállitja
- R15-ben return-kód jelenhet meg.

4. TUDNIVALÓK A FORDITÓPROGRAMRÓL

4.1. A PL360 szimbólumainak reprezentációja

Csak nagybetük állnak rendelkezésre. A kulcsszavakat, melyeket a nyelv leirásában /2. fejezet/ aláhuzással jelöltünk, a konkrét forditóprogram számára nem kell semmivel kitüntetni. Ennek következtében kulcsszavakat nem szabad azonositóként használni. Kulcsszavak, azonositók és konstansok belsejében blank karaktert nem szabad irni. Egymás után következő kulcsszavak, azonositók és konstansok közé legalább egy blank karaktert kell irni. Egyébként blank karakterek tetszőlegesen alkalmazhatók.

A megengedett alapjelek: nagybetük, számjegyek és speciális karakterek közül a következők:

+ - * / ( ) = < > ¬ , ; . : _ " ' és a := kombináció

További alapjelek a kulcsszavak:

DO IF OF OR
ABS AND END FOR NEG SYN XOR
BASE BYTE CASE DATA ELSE GOTO LONG NULL REAL SHLA SHLL SHRA SHRL STEP THEN
ARRAY BEGIN CLOSE DUMMY SHORT UNTIL WHILE ENTRY MIXED
COMMON GLOBAL
COMMENT INTEGER LOGICAL SEGMENT
EXTERNAL FUNCTION OVERFLOW REGISTER
CHARACTER PROCEDURE

4.2. Standard azonositók

R0, R1, R2, R3, ..., R13, R14, R15
F0, F2, F4, F6,

```
F01, F23, F45, F67              /Regiszter-azonositók/
MEM
B1, B2, B3, ..., B12            /Cella-azonositók/
LA, MVI, MVC, CLI, CLC, LM, STM, SLDL, SRDL, IC, STC, CVD,
UNPK, ED, EX, TR, SET, RESET, TEST
                                /Függvény-azonositók/
```

4.3. Standard eljárások

READ: 80 karakteres rekordot olvas SYSIN-ről /OS/, ill. SYSIPT-ről /DOS/ az R0 regiszter által adott cimre. A feltételkódot 0-ra állitja, ha nem volt eddig of file, egyébként 1-re
/R0:=buff; READ; if=then noteof else eof;
példa end of file vizsgálatra./

WRITE: 133 karakteres rekordot ir SYSPRINT-re /OS/ ill. SYSLST-re /DOS/ a R0 regiszter által adott cimről. Az első karakter kocsivezérlés.

PUNCH: 80 karakteres rekordot ir SYSPUNCH-ra /OS/ ill. SYSPCH-ra /DOS/ az R0 regiszter által adott cimről. /Fizikailag 81 karakter iródik ki, az első karakter vezérlőkarakter/.

4.4. Forditóprogram imputja és listája

A forditóprogram 80 karakteres rekordokat vár, az első 72 karakteren lehet elhelyezve a PL360 program, az utolsó nyolc karakter csak listázásra kerül. Egy rekord 72. karaktere után a következő rekord első karaktere következik.

Az output listán a hibás sorok mindig listázásra kerülnek. A hibás sorokban pozicióindikátor mutatja az utoljára beolvasott karaktert. A hibaüzeneteket függelékben adjuk meg. A baloldalán a következő információkat találjuk:

Program szegmens száma, a kurrensprogram tárgykód relativ cime /hexadecimális/, adatszegmens száma, a kurrens adat relativ cime. Ezután a forrásprogram rekordjainak sorszáma következik. A forrásrekord listája után a jobboldalon a blokkok szint-száma van feltüntetve.

4.5. Listázás vezérlése.

A listevezetést $ -al kezdődő kártyák szabályozzák: /ezek nem kerülnek listázásra/.

$ PAGE: lapot vált

$ LIST: listázás /ha nem irjuk, van lista/

$ NOLIST: listázás letiltása

$ TITLE: uj lapot kezd és a $TITLE kártya 10-67 pozicioján lévő szöget helyezi el fejlécként az uj lapra és minden további lapra. /További $ TITLE kártyáig/

$ nn: az nn szám 1-31 értékek valmelyike.

nn=1 : ESD /External symbol Dictionary/ nyomtatása

nn=2 : változók, függvények, procedure-ok neveinek kiemelése

nn=4 : tárgykód hexadecimális nyomtatása a szegmensek végén

nn=8 : kulcsszavak aláhuzása

nn=16: makrokifejezések nyomtatása /+++-al jelezve/

Az opciók kombinálása a megfelelő nn értékek összeadására történik. Pl.: $20: tárgykód dump-ja és makrókifejezések listája.

4.6. A COPY lehetőség.

$ COPY numbernév hatására a rendszerkönyvtárból lemásolásra kerül a megnevezett forrásprogram-részlet. DOS operációs

rendszer esetén a keresés először a rendszer privát source könyvtár /SYSSLB/-ből /ha IIASSIGN-al megadtuk/ utána a rendszer source-könyvtárból. Mindkét esetben a P. - alkönyvtárból - történik. A membername paramétert a 7. oszloptól kell irni. A bemásolt rekordok elé + jel kerül. A mésolás után a forrásprogram olvasása SYSIN-ről /OS/ illetve SYSIPT-ről /DOS/ folytatódik. A bemásolásra kerülő forrásprogram nem tartalmazhat ujabb $COPY-t.

4.7. Limitációk

1. Azonositóknak csak az első 10 karaktere számit értékes karakternek.
2. Goto utasitással nem lehet szegmensből kiugrani.

4.8. A forditóprogram hivása és paraméterek átadása.

A forditóprogram a következő paramétereket tudja értelmezni:

LINECNT=nnn az egy lapra irható sorok maximális száma. Alapértelmezés:60

MAXERROR=nnn a hibajelzések maximális száma. Alapértelmezés: 50

MAXSEGS =nnn a definiálható szegmensek maximális száma. Alapértelmezés: 120

A forditó hivása és esetleges paraméterek átadása:

DOS	OS
//EXEC PL360	//EXEC PL360 PARM=' paraméterek
$$ paraméterek vesszővel elválasztva	vesszővel elválasztva

Lehetséges még bizonyos, az operációs rendszerben használatos paraméterek átadása.

Ezek:

Paraméter	file DOS-ban	file OS-ban	Alapértelmezés
LIST, NOLIST	SYSLST	SYSPRINT	LIST
LINK, NOLINK	SYSLNK	-	LINK
LOAD, NOLOAD	-	SYSGO	LOAD
DECK, NODECK	SYSPCH	SYSPUNCH	NODECK

A paraméterek átadása DOS-ban //OPTION vezérlő utasitással, OS-ben az //EXEC vezérlő utasitás PARM paraméterével történik.

5. FÜGGELÉKEK

5.1. Makrólehetőség

5.1.1. Makrók definiálása.

Egy definicióban egy vagy több makrót definiálhatunk. A makrókat ugy definiáljuk, hogy a MACRO kulcsszó után felsoroljuk a m a k r ó d e f i n i c i ó k a t, vesszővel ellátva.

Egy m a k r o d e f i n i c i ó a következő alaku:

azonositó = string & vagy

azonositó /paraméter, paraméter, .../ = string &

Az első esetben paraméter nélküli, a másikban paraméteres makróról beszélhetünk. A paraméterek azonositók lehetnek, vagy & után irt azonositók. Az utóbbi lehetőség nem lényeges, csak annak kiemelésére szolgál, hogy paraméterről van szó - ugyanis az = utáni string-ben & előzi meg a paramétereknek megfelelő azonositókat. A paramétereket f o r m á l i s p a r a m é t e r e k n e k is nevezzük, megkülönböztetésül a makrohivásban használt a k t u á l i s p a r a m é t e r e k t ő l.

Az = jobboldalán álló string bármilyen karaktert tartalmazhat. Egyes karaktereknek speciális jelentőségük van:

& jelzi, hogy formális paraméter következik. Az & után olyan azonositónak kell állni, mely az = baloldalán paraméterként szerepel. Az & jelzi egyben a makrodefinició végét. Ha & -et akarunk értékes karakterként elhelyezni a string-ben, && -t kell irnunk. A string-et lezáró & után nem állhat betü, sem másik &.

. jelzi a paraméter végét, ha a paraméter után betü, számjegy, vagy (következik. Ha .-ot értékes karakterként akarunk elhelyezni a string-ben, ..-ot kell irnunk.

Példa: MACRO PUSH = R1O:=R1O+SVLG; STM(R14,R15,B1O) &; ez paraméter nélküli makró. Makróhivás esetén az = jobboldala bemásolódik & előtti karakterig.

```
MACRO ADD1 (X)= R14:= &X+1&;  /MACRO ADD1 (&X)=....
                                ugyanezt jelentené/
MACRO TESTFLAG (F) = IF &F.THEN GOTO  X& ;
```

Itt F.-ot kellett irni, különben az F paramétert nem ismerné fel a makróprocesszor.

A makrodefinició jobboldalán makróhivások /lásd később/ is szerepelhetnek, rekurziót azonban tilos alkalmazni. Makródefiniciót viszont tilos elhelyezni a string-ben, továbbá a makróhivás sem eredményezhet makródefiniciót.

Makródefinicióban irhatunk olyan változókat, melyek nincsenek deklarálva, a makróhivást azonban ilyenkor csak a deklaráció után alkalmazhatjuk. Például a ADD1 makrót az X változó deklarációja után alkalmazhatjuk csak.

5.1.2. Makróhivás

A m a k r ó h i v á s ugy történik, hogy leirjuk a makró nevét, és /ha a definicióban voltak paraméterek/

zárójelek között, vesszőkkel elválasztva annyi karaktersorozatot, ahány formális paraméter szerepelt a makródefinició baloldalán. Ezeket a karaktersorozatokat a k t u á l i s p a r a m é t e r e k n e k nevezzük.

A makróhivás hatására makródefinicióban szereplő string bemásolódik a forrásprogramba ugy, hogy a formális paraméterek helyett az aktuális paraméterek szerepelnek.

Példa: az ADD1 (I3) hivás hatására

R14:=I3+1 másolódik a hivás helyére.

Az aktuális paraméterek helyettesitésére a következő szabályok érvényesek:

1. " /idézőjel/-ek közé tett karaktersorozat a nyitó és záró idézőjelekkel együtt másolódik. Az idézőjelek közötti string mindig egy aktuális paraméter része, tehát vesszők, zárójelek is másolásra kerülnek.
   Idézőjel kettős idézőjelként helyezhető el, mindkét idézőjel másolásra kerül. /A dupla idézőjelet majd a a forditóprogram értelmezi egy idézőjelnek/. Például

   X"(ABC,EFG,H)"Y aktuális paraméter helyére X"(ABC,EFG,H)"Y helyettesitődik "A" "B" " " aktuális paraméter helyére "A" "B" " " kerül. /A forditó ezt az A "B" string-nek értelmezi/.
   Kivételt képez az idézőjelek közötti ' /aposztrof/ karakter, melynek szabályait 2. alatt találjuk. Egyszeres idézőjeltől egyszeres idézőjelig tartó karaktersorozatot "- s t r i n g-nek nevezzük.

2. ' /aposztrof/ karakterek közé zárt karaktersorozat a nyitó és záró aposztrof nélkül másolódik. Az aposztrofok közötti string mindig egy aktuális paraméter része, tehát vesszők és zárójelek is másolásra kerülnek. Aposztrof karakter kettős aposztrofként helyezhető el, egyik közülük nem kerül másolásra. Egyszeres

aposztróftól egyszeres aposztrofik tartó karaktersorozatot '- s t r i n g n e k nevezzük.

X'(ABC,EFG,H)'Y X(ABC,EFG,H)Y-al
X"ABC' 'DEF''''G'Y pedig X ABC 'DEF''G Y-al helyettesitődik

X'ABC "DEF""G'H"I példa átlapoló '-stringre és "-stringre

Ez X ABC"DEF""G H"I-ként helyettesitődik.

3. A kezdőzárójel, végzárójel és vessző karaktereket a k t u á l i s p a r a m é t e r d e l i m i t e r e k n e k nevezzük. Ezt a.p.d.-vel röviditjük. Egy a.p.d. s z a b a d, ha nincs benne "-stringben vagy '-stringben. A z á r ó j e l e z é s i s z i n t e t a következőképpen definiáljuk:

Kezdetben a zárójelezési szint O, minden szabad nyitózárójel eggyel növeli minden szabad csukózárójel eggyel csökkenti a zárójelezési szintet. /Balról jobbra haladva/.

4. Az aktuális paraméterek megállapitása.
A makróhivást követő nyitó zárójel után van az első aktuális paraméter kezdete. Vége az első szabad 1-es szintü vessző, vagy az első 1-szintü csukó zárójel előtti karakter. Az első esetben, ha a szabad vessző fejezte be a paramétert, akkor a következő aktuális paraméter összeállitása kezdődik. Az utolsó aktuális paraméter az 1-es szintü csukó zárójel előtti karakterig tart. Ha az aktuális paraméterek száma kisebb, mint a formálisaké, akkor a maradék formális paraméterek helyébe az üres string kerül, ha pedig több van, akkor a maradék aktuális paraméterekkel nem történik helyettesités.

Példa: M1 (X, M2 (A,B),Y) hivás aktuális paraméterei X M2(A,B)és Y

5.2. PL360 összefoglalás

Kitüntetett regiszterek: R15 : program-bázisregiszter /implicit/

R13 : adat-bázisregiszter /implicit/

R14 : visszatérési cim, ha a program utasitás . alakban volt megadva./Ha a program global procedure alakban van, akkor deklarálni kell a visszatérési cim regiszterét/.

RO : tilos használni indexként, case utasitásban.

Deklarációk /D/: byte /character/, integer /logical/, short integer, real, long real
array egész of tipus és array datadef tipus = (....)

function név (formátumszám, utasitáskód)

[segment / global / external / entry] procedure név (egész regiszter); utasitás

[segment / common / dummy] base egész register, [global / external / common] azonositó

base egész regiszter, close base

Cellahivatkozás /CH/ azonositó, azonositó /index/ ahol
index: egész reg± egész konstans

Utasitások /U/ és egyszerü utasitások /EU/

Blokk:	begin D;D;...D;S;S;...S; end	EU
goto :	go to cimke	EU
Fv. és proc.hivás:	név leirása, függvénynél paraméterekkel	EU
null :	null	EU
case :	case egész reg of begin	EU

utasitás 1;
utasitás 2;
:
utasitás n;
end

/A regiszter tartalma 4-el szorzódik végrehajtás után/

cella-értékadás: CH:= regiszter

regiszter értékadás: regiszter:= [abs / neg / neg abs] {konstans / regiszter / CH}

egész regiszter:= string
egész regiszter:= @CH

} EU

Regiszter értékadás jobboldalához aritmetikai operátort és operandust, egész-regiszter értékadás jobboldalához logikai, vagy shift operátort és operandust irva, továbbra is regiszter-értékadást /egész-regiszter értékadást/ kapunk.

aritmetikai op: + - / * ++ -- =:
logikai op: and or xor
shift op: shll shla shrl shra

for-utasitás: for egész reg. értékadás step növekmény until határ do U

if-utasitás: if feltétel then vagy U

if feltétel then EU else U

while-utasitás: while feltétel do U

feltétel az if és while utasitásban:

CC-vizsgálat = ¬= < <= >= > overflow owerflow ¬mixed mixed

reláció: regiszter relációjel { konstans / regiszter / CH } ahol

relációjel: = ¬= < <= >= >

byte-test: byte tipusu CH vagy ¬ byte tipusu CH

összetett feltétel: CC-vizsgálat, reláció, byte test sorozat összekapcsolva and vagy or operátorokkal. /and és or nem vegyithető/

Táblázat CC-vizsgálathoz:

szimbólum	CC
=	0
¬ =	1 vagy 2
<	1
<=	0 vagy 1
>=	0 vagy 2
>	2
overflow	3
¬ overflow	0,1 vagy 2
mixed	1
¬ mixed	0 vagy 3

Program: U . vagy
global procedure

Konstansok irása: negativ előjel : _ pozitiv előjel: nincs
skála-faktor 10-es alapu, elé ' kell.
félszavas egész végére: S
tizedespont: .
valós végére: R /ha .-ból vagy '-ből kiderül, hogy valós a konstans akkor elhagyható/.
duplaszavas valós végére: L
byte-konstans: "char" vagy#␣␣X
hex.konstans: #␣ ␣␣ . . .

5.3. PL360 kódgenerálásról

A PL360 generált kódját assembler jelöléssel adjuk - a fordító közvetlenül a Linkage aditor számára generál kódot.

5.3.1. A goto utasitás B /branch unconditional/ utasitásra fordul.

5.3.2. A függvényhivás egyetlen gépikódu utasitást generál a függvény deklarációja alapján.

5.3.3. A procedure hivás egy vagy három utasitást generál, aszerint, hogy a hivás ugyanabban a szegmensben van-e, mint a procedure deklarációja.

```
BAL   m,p              ha a hivás P procedure szegmensében
                       van
L     15,ujbázis       ha a hivás nem abban a
BAL   m,p              szegmensben van, ahol P deklarálva
L     15, régi bázis   volt
```

5.3.4. A null utasitás nem generál kódot.

5.3.5. A case utasitás a következő kódot generálja:

```
      SLL     m,2
      B       SW(m)
L1 utasitás1 kódja
      B       LX
L2 utasitás2 kódja
      B       LX
        :
Ln      utasitásn kódja
SW    B       LX
      B       L1
      B       L2
        :
      B       Ln
LX
      :
```

Itt m a case-ben szereplő regiszter száma, n az utasitások száma a case-ben

5.3.6. Cella-értékadás. Ennek formája: Cella-hivatkozás:=regiszter.

A generált utasitást a következő táblázat mutatja:

Cella-hiv. tipusa	regiszter tipusa	generált utasitás
egész	egész	ST
félszavas egész	egész	STH
valós	valós	STE
valós	duplaszavas valós	STE
duplaszavas valós	duplaszavas valós	STE

5.3.7. Regiszter-értékadás. Itt táblázatban megadjuk, hogy az értékadás jelére /:=/ és az aritmetikai operátorokra milyen kód fordul.

Konstansokat literálként kezel a forditó.

Regiszter az értékadás baloldalán	A jobboldal tipusa	:=	+	-	*	/	++	--	=:
egész	egész regiszter	LR	AR	SR	MR	DR	ALR	SLR	LR
egész	egész cella	L	A	S	M	D	AL	SL	ST
egész	félszavas egész cella	LH	AH	SH	MH				STH
valós	valós regiszter	LER	AER	SER	MER	DER	AUR	SUR	LER
valós	valós cella	LE	AE	SE	ME	DE	AU	SU	STE
duplaszavas valós	valós regiszter	LER	AER	SER	MER	DER	AUR	SUR	LER
duplaszavas valós	duplaszavas valós regiszter	LDR	ADR	SDR	MDR	DDR	AWR	SWR	LDR
duplaszavas valós	valós cella	LE	AE	SE	ME	DE	AU	SU	STE
duplaszavas valós	duplaszavas valós cella	LD	AD	SD	MD	DD	AW	SW	STD

5.3.8. A for utasitás kódja: egész-regiszter értékadás.

```
        B     L2
L1      utasitás
        A     m,INC
L2      C     m,LIM
        BC    c,L1
```

Az L2-n lévő compare utasitás lehet C, CH, vagy CR a felső határ LIM tipusa szerint. C függ a növekmény /INC/ előjelétől.

5.3.9. Az if utasitás kódja: a kódot a következő esetekre adjuk meg:

if $felt_1$ ... and $felt_{n-i}$ and $felt_n$ then EU else U

```
        felt1
        BC    c1,L1
        :
        feltn-1
        BC    cn-1,L1
        feltn
        BC    cn,L1
        egyszerü utasitás
        B     L2
L1      utasitás
L2
```

A C_i-ket az i. feltétel határozza meg, mely maga is Compare utasitásra, vagy üres utasitásra fordul /ha CC vizsgálat a feltétel/. A compare utasitás tipusa függ az operandusoktól.

if felt, or ... or $felt_{n-1}$ or $felt_n$ then EU else U

```
        felt1
        BC    c1, L1
        .
        feltn-1
        BC    cn-1,L1
```

```
        felt_n
        BC      C_n,L2
L1      egyszerü utasitás
        B       L3
L2      utasitás
L3
```

Az előző esethez hasonló a helyzet.

5.3.10 while utasitás kódja:

```
L1      feltétel
        BC      c,L2
        utasitás
        B       L1
L2
```

5.4. A PL360 forditó hibaüzenetei.

Hiba száma	Üzenet	Jelentés
00	SYNTAX	A program szintaktikusan hibás. Az elemzés a következő utasitással folytatódik.
01	VAR ASS TYPES	Cella-értékadás nem kompatibilis tipusok között.
02	FOR PARAMETER	For utasitásban a regiszter nem egész - a lépésszám nem egész, vagy félszavas egész - a felső határ nem megengedett.
03	REG ASS TYPES	Regiszter-értékadás nem kompatibilis tipusok között.
04	BIN OP TYPES	Aritmetikai, vagy logikai müvelet nem kompatibilis tipusokkal.
05	SHIFT OP	Shift operátor valós értékkel.
06	COMPARE TYPES	Összehasonlitásban nem kompatibilis operandusok.
07	REG TYPER OR#	A regiszter tipusa, vagy száma nem megengedett.

08	UNDEFINED ID	Deklarálatlan azonositó. Ezentul a fordító ugy tekinti, mintha R1 lenne. Ez ujabb hibaforrás lehet.
09	MULT LAB DEF	Ugyanaz az azonositó ugyanabban a blokkban többször cimke.
10	EXC INI VALUE	Kezdeti értékadás tömbnek tul sok elemmel.
11	NOT INDEXABLE	Tilos indexelés.
12	DATA OVERFLOW	A deklarált változó cime az adatszegmensben 4095-nél nagyobb - uj szegmenst kell nyitni.
13	NO OF ARGS	Függvényben nem megfelelő a paraméterek száma.
14	ILLEGAL CHAR	Illegális karakter - átugorja a fordító.
15	MULTIPLE ID	Ugyanabban a blokkban egy azonositó többször van deklarálva. Az ujabb deklarációt nem veszi tekintetbe a fordító.
16	PROGRAM OFLOW	Az aktuális programszegmens tul nagy. Ujra kell szegmentálni.
17	INITIAL OFLOW	A fordító kezdeti értékek beállitására szolgáló területe elfogyott. Adatszegmentációval, vagy az adatok alkalmas átrendezésével lehet segiteni
18	ADDRESS OFLOW	Indexként használt konstans olyan, hogy a kapott relativ cim nem 0 és 4095 közé esik.
19	NUMBER OFLOW	Tul nagy szám.
20	MISSING .	End of file az inputon anélkül, hogy a programot lezáró . megjelent volna.
21	STRING LENGTH	0 vagy 256-nál hosszabb string.
22	AND/OR MIX	Összetett feltétel if és while utasitásban, mely keveri az AND és OR operátort.
23	FUNC DEF NO	A formátumszám függvénydeklarációban illegális.

24	ILLEGAL PARAM	A paraméter függvényhivásban nem egyezik a függvény definiciójában megengedett tipussal.
25	NUMBER	Konstans illegális tipusu, vagy értéke nem megengedett.
26	SYN MIX	Szinonim deklarációban cella és regiszter deklaráció nem vegyithető.
27	SEG NO OFLOW	99 szegmensnél több van.
28	ILLEGAL CLOSE	CLOSE base utasitás - de nincs nyitott szegmens a blokkban.
29	NO DATA SEG	Deklaráció - de nincs nyitott adatszegmens.
30	ILLEGAL INIT	COMON adatszegmensbe nem lehet kezdeti értéket adni.

5.5 Két példa PL360 programra

Az első példa egy PL360-ban irt procedure valós szám kártyáról való beolvasására. /A valós szám szintaxisa a PL360-ban előirt/.

```
procedure Inreal(R4)
begin comment  This procedure reads characters forming a real
    number according to the  PL360 syntax. A procedure nextchar
    (R3) is used to obtain the next character in sequence in
    register R0. The answer appears in the long real register
    F01. Registers  R0...R4 and all real registers are used;
    integer register char syn R0, accum syn R1, scale syn R2, ext syn R3;
    long real register answer  syn  F01;
    byte sign, exposing;
    long real converted = #4E0000000000000000L;
    integer convert syn converted (4);
    function SRDL (9, #8C00), LTR(1, #1200);
    nextchar; RESET(sign);
    while char<"0" do
    begin if char = "-" then SET(sign) else RESET(sign); nextchar;
    end;
    comment  Accumulate the integral part in accum;
    accum := char and #F; nextchar;
    while char >="0" do
    begin char := char and #F; accum * 10S+char; nextchar;
    end;
    scale := 0;
    convert := accum; answer := converted+0L;
    if char = "." then
    begin comment Process fraction. Accumulate number in answer;
       nextchar;
       while char >= "0" do
       begin char := char and #F; convert := char;
          answer ::= answer * 10L+converted; scale := scale -1;
          nextchar;
       end;
    end;
    if char = "'" then
    begin comment Read the scale factor and add it to scale;
       nextchar; if char = "-" then
```

```
        begin SET(exposign); nexcthar;
        end else
        if char = "+" then
        begin RESET(exposign); nexcthar;
        end else RESET(exposign);
    accum := char and F;  nextchar;
    while char >= "0" do
    begin char := char and #F; accum := accum * 10S+char; nextchar
    end;
    if exposign then scale := scale - accum else scale :=
                                          scale + accum;
end;
if scale ¬ = 0 then
begin comment Compute F45 := 10 ↑ scale;
    if scale < 0 then
    begin scale := abs scale; SET(exposign);
    end else RESET(exposign);
    F23 := 10L; F45 := 1L; F67 := F45;
    while scale ¬ = 0 do
    begin SRDL(scale,1);
        comment  divide by 2, shift remainder into scale extension
        F23 := F23*F67; F67 := F23; LTR(ext,ext);
        if < then F45 := F45*F23; comment < if remainder is 1;
    end;
```

A második példában egy PL360 procedure büvösnégyzeteket számól, 3,5 és 9 nagyságuakat. A procedure-t a következő FORTRAN program hivja:

```
C     * * *    FORTRAN DRIVER - CALLS PL360 PROCEDURE * * *
C
      WRITE(6,100)
      DO 10 I = 3,9,2
       IF (I. NE. 7) CALL MAGSQR(I)
 10   CONTINUE
      RETURN
100   FORMAT  (24H1 MAGIC SQUARE GENERATOR)
      END
```

A büvös négyzet konyomtatását is FORTRAN rutin végzi - ezt a PL360 procedure hivja. Az output rutin a következő:

```
C        ***   FORTRAN OUTPUT - CALLED BY PL3600  ***
C     SUBROUTINE OUTPUT
      COMMON /PARAMS/ N, MSQR(256)
      WRITE (6,100) N
C        THESE STATEMENTS MIRROR THE MAPPING USED IN THE PL360
               PROGRAM
      LIMIT = 16*N
      DO 10 I = 16,LIMIT,16
       WRITE (6,101) (MSQR(I+J+1),  j=1,N)
 10   CONTINUE
100   FORMAT ( 5H10 N = , I5 / 1H  )
101   FORMAT( 1H , 15I4 )
      RETURN
      END
```

Végül, a PL360 procedure, mely a büvös négyzetet előállitja:

```
GLOBAL PROCEDURE MAGSQR (R14);
 BEGIN COMMENT
     MAGIC SQUARE GENERATOR - ALGORITHM 118 (CACM, AUG. 1962) ;
    STM(R14,R12,B13(12));
    BEGIN
      EXTERNAL PROCEDURE OUTPUT (R14); NULL
      COMMON DATA PARAMS BASE R12;
       INTEGER DIMENSION;
       ARRAY 256 INTEGER X;
      CLOSE BASE;
      SEGMENT BASE R11;
       ARRAY 18 LOGICAL SAVE;
       SHORT INTEGER NSQR;
       INTEGER REGISTER N SYN RO, I SYN R1, J SYN R2, T SYN R3,
        IJ SYN R4,  K  SYN R5;
      COMMENT  COMPLETE STANDARD LINKAGE;
       R1O :=  SAVE; MEM(R1O=4) := R13; MEM(R13+8) := R1O;
       R13 := R1O;
      COMMENT  RECOVER PARAMETER;
       R1 := MEM(R1); N := MEM(R1); DIMENSION := N;
      COMMENT  ZERO ARRAY;
       T := 0; FOR IJ := O STEP 4 UNTIL 1020 DO x(IJ) := T;
      COMMENT MAIN ALGORITHM BEGINS HERE;
      NSQR := N; R1  := N*NSQR; NSQR := R1;
      I := N+1 SHRL L; J := N;
      FOR K := 1 STEP UNTIL NSQR DO
      BEGIN T := I SHLL 6; IJ := J SHLL 2 + T; T := X(IJ);
         IF T = O THEN
         BEGIN I := I-1; J := J-2;
            IF I < 1 THEN I := I+N;
            IF J < 1 THEN J := J+N;
            T := I SHLL 6; IJ := J SHLL 2 + T;
         END;
         X(IJ)  := K;
```

```
        OUTPUT;
        R13 := MEM(R13+4);
    END;
    LM(R14,R12,B13(12));
END.
```

5.6 A PL360 szintaxisa

```
 1    <K REG>        ::= <ID>
 2    <T CELL ID>    ::= <ID>
 3    <PROC ID>      ::= <ID>
 4    <FUNC ID>      ::= <ID>
 5    <T CELL>       ::= <T CELL ID>
 6    <T CELL>       ::= <T CELL1>   )
 7    <T CELL>       ::= <T CELL2>   )
 8    <T CELL1>      ::= <T CELL2>   <ARITH OP> <T NUMBER>
 9    <T CELL1>      ::= <T CELL3>   <T NUMBER>
10    <T CELL2>      ::= <T CELL3>   <K REG>
11    <T CELL3>      ::= <T CELL ID>   (
12    <UNARY OP>     ::= ABS
13    <UNARY OP>     ::= NEG
14    <UNARY OP>     ::= NEG  ABS
15    <ARITH OP>     ::= +
16    <ARITH OP>     ::= -
17    <ARITH OP>     ::= *
18    <ARITH OP>     ::= /
19    <ARITH OP>     ::= +  +
20    <ARITH OP>     ::= -  -
21    <ARITH OP>     ::= =:
22    <LOG OP>       ::= AND
23    <LOG OP>       ::= OR
24    <LOG OP>       ::= XOR
24A   <SHIFT OP>     ::= SHLA
24B   <SHIFT OP>     ::= SHRA
```

```
24C  <SHIFT OP>    ::= SHLL
24D  <SHIFT OP>    ::= SHRL
25   <K REG ASS>   ::= <K REG>   :=  <T CELL>
26   <K REG ASS>   ::= <K REG>   :=  <T NUMBER>
27   <K REG ASS>   ::= <K REG>   :=  <STRING>
28   <K REG ASS>   ::= <K REG>   :=  <K REG>
29   <K REG ASS>   ::= <K REG>   :=  <UNARY OP>  <T CELL>
30   <K REG ASS>   ::= <K REG>   :=  <UNARY OP>  <T NUMBER>
31   <K REG ASS>   ::= <K REG>   :=  <UNARY OP>  <K REG>
32   <K REG ASS>   ::= <K REG>   :=  @ <T CELL>
33   <K REG ASS>   ::= <K REG ASS>  <ARTTH OP>  <T CELL>
34   <K REG ASS>   ::= <K REG ASS>  <ARITH OP>  <T NUMBER>
35   <K REG ASS>   ::= <K REG ASS>  <ARITH OP>  <K REG>
36   <K REG ASS>   ::= <K REG ASS>  <LOG OP>  <T CELL>
37   <K REG ASS>   ::= <K REG ASS>  <LOG OP>  <T NUMBER>
38   <K REG ASS>   ::= <K REG ASS>  <LOG OP>  <K REG>
39   <K REG ASS>   ::= <K REG ASS>  <SHIFT OP> <T NUMBER>
40   <K REG ASS>   ::= <K REG ASS>  <SHIFT OP> <K REG>
41   <FUNC1>       ::= <FUNC2>  <T NUMBER>
42   <FUNC1>       ::= <FUNC2>  <K REG>
43   <FUNC1>       ::= <FUNC2>  <T CELL>
44   <FUNC1>       ::= <FUNC2>  <STRING>
45   <FUNC2>       ::= <FUNC ID>  (
46   <FUNC2>       ::= <FUNC1>  ,
47   <CASE SEQ>    ::= CASE  <K REG>  OF  BEGIN
48   <CASE SEQ>    ::= <CASE SEQ> <STATEMENT> ;
49   <SIMPLE ST>   ::= <T CELL>  :=  <K REG>
50   <SIMPLE ST>   ::= <K REG ASS>
51   <SIMPLE ST>   ::=  NULL
52   <SIMPLE ST>   ::= <PROC ID>
53   <SIMPLE ST>   ::= <FUNC ID>
54   <SIMPLE ST>   ::= <FUNC1>  )
55   <SIMPLE ST>   ::= <CASE SEQ>  END
56   <SIMPLE ST>   ::= <BLOCKBODY>  END
57   <REL OP>      ::= <
58   <REL OP>      ::= =
59   <REL OP>      ::= >
```

```
60   <REL OP>      ::= <  =
61   <REL OP>      ::= >  =
62   <REL OP>      ::= ¬  =
63   <NOT>         ::= ¬
64   <CONDITION>   ::= <K REG> <REL OP> <T CELL>
65   <CONDITION>   ::= <K REG> <REL OP> <T NUMBER>
66   <CONDITION>   ::= <K REG> <REL OP> <K REG$
67   <CONDITION>   ::= <K REG> <REL OP> <STRING>
68   <CONDITION>   ::=  OVERFLOW
69   <CONDITION>   ::= ¬  OVERFLOW
70   <CONDITION>   ::= MIXED
71   <CONDITION>   ::= ¬  MIXED
72   <CONDITION>   ::= <REL OP>
73   <CONDITION>   ::= <T CELL>
74   <CONDITION>   ::= <NOT> <T CELL>
75   <COMP COND>   ::= <CONDITION>
76   <COMP COND>   ::= <COMP AOR> <CONDITION>
77   <COMP AOR>    ::= <COMP COND> AND
78   <COMP AOR>    ::= <COMP COND> OR
79   <COND THEN>   ::= <COMP COND> THEN
80   <GOTO ST*>    ::= <GOTO ID>
81   <GOTO ST>     ::= <GOTO ST*>
82   <TRUE PART>   ::= <SIMPLE ST> ELSE
83   <TRUE PART<   ::= <GOTO ST*> ELSE
84   <WHILE>       ::= <WHILE>
85   <COND DO>     ::= <COMP COND> DO
86   <ASS STEP>    ::= <K REG ASS> STEP <T NUMBER>
87   <LIMIT>       ::= UNTIL  <K REG>
88   <LIMIT>       ::= UNTIL  <T CELL>
89   <LIMIT>       ::= UNTIL  <T NUMBER>
90   <DO>          ::= DO
91   <STATEMENT->  ::= <SIMPLE ST>
92   <STATEMENT->  ::= IF  <COND THEN>  <STATEMENT->
93   <STATEMENT->  ::= IF  <COND THEN>  <GOTO ST>
94   <STATEMENT->  ::= IF  <COND THEN>  <TRUE PART> <STATEMENT->
95   <STATEMENT->  ::= IF  <COND THEN>  <TRUE PART> <GOTO ST>
96   <STATEMENT->  ::= <WHILE>  <COND DO>  <STATEMENT*>
```

```
 97   <STATEMENT-> ::= FOR  <ASS STEP>  <LIMIT>  <DO>  <STATEMENT>
 98   <STATEMENT*> ::= <STATEMENT->
 99   <STATEMENT*> ::= <GOTO ST>
100   <STATEMENT>  ::= <STATEMENT*>
101   <SI T TYPE>  ::= SHORT  INTEGER
102   <SI T TYPE>  ::= INTEGER
103   <SI T TYPE>  ::= LOGICAL
104   <SI T TYPE>  ::= REAL
105   <SI T TYPE>  ::= LONG REAL
106   <SI T TYPE>  ::= BYTE
107   <SI T TYPE>  ::= CHARACTER
108   <T TYPE>     ::= <SI T TYPE>
109   <T TYPE>     ::= ARRAY  <T NUMBER>  <SI T TYPE>
110   <T TYPE>     ::= ARRAY  DATADEF  <SI T TYPE>
111   <FILL>       ::= <STRING>
112   <FILL>       ::  @  <T CELL>
113   <FILL>       ::= <T NUMBER>
114   <FILL>       ::= <REP LIST3>
115   <REP LIST1>  ::= <T NUMBER>  (
116   <REP LIST1>  ::= (
117   <REP LIST1>  ::= <REP LIST2>
118   <REP LIST2>  ::= <REP LIST1>  <FILL>
119   <REP LSIT3>  ::= <REP LIST2>  )
120   <T DECL1>    ::= <T TYPE>  <ID>
121   <T DECL1>    ::= <T DECL2>  <ID>
122   <T DECL2>    ::= <T DECL4>  ,
123   <T DECL3>    ::= <T DECL1>  =
124   <T DECL4>    ::= <T DECL1>
125   <T DECL4>    ::= <T DECL3> <FILL>
126   <FUNC DC1>   ::= FUNCTION
127   <FUNC DC1>   ::= <FUNC DC7>  ,
128   <FUNC DC2>   ::= <FUNC DC1>  <ID>
129   <FUNC DC3>   ::= <FUNC DC2>  (
130   <FUNC DC4>   ::= <FUNC DC3>  <T NUMBER>
131   <FUNC DC5>   ::= <FUNC DC4>  ,
132   <FUNC DC6>   ::= <FUNC DC5>  <T NUMBER>
133   <FUNC DC7>   ::= <FUNC DC6>  )
```

```
134   <SYN DC1>     ::= <T TYPE>  <ID>  SYN
135   <SYN DC1>     ::= <SI T TYPE>  REGISTER  <ID>   SYN
136   <SYN DC1>     ::= <SYN DC3>  <ID>  SYN
137   <SYN DC2>     ::= <SYN DC1>  <T CELL>
138   <SYN DC2>     ::= <SYN DC1>  <T NUMBER>
139   <SYN DC2>     ::= <SYN DC1>  <K REG>
140   <SYN DC3>     ::= <SYN DC2>  ,
141   <PROC HD1>    ::= <PROCEDURE <ID>
142   <PROC HD2>    ::= <PROC HD1>  (
143   <PROC HD3>    ::= <PROC HD2>  <K REG>
144   <PROC HD4>    ::= <PROC HD3>  )
145   <PROC HD5>    ::= <PROC HD4>  ;
146   <PROC HD6>    ::= <PROC HD5>
147   <PROC HD6>    ::= <GLOBAL <PROC HD5>
148   <PROC HD6>    ::= EXTERNAL <PROC HD5>
149   <PROC HD6>    ::= ENTRY   <PROC HD5>
150   <PROC HD6>    ::= SEGMENT   <PROC HD5>
151   <DSEG TYPE>   ::= GLOBAL DATA  <ID>
152   <DSEG TYPE>   ::= EXTERNAL DATA  <ID>
153   <DSEG TYPE>   ::= COMMON  DATA <ID>
154   <DSEG TYPE>   ::= COMMON
155   <DSEG TYPE>   ::= SEGMENT
156   <DSEG TYPE>   ::= DUMMY
157   <DECL>        ::= <T DECL4>
158   <DECL>        ::= <FUNC DC7>
159   <DECL>        ::= <SYN DC2>
160   <DECL>        ::= <PROC HD6>  <STATEMENT*>
161   <DECL>        ::= <DSEG TYPE>  BASE  <K REG>
162   <DECL>        ::= CLOSE BASE
163   <LABEL DEF>   ::= <ID> :
164   <BLOCKHEAD>   ::= BEGIN
165   <BLOCKHEAD>   ::= <BLOCKHEAD>  <DECL>  ;
166   <BLOCKBODY>   ::= <BLOCKHEAD>
167   <BLOCKBODY>   ::= <BLOCKBODY>  <STATEMENT>  ;
168   <BLOCKBODY>   ::= <BLOCKBODY>  <LABEL DEF>
168A  <PROGRAM->    ::=
```

```
168B  <PROGRAM->    ::= GLOBAL  <PROC HD5>
169   <PROGRAM->    ::= .
170   <PROGRAM->    ::= .  GLOBAL  <PROC HD5>
171   <PROGRAM*>    ::= <PROGRAM->  <STATEMENT*>
172   <PROGRAM>     ::= <PROGRAM*>  .
```

A <STRING> , <ID> és <T-NUMBER> nincs definiálva itt, ezeknek informális definiciója egyszerübb. /2.2.3.,2.2.4.,2.2.5/

PL360 MAKRÓ SYNTAX

```
<MACRO DECL> ::= MACRO  <MACDEF>
<MACRO DECL> ::= <MACRO DECL>  ,  <MACDEF>
<MACDEF>     ::= <MACDEFHEAD>  =  <MACRO STRING>  &
<MACDEFHEAD> ::= <MACRO ID>
<MACDEFHEAD> ::= <MACRO ID>  (  <PARLIST>  )
<MACRO ID>   ::= <ID>
<PARLIST>    ::= <PAR>
<PARLIST>    ::= <PARLIST>  , <PAR>
<PAR>        ::= <ID>
<PAR>        ::= &  <ID>
<MACRO CALL> ::= <MACRO ID>
<MACRO CALL> ::= <MACRO ID>  (  <APARLIST>  )
<APARLIST>   ::= <PARSTRING>
<APARLIST>   ::= <APARLIST>  ,  <PARSTRING>
```

<MACRO STRING> és <PARSTRING> definiciója formálisan nehézkes. Ezeket 5.1.1. és 5.1.2.-ben informálisan definiáltuk.

REFERENCIÁK

[1] N. Wirth, a programming language for the 360 computer. Techn.Rep. CS 33, Stanford U. Stanford, Calif. Dec.1965.

[2] N. Wirth, The PL360 System. Techn. Rep. CS 68, Stanford U. Stanford, Calif. 1967.

[3] N. Wirth, PL360, A programming Langnage for the 360 computers, JACM, vol 15, No.1. Jan. 1968. 37-74.

[4] N. Wirth, Documentation of the computer PL360. Technical Report.

[5] Norwegian Computer Center, SIMULA technical documentation - PL360 modifications.

[6] J. Palme, Benchmark evaluations of the 360/370 Simula compiler FOA P Report C 8385-M3/E5/ Jan. 1974.

[7] IBM 360, Principles of operation IBM kézikönyv, A22-6821 jelzet.

[8] Tomka Erzsébet, Programozás ASSEMBLER nyelven. SZÁMOK, 1973.

/9/ P. Naur, Revised report on the algoritmic language ALGOL 60, CACM 6, jan.1963.

[10] DOS és OS Linkage Editor IBM Systems Reference Library C 24-5036 és C28-6538.