

PC - 4000

1. Javaslát a PC-4000 kisrendűtűge
programozási rendszernek a további fejlesz-
téséről.
2. ASS-4000
Assembler programozási keretrendszer
3. PCL (strukturált makroassembler)
Felhasználói keretrendszer
4. Szimulátor (PDP 11/70)
Felhasználói keretrendszer
- 5.) OS-4000 operációs rendszer
Felhasználói

Dr. Manfred Kriekalle

A PC-4000 KISSZÁMITÓGÉP
PROGRAMOZÁSI RENDSZERÉNEK TOVÁBBFEJLESZTÉSÉRŐL

Javaslat a Practicom
mikroprocesszoros PC
operációs rendszerének és
programozási környezetének
kidolgozására

Kraft Walter
1980

TARTALOM:

1. Általános
 - 1.1. A jelenlegi helyzet
 - 1.2. Továbbfejlesztési javaslat
 - 1.3. Megjegyzések a továbbfejlesztésről, igénybecslés.
2. Az operációs rendszer moduljainak vázlatos leírása
 - 2.1. Keyboard Monitor
 - 2.2. Device handlerok
 - 2.3. File-kezelő rendszer
 - 2.4. User interface
 - 2.5. Utility programok
3. A kereszt rendszer moduljainak vázlatos leírása
 - 3.1. Kereszt-assembler
 - 3.2. Makro-generátor
 - 3.3. Szimulátor

1. Általános

1.1. A jelenlegi helyzet:

A gép felhasználását és a programok fejlesztését az alábbiak nehezítik:

- A programok kezelése nem mágneslemezorientált,
- A programfejlesztéshez szükséges programok viszonylag nagy helyet foglalnak el,
- A forrásprogramok létrehozására és javítására nincs szöveg-szerkesztő program,
- A program tesztelés csak a hardware által biztosított csapdázási lehetőségekkel történhet.

1.2. Továbbfejlesztési javaslat:

A fentiek kiküszöbölésére a gépre az alábbi fejlesztéseket tudjuk implementálni.

A. Egy egyszerű, de hatékony diszkes operációs rendszer létrehozása, az alábbi alkotóelemekből:

- Keyboard Monitor
- Periféria-függetlenséget biztosító device handler-ek,
- File-kezelő rendszer
- User interface a file kezelő rendszerhez ill. a keyboard monitorhoz

- Utility programok:

Editor

Copy

Debugger

B. Kereszt fordítási és futtatási rendszer egy inter-
aktiv módon működő, viszonylag olcsó gépre.

A rendszer elemei:

- Kereszt-assembler
- Makro-előtét
- Szimulátor

A szimulátor egyben debug lehetőséget is biztosít.

1.3. Megjegyzések a továbbfejlesztésről /igénybecslés/

- A kereszt rendszert magasszintű nyelven IBM,
PDP /TPA/ vagy más gépre kell írni.

Igy a PC-4000 programjainak "nagy" gépen való
kidolgozására nyílik lehetőség, az összes "nagy"-
gépes adottság /editor, könyvtárak stb./ felhasz-
nálásával.

Megjegyezzük, hogy viszonylag olcsó time-
sharing rendszerek bérelhetők. /PDP-11, Data-point,
stb/

- A fejlesztések várható ráfordítás-igénye
/a Wolwerton módszerrel becsülve/:

a./ Az operációs rendszer:

3-6 emberév

b./ A kereszt-rendszer

assembler: 3-4 mérnökhónap

makro: 2-3 mérnökhónap

szimulátor: 4-5 mérnökhónap

- Az operációs rendszer viszonylag könnyen átir-
ható kell, hogy legyen az új PC gépre.

2. Az operációs rendszer moduljainak vázlatos leírása:

2.1. Keyboard Monitor:

Végrehajtja a felhasználó konzolon adott parancsait
/pl: RUN, SAVE/.

Könnyen bootstrap-elhető ill. reinicializálható
program.

2.2. Device handlers:

Szubrutinok a perifériák I/O-jának egységes keze-
lésére. Standard hívási eljárással aktivizálhatók,
"maszkolják" a perifériák eltérő tulajdonságait a felhasz-
nálás felé. Így periféria-független I/O érhető el.

2.3. File-kezelő rendszer

UNIX típusu, szó-orientált file-okat kezel a mág-
neslemezen.

A file-ok mozgatására az alábbi funkciókkal nyílik lehetőség

open	- file nyitása
create	- file létrehozása
read	- egy szóhalmaz olvasása
write	- egy szóhalmaz kiírása
seek	- pozicionálás
close	- file zárása
delete	- file törlése

A file-kezelő rendszer a Keyboard Monitor alatt működik.

2.4. User interface

A Keyboard Monitor funkciót biztosítja a felhasználó programjainak számára. Tehát pl. segítségével mód van programok láncolására, file kezelésre programból.

2.5. Utility programok

2.5.1. Editor:

Szimbolikus szövegszerkesztő program. Forrás és adatfile-ok létrehozására és módosítására szolgál.

2.5.2. Copy:

File-másolást végez.

2.5.3. Debugger:

Interaktív hibakeresési segédlet.

3. A kereszt rendszer moduljainak vázlatos leírása

A továbbiakban feltesszük, hogy a kereszt-rendszer PDP/11-re került megírásra.

3.1. Kereszt-assembler

PC-4000 assembler forrásprogramok fordítását végzi a PDP-11-en. A fordításról lista készül, a tárgyprogram megőrződik.

3.2. Makro-előtét

Az assemblerrel összeépíthető makró értelmező fázis. A forrásprogramban levő paraméterezett makrohívásokat kifejti. Standard makrokönyvtár használata lehetséges. Mindkét program /a forrás ill. a kifejtett/ PDP diszkes file.

3.3. Szimulátor

A PC-4000 gépi kódú programok végrehajtását biztosítja a PDP-11-en, perifériafüggetlen handler szintig.

Lehetővé teszi a PC programok logikai tesztelését a PDP-n.

A S S - 4 0 0 0

Assembler programozási kézikönyv

Bánkfalvi Zoltán, elnöke

{ Abaffy József
Békési Gábor
Krafft Walter } MKKE
1980

Tartalomjegyzék

1. Bevezetés
2. Karakterkészlet
3. Utasítások
 - 3.1. Utasításformátum
 - 3.2. Szimbolikus gépi utasítások
 - 3.3. Direktívák
4. Cimkék
5. Operandusok
 - 5.1. Direkt operandusok
 - 5.2. Cimkifejezések
 - 5.3. A direktíva-operandusok

1. Bevezetés

A PC-4000 VILATI gyártmányu kisszámitógép egyszerű utasításkészletével, gazdag perifériaválasztékával az adatfeldolgozás, ill. a műszaki-matematikai alkalmazások hasznos bázisgépe lehet.

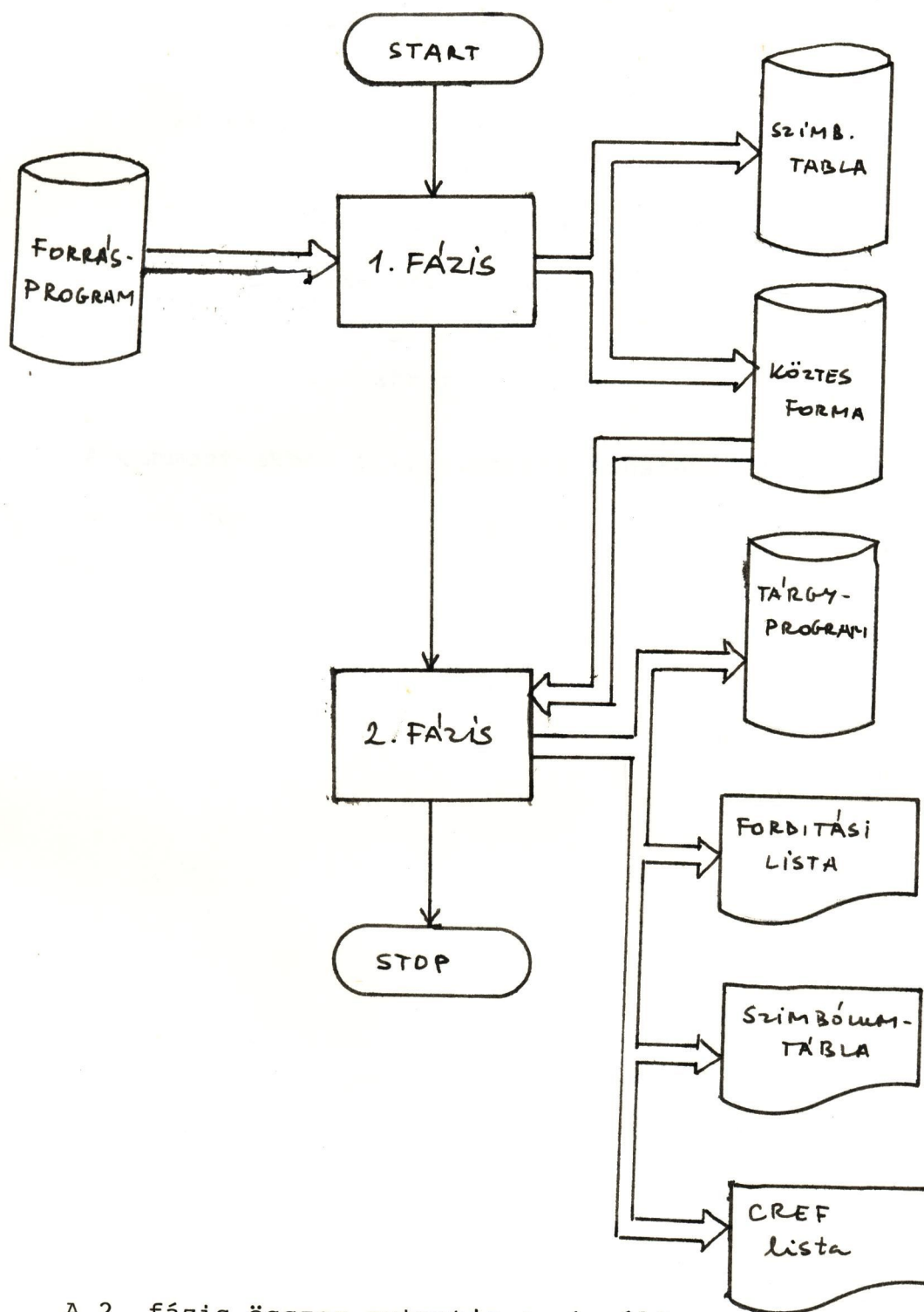
Az ASS-4000 a PC-4000 kisgép assembler szintű nyelve. A régebbi PC assembler egyedi megoldásaival szemben az ASS-4000 a klasszikus assembler elveket képviseli, így megtanulásához számos, kitűnő anyag (Pl: [1]) áll rendelkezésre.

Az ASS-4000 fordítóprogram első verzióban kereszt-assemblerként, a PDP-11/70 kisszámitógépen működik. A kereszt assembler egy szimulátorral kiegészítve komplett tesztrendszerrel biztosít.

Az ASS-4000 kétfázisú assembler. Az első fázis beolvassa az input file-t és létrehozza a szimbólumtáblát. A második fázis az első fázisban létrehozott szimbólumtábla segítségével előállítja a tárgyprogramot ill. fordítási listát készít.

A tárgyprogram abszolút címes; lehetőség van azonban áthelyezhető formátumú tárgyprogramok előállítására is.
/ld. 7. fejezet/

Az assembler működését az 1. ábra szemlélteti.



A 2. fázis összes outputja opcionális.

2. Karakterkészlet

Az AS-4000 a következő karaktereket fogadja el:

- numerikus karakterek (0-9)
- - alfabetikus karakterek (A-Z),
- speciális jelek: +, -, *, vessző, &, = ,
aposztróf

A comment-ekben ill. a string konstansokban előfordulhatnak továbbá a !, ., #, %, (,), /, <, >, ? ,
; , : karakterek is.

3. Utasítások

3.1. Utasításformátum

Az ASS-4000 utasításainak formátuma:

cimke	utasítás	operandus	comment
-------	----------	-----------	---------

A cimke és a comment opcionális, több utasításnál az operandus is az, /ill. néhol nincs rá szükség/.

Az utasítás egyes részeit blank karakterek vagy a tabulátor karakter választja el. Az áttekinthető formátum miatt érdemes a

cimkét az 1. pozícióban,
utasítást a 8. pozícióban,
operandust a 16. pozícióban kezdeni.

/Ez a PDP-n egyszerű tabulálással könnyen betartható/.

A comment és az operandus között legalább egy elválasztó blank karakter szükséges.

A *gal kezdődő sorokat a gép teljes soros comment-nek értelmezi.

Az utasításokat két csoportra oszthatjuk:

- szimbolikus gépi utasítások,
- direktívák.

A direktívák a fordítási folyamathoz szükséges információkat biztosítják, a tárgyprogramban konkrét megfelelőjük általában nincs. A szimbolikus gépi utasítások a PC 4000 gép utasításkészletének megfelelői.

3.2. Gépi utasítások

A szimbolikus gépi utasításokat az 1. táblázat tartalmazza.

Az utasítás átlépéseknél lényeges tudni, hogy ha az utasítás a szó első utasítása, akkor az átlépés a második utasításra vonatkozik, ha a szó második utasítása, akkor az átlépés a teljes következő szóra értendő, és a skip előtti utasítás még egyszer végrehajtódik.

Hasonló komplikáció lép fel a RAD utasításnál, mely egy alprogramból való visszatérés helyét hivatott kijelölni. Ha a RAD a szó első utasítása, akkor a visszatérés a RAD utáni második utasításra történik, ha második utasítás, akkor egy teljes szó kimarad, tehát a visszatérés a RAD utáni harmadik utasításra történik.

Ezeket a zavaró kétértelműségeket az assembler automatikusan kiküszöböli úgy, hogy a skip-eket mindig egy szó első részében, a RAD-ot pedig a szó második felében helyezi el.

Igy az alprogramhívás standard módja egy paraméter esetén a következő:

```

RAD      alpr.
LDA      paraméter
←      JMP      alpr. + 1
→      ...
visszatérés
```

Több paraméter esetén a standard hívási mód:

```

RAD      alpr.
NOP
←      JMP      alpr. + 1
→
visszatérés
```

Az alprogramok első utasításai mindig a következők:

```

ALPR      NOP
NOP      a visszatérési ugrás
1. végrehajtandó alpr. utasítás
```

Az a tény, hogy egy szóban két utasítás helyezkedik el, problémát okoz az ugrásoknál is, hiszen ugrási lehetőségünk csak szóra történő ugrást enged meg és nem utasításra történő ugrást. A nehézségek elkerülése végett az assembler minden címkézett gépi utasítást szókezdetre igazít.

```

[ NOP ]
CIMKE → utasítás
```


ØK	SHL	n	shift left	léptetés n bittel balra	$n \leq 32$
ØM	SHR	n	shift right	léptetés n bittel jobbra	$n \leq 32$
ØO	SHC	n	shift cycle	léptetés jobbra n bittel ciklikusan	$n \leq 32$
ØØØØ	NOP		no operation	üres utasítás	
1K	INCR	n	increment	az AC növelése n-nel	$n < 64$
1M	DECR	n	decrement	az AC csökkentése n-nel	$n < 64$
8	LAN	addr	load address numeric	a cím átkonvertálása numerikus értékbe	$addr < 4096$
9ØØ2	NEG		negatura	$\langle AC \rangle = - \langle AC \rangle$	
9ØØØ	NOT		logikai NOT	$\langle AC \rangle = \neg \langle AC \rangle$	
2JØØ	IFZ(SNZ)		(no skip) if zero	utasítás átlépés, ha $AC \neq \emptyset$	
2NØØ	IFN(SKP)		(no skip) if negative	utasítás átlépés, ha $AC > \emptyset$	
2PØØ	IFP(SKN)		(noskip) if positive	utasítás átlépés, ha $AC < \emptyset$	
2OØØ	IFO(SNO)		(noskip) if overflow	- " - , ha $\neg$ overflow	
24ØØ	IFE(SNE)		(noskip) if end	- " - , $\neg$ végzőrójel	
2KØØ	IFK(SNK)		(noskip) if K set	- " - , ha K nem bekapcsolt	

2L00	IFL(SNL)		(noskip) if L set	utasítás átlépés, ha L nem bekapcsolt	
2M00	IFM(SNM)		(noskip) if M set	- " - , ha M nem bekapcsolt	
5	JUMP	addr	jump	ugrás a megadott címre	addr < 4096
4	RAD	addr	return address	alprogramból visszatérő utasítás beállítása	
6	LDA	addr	load acc. from addr	<addr> → <AC>	
7	STA	addr	store acc. to addr	<AC> → <addr>	
J	AND	addr	logical 'and'	az előző acc. tartalmának és az n rekesz tartalmának logikai szorzata kerül az acc-ba.	
K	ADD	addr	addition	<AC>=<AC>+<addr>	
L	SUB	addr	subtraction	<AC>=<AC>-<addr>	
M	MULT	addr	multiplication	<AC>=<AC>*<addr>	
N	DIV	addr	division	<AC>=<AC>/<addr>	
O	MAM	addr	memory ⇌ ac	<AC> ⇌ <addr>	
35	MEM	m,n	memory	memory page váltás	
3I	GETC		get character	1 karakter bevitele AC-ba	

32	GETD		get digit	1 numerikus karakter bevitele AC-ba
30	PUTC		put character	1 karakter kivitele AC-ból
3P	PUTD		put digit	1 numerikus karakter kivitele AC-ból
31	DEV		device	1 karakter kivitele az AC-ból /a karakter periféria vezérlő kód/
22	IFB(CCI)	m	(noskip) if byte	ugrás ha az acc utolsó byte-ja egyenlő az operandussal
23	IFS	m	(noskip) if secon tetr.	ugrás ha $m = \langle AC \rangle_{8-5}$
26Ø	IFF	m	(noskip) if first tetr.	ugrás ha $m = \langle AC \rangle_{4-1}$
Ø2	ROT	m,n	rotate	ciklikus léptetés 96 biten, jobbra
Ø3	CRA	m,n	cycle register → AC	reg.cikl.léptetése jobbra és bevitel AC-ba
Ø4	CAR	m,n	cycle AC → reg.	AC cikl. léptetése jobbra és bevitel a reg.be.

1. Táblázat

3.2. Direktívák

A direktívákkal a programozás a fordítási folyamatot irányítja ill. a fordításhoz szükséges információkat közli a fordítóprogrammal.

A direktívákat a 2. táblázat foglalja össze.
Az egyes direktívák jelentései:

DS /define storage/

Az operandusban megadott számú szó foglalása.
Az operandus egy decimális vagy hexadecimális konstans.

DC /define constant/

Az operandusban megadott konstans elhelyezése.

Konstans lehet:

- decimális
- hexadecimális /& jellel kezdve!/
/
- string /max. 4 kar., aposztrófok között/

ADDR /address constant/

Az operandusban megadott címkifejezés címének hozzárendelése a rekeszhez.

PROG /Program starting address/

A program abszolút kezdőcímének megadása. Az operandus decimális vagy hexadecimális konstans.

END /A forrásprogram vége./

A forrásprogram végét jelzi, és egyben a fordítás befejezését is okozza.

OWN /Saját utasítás - szó illesztés/

A fordítás ettől a ponttól kezdve az NOWN utasításig a skip és RAD utasítások automatikus illesztését felfüggeszti.

NOWN /No - own/

A fordítóprogram újra automatikusan illeszti a skip és RAD utasításokat.

NEW /new word/

A köv. utasítás új szóba kerül.

PC /lapváltás a fordítási listában/

A fordítási listában új lap kezdődik. A sorszámozás folyamatos.

Használata a logikailag elkülönülő programrészek listába való szétválasztására célszerű.

PAGE n /Page definiálás/

A PC-4000 kisgép memóriája 4096 szavas page-ekből épül fel. Direkt címzés egy page-on belül lehetséges

csak. Page-en kívüli adathivatkozásokat illetve a más page-en folytatódó programra való rátérést a MEM gépi utasítás révén tudjuk realizálni.

Az ASS-4000 a felhasználói programot automatikusan az 1. page-re kezdi címezni. Ha a program vagy az adatok átnyulnak másik page-re, a PAGE direktívával kell ezt jelezni a fordítóprogram számára. A memóriakiosztás a továbbiakban az új lap elején /vagy az új PROG direktívában definiált címnél/ folytatódik.

Az assembler tehát címzési rendszerében a page-váltást figyeli. A futás közbeni page áttéréseket azonban nem szervezi meg automatikusan, vagyis nem illeszt automatikusan MEM utasításokat a megfelelő helyekre, ez a tevékenység a programozót terheli.

A direktiva-táblázat:

DS	$\left\{ \begin{array}{l} \langle \text{dec} \rangle \\ \&\langle \text{hexa} \rangle \end{array} \right\}$	térfoglalás
DC	$\left\{ \begin{array}{l} \langle \text{dec} \rangle \\ \&\langle \text{hexa} \rangle \\ '\langle \text{string} \rangle' \end{array} \right\}$	konstans definíciója
ADDR	$\langle \text{cimkifejezés} \rangle$	cimkonstans definíciója
PROG	$\left\{ \begin{array}{l} \langle \text{dec} \rangle \\ \&\langle \text{hexa} \rangle \end{array} \right\}$	program kezdőcim kijelölése
END		forrásprogram vége
OWN		saját szóillesztés
NOWN		automatikus szóhatárfigyelés
NEW		szókezdetre illesztés
PC		lapváltás a fordítási listában
PAGE	$\langle \text{dec} \rangle$	memória page kijelölése

2. Táblázat

4. Cimkék.

Az assembler nyelvben a felhasználó általában szimbolikus operandusokkal /adatokkal/ dolgozik ill. a programelágazásoknál, utasításmódosításoknál egy címke segítségével hivatkozik utasításokra.

Az adatszimbólumokat a programban a DS, DC ill. ADDR direktívákkal definiálni kell, tehát minden szimbólumnak meg kell jelemie valahol címkeként.

A címke egy max. 6 karakteres, alfabetikus karakterrel kezdődő és csak alfanumerikus karaktereket tartalmazó string.

Adat esetén az azt tartalmazó gépi szó címét jelöli, utasításcímke esetén annak a szónak a címét jelenti, melyben a megjelölt utasítás található. A szókezdetre igazítást az assembler a fordítás során automatikusan végzi.

5. Operandusok

A gépi utasítások operandusai lehetnek:

- direkt operandusok:

pl.: SHL 6

DECR 5

- címkefejezések.

A címkefejezések szimbolikus címkékből, decimális v. hexadecimális konstansokból és literálokból épülnek fel, a + és - műveleti jelek révén.

5.1. Direkt operandusok

Megadhatjuk őket decimális vagy hexadecimális formában. A hexadecimális konstansokat minden esetben a & jellel kell kezdenünk.

Pl.: SHL 15

SHL &0

jelentése ugyanaz.

A MEM, ROT, CRA, CAR utasítások két direkt operandussal, az SHL, SHR, SHC, INCR, DECR műveletek egy direkt operandussal rendelkeznek.

Más utasításoknál direkt operandus nincs.

5.2. Cimkifejezések

Kiértékelésük után egy gépi szó címét adják. Cimkékből, konstansokból, literalokból és műveleti jelekből épülnek fel.

Pl.: X+3

Az eredmény címet mod(4096) kapjuk.

Cimkék:

Adatszimbólumok vagy utasításcimkék. 6 karakteres string-ek, alfabetikus karakterrel kezdődnek, a többi kar. lehet numerikus vagy alfabetikus.

Konstansok:

decimális konstansok vagy hexadecimálisak. A hexadecimálisakat & jellel kell kezdeni.

Literalok

Mindig = jellel kezdődnek. Literalként megadhatunk decimális számot, hexadecimális számot, max.4 karakteres stringet, és teljes utasítást.

Pl.:	LDA	=314	dec. literal
	LDA	= &40K3	hexa literal
	LDA	= 'STRI'	string literal
	LDA	=(LDA X)	utasítás literal

A hexadecimális literal ismertetőjele a & jel, a string-et két aposztróf jel közé kell zárni, az utasítás-literal zárójelbe kerül.

Az utasítás-literalokat akárhányszoros mélységben lehet skatulyázni.

Műveleti jelek:

A + és - műveletek lehetségesek.

5.3. Direktívák operandusai

Operandussal rendelkezhetnek a DS, DC, ADDR, PROG, és PAGE direktívák, a direktívatáblázatban megadott módon.

A DC direktívában szerepelhet string-konstans is.

String-konstans:

Aposztrófok közé zárt, max. 4 karakteres string.

Irodalomjegyzék

- [1]: DONOVAN: Rendszerprogramozás
Műszaki Kiadó, 1978

P C L

Felhasználói Kézikönyv

Marx Károly Közgazdaságtudományi
Egyetem
Matematikai és Számítástudományi
Intézet

1981.

Tartalom

Bevezetés

1. PCL alapismeretek
2. Makrók
 - 2.1. Alapfogalmak
 - 2.2. A makrogenerátor alapfunkciói
 - 2.3. Standard könyvtár
 - 2.4. Beágyazott makrohívások
3. A strukturált kontrollszerkezetek
 - 3.1. A strukturált programozás
 - 3.2. PCL strukturált utasítások
 - 3.3. Feltételek
4. Blokkok
5. Strukturált lista
6. Egyéb assembler fejlesztések
 - 6.1. Szimbólumok
 - 6.2. Commentek
 - 6.3. Az INCLUDE utasítás
7. A PCL fordító futtatása
8. Példa
9. Hibaüzenetek

Bevezetés

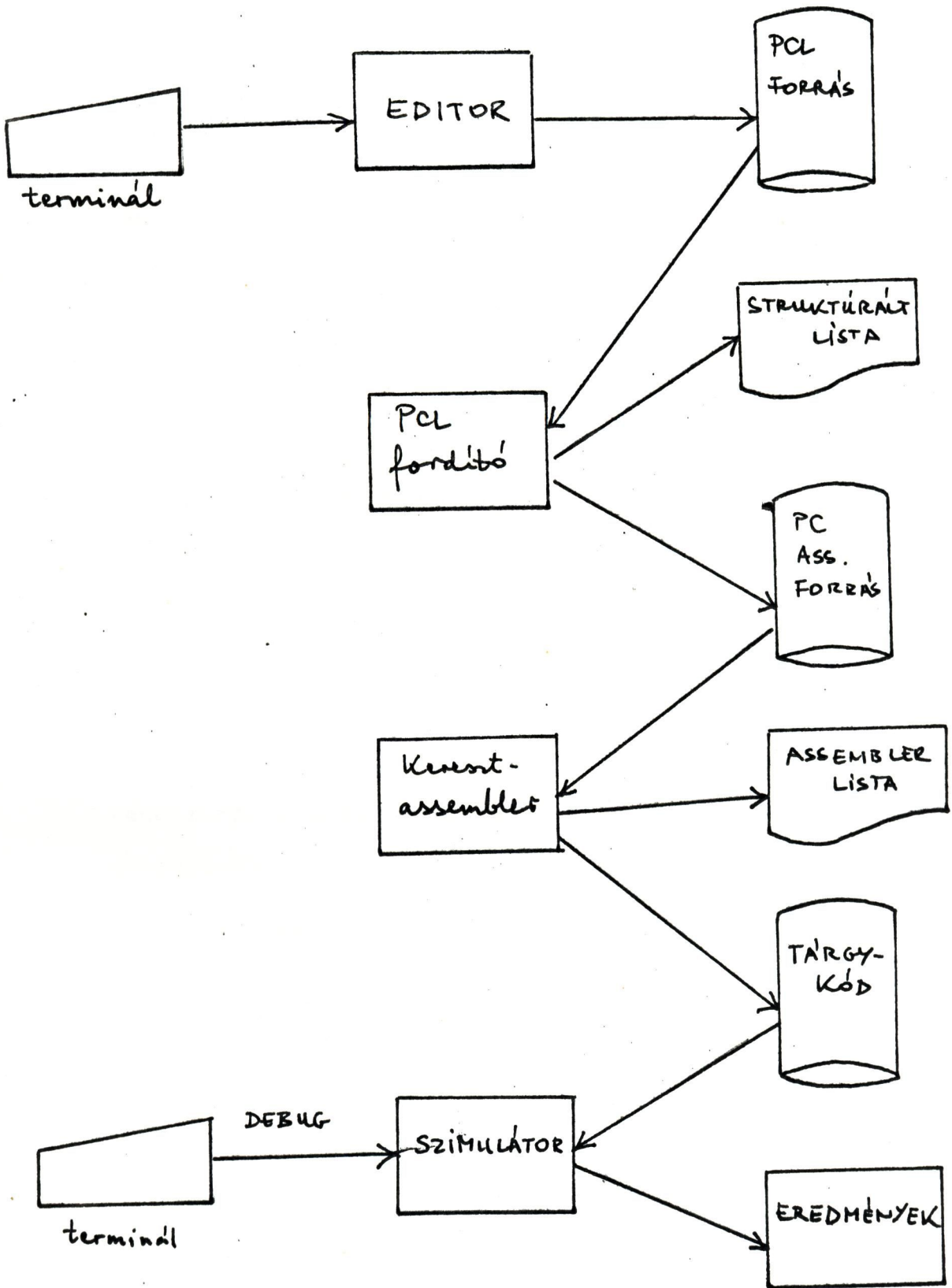
A PCL /PractiComp Low-Level Language/ egy olyan alacsony szintű, assembler alapú programozási nyelv, mely a PC-4000 kisházi számítógép alkalmazói és rendszer programozását könnyíti, amennyiben:

- Makrózási lehetőséget biztosít;
- Egységesíti a programok vezérlését a strukturált kontrollszerkezetek bevezetésével;
- Több formai könnyítést hoz az assemblerhez képest;
- Grafikus dokumentációt készít.

A PCL első verziója a PDP 11/70 számítógépen működő kereszt programfejlesztési rendszer keretében jött létre. Tervbe vettük egy, a PCL lényeges elemeit /makrózás és a strukturált blokk szerkezet/ tartalmazó verzió honosítását is.

A PDP 11/70 keresztrendszer a PCL fordítóból, egy kereszt-assemblerből és egy szimulátorból áll. Lehetővé teszi a PCL-ben /vagy assemblerben/ írt programok fordítását, tesztelését ill. futtatását a PDP 11/70 gépeken.

A keresztrendszeren történő munkafolyamatot az alábbi ábra szemlélteti:



Az Editor egy PDP rendszerprogram, forrásprogramokat tartalmazó file-ok létrehozására, javítására kiválóan alkalmas. Segítségével display-n vagy teletype-on keresztül folytatott párbeszédben felírjuk programunkat a diszkre.

A PCL fordító ebből egy assembler programot generál.

A keresztassembler tulajdonképpen a PC-4000 eredeti assemblerének egy kissé módosított változata, amelyet a szimulátor révén mozgatunk. Létrehozza a tárgykódot.

A szimulátor ezt a tárgykódot végrehajtja, s közben számos lehetőséget nyújt trace, breakpoint, dump és más eszközökkel támogatott hibakeresésre. A szimulátor a perifériákat fizikai szinten kezeli, így periféria kezelő rendszerprogramok is fejleszthetők a keresztrendszer segítségével.

A keresztrendszer gyors, kényelmes. A PCL használatából eredő előnyök és a teljes keresztrendszer által nyújtott kedvező feltételek a PC-4000 programozását minőségileg magasabb fokra emelik.

1. PCL alapismeretek

A PCL nyelvben assembler szintű programokat írunk,
de

- használhatunk makroutasításokat,
- strukturáltan programozhatunk,
- automatikus dokumentációs segédlettel rendelkezünk,
- használhatunk bizonyos továbbfejlesztéseket az assembler nyelvben.

A PCL fordítóprogramot úgy képzelhetjük el, mint egy blokk-transzlator és egy egyszerű makrogenerátor keverékét, kibővítve egy kirajzolást ill. előkészítést végző előfázissal.

A PCL szintaxis rendkívül egyszerű.

- A PCL kulcsszavak /makrohívások/ a \$ jellel kezdődnek.
- PCL utasítás és normál assembler sor nem kerülhet egy sorba.

A további fejezetekben részletesen ismertetjük a PCL programozás tudnivalóit ill. a fordító kezelését.

Meg kell említenünk, hogy a PCL egy keretrendszer. Tényleges haszna nagymértékben függ attól, hogy a felhasználó hogyan él a PCL-ben biztosított lehetőségekkel, pl.

hogy milyen makrókat használ, mennyire teszi magáévá a strukturált programozási stílust.

Előnyei az alábbi összetevőkből adódnak:

- A kódolás magasabb szintre hozható /makrók!/.
/pl. indexelt vagy indirekt címzés/.
- Programozási diszciplinák /pl. strukturált programozás/ vezethetők be, standard-ek írhatók elő.
- A gépen hardware-ben nem létező tulajdonságok programozhatók elemi utasítás szinten.
/pl. indexelt vagy indirekt címzés/.
- A programok dokumentációs értéke nagymértékben javul.

2. Makrók

2.1. Alapfogalmak

A gépközelségű programozás könnyítésére a szubrutin-technika mellett igen gyakran használják a makrókon alapuló megoldást. Lényege, hogy a program különböző helyeire előre definiált kódsorozatokot, un. makrókat illesztenek be. A beillesztést végző rendszerprogram a makro-generátor, a beillesztett kódsorozatok a makrók. A beillesztések helyeit makro-hívási helyeknek nevezzük; a kódsorozatok definíciói az un. makro-definíciók lehetnek a felhasználói programban ill. használhatunk előre definiált makrókat is.

A makrók révén a gépközelségű programozás komfortja nagymértékben javul. A makrókat használó assembler programok /a macro-assembler programok/ rövidebbek és sokkal jobban olvashatóbbak a normál assembler nyelvű programoknál.

Tekintettel arra, hogy a felhasználó tetszés szerint definiálhat makrókat, a macro-generátor által nyújtott komfort nagymértékben függ a felhasználtótól.

A leggyakrabban használt makrókat standard makrónak érdemes definiálni, s akkor a továbbiakban definíció nélkül hívhatjuk őket.

- 2.2 -

A PCL makro-generátor a PC-4000 assemblerre számára biztosítja a makrózási lehetőséget. Az előfázisként működő makrogenerátor a makro-assembler nyelvű programból közönséges assembler programot állít elő.

A PCL révén egységesen kezelhetők:

- a dupla pontosságu aritmetika,
- indexelt és indirekt címzés,
- I/O szekvenciák,
- strukturált kontrollszerkezetek.

2.2. A makrogenerátor alapfunkciói

Makrók definíciója

Makrókat az alábbi formában definiálhatunk:

`<makrodefiníció> ::= '<$'<makronév>'='<makrotörzs>'>'`

ahol a makronév egy min. 2 és max 10 karakteres alfabetikus string, a makrotörzs pedig PCL utasítások sorozata.

Például:

```
< $input =  
  fread  
  load 2f, 0  
>
```

A definíciónak nyilvánvalóan az első hívás előtt kell állnia.

Paraméterek

A hívások helyén beépülő kódsorozat többnyire az egyes hívási helyeken minimális eltérést mutat. Ezeket a változó részeket a paraméterek révén kezeljük. A makrodefinícióban megadott formális paraméterek a hívásnál az aktuális paraméterekkel cserélődnek ki.

A formális paramétereket a \$ jellel és egy utána következő számjeggyel jelöljük. Ebből következően max. 9 formális paraméter lehet.

Például:

```
<$move =  
    load $1, store $2  
>
```

egy két formális paraméterrel rendelkező makró, mely híváskor az éppen akkor megadott két rekesz közt végzi a move műveletet.

A formális paramétereket a makrodefinícióban nem kötelező megadni, decommentként elhelyezhetjük a makronév és a definíciót indító '=' jel között.

Például:

```
<$sum($1,$2,$3)=  
    ...  
>
```


Makro hívás

Formája:

`<makrohívás>::=<makronév>[(paraméter[{,paraméter}])]`

Például:

`$move(a,b)`

`$move(sum, a5)`

A makrohívásokat \$ jellel kell kezdenünk.

A kezdő zárójel közvetlenül a név után áll.

A példában a helyettesítő kódsorozat:

`load a, store b`

`load sum, store a5`

Ha az akt. paraméterek száma <, mint a formálisoké,
akkor a kifejtésben a nem def. paraméterek helyére '?'
helyettesítődik.

Generált címkék

Mivel a makrotörzs minden kifejtéskor helyettesítődik, problémát jelentenek a makrókban előforduló címkék.

Például: egy n integer számot összegző számsorozatot
összegző makró az alábbi lehet:

/A paraméterek

\$1: a számsorozat kezdőcíme

\$2: n

\$3: az összeg/.

```
<$sum =  
    num 0, store $3          S=0  
    load $2, store index     n letárolása  
ciklus:  load $3, add $1      S=S+a(i)  
        store $3, 0  
        load ciklus, count+1  i=i+1  
        store ciklus, load index  
        count -1, store index  
        if p, to ciklus  
        0 , to +2:  
index:   0 , 0  
        resct 30_31  
        ....
```

A többször definiált címkek elkerülésére az alábbi konvenciót tesszük:

Ha egy makrodefinícióban nem paraméterként átadott címkeket használunk, úgy annak lokálisnak kell lennie. Lokálitást úgy biztosítunk, hogy a makrotörzs végén a címkeket a

```
resct <cimke>_<cimke>.....
```

assembler direktívával feloldjuk!

A resct direktívák illesztése a programozó feladata.
A resct leírását bővebben ld. a 6.1 pontban!

2.3. Standard könyvtár

Gyakran használt makrókat nem érdemes minden programban újra meg újra megadni. Felvihetjük őket egy szekvenciális file-ba, az un. standard könyvtárba, melyből a makrogenerátor a feldolgozás kezdetekor tárba tölti őket.

A standard könyvtárba vitel a közönséges editorral történik.

A PCL fordítóprogram rákérdez a makrokönyvtár nevére. A felhasználó megadhatja a PCL-lel együtt kifejlesztett makrokönyvtár nevét, /az un. PCL makrokönyvtárt/, vagy egy saját makrokönyvtárt, amibe nyilvánvalóan induláskor a PCL makrokönyvtárt át kell másolnia.

2.4. Beágyazott makrohívások

A makrogenerátor tudása nagymértékben növekszik azzal, hogy megengedjük a makrohívásokat makrodefiníción belül. Az igény erre nyilvánvaló: szubrutinokban is igen gyakran hívunk más szubrutint, a makróknál is biztosítani kell ezt a lehetőséget.

Például:

```
<$get f =  
  load f2, store $1  
>  
  
<$input =  
  fread  
  $get f($1)  
>
```

definíciók után az alábbi hívás:

```
$input(S)
```

kifejtése:

```
fread  
load f2, store S
```

3. A strukturált kontrollszerkezetek

3.1. A strukturált programozás

A strukturált programozás a programok logikai szerkezetének előírásszerű megszervezését jelenti. Az előírások igen egyszerűek:

- Vezérlésátadások csupán ún. "strukturált vezérlésátadásokkal" történhetnek. Ezek között a leg-
alapvetőbb 3 utasítás az alábbi:

a./ while <feltétel> do

```
    [ciklustörzs  
endwhile
```

b./ repeat

```
    [ciklustörzs  
until <feltétel>
```

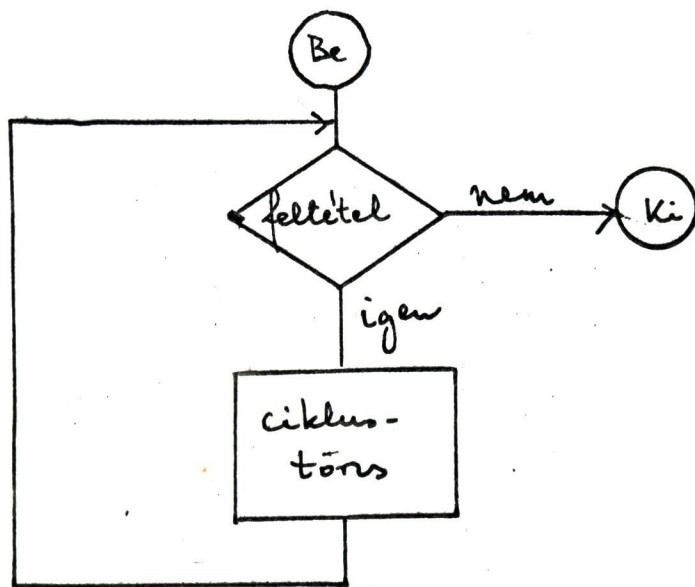
c./ if <feltétel> then

```
    [true ág  
else  
    [false ág  
endif
```

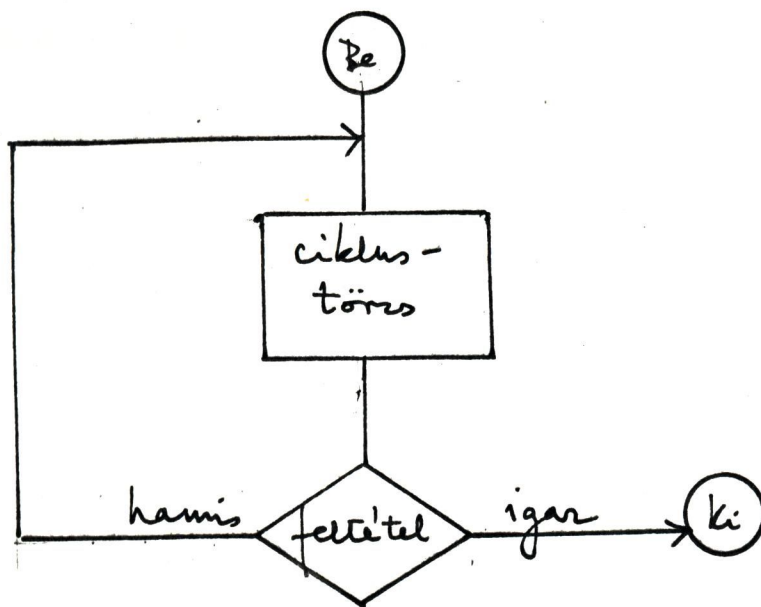
(az else rész elmaradhat).

Az utasítások jelentését blokkdiagramokkal szemléltetjük:

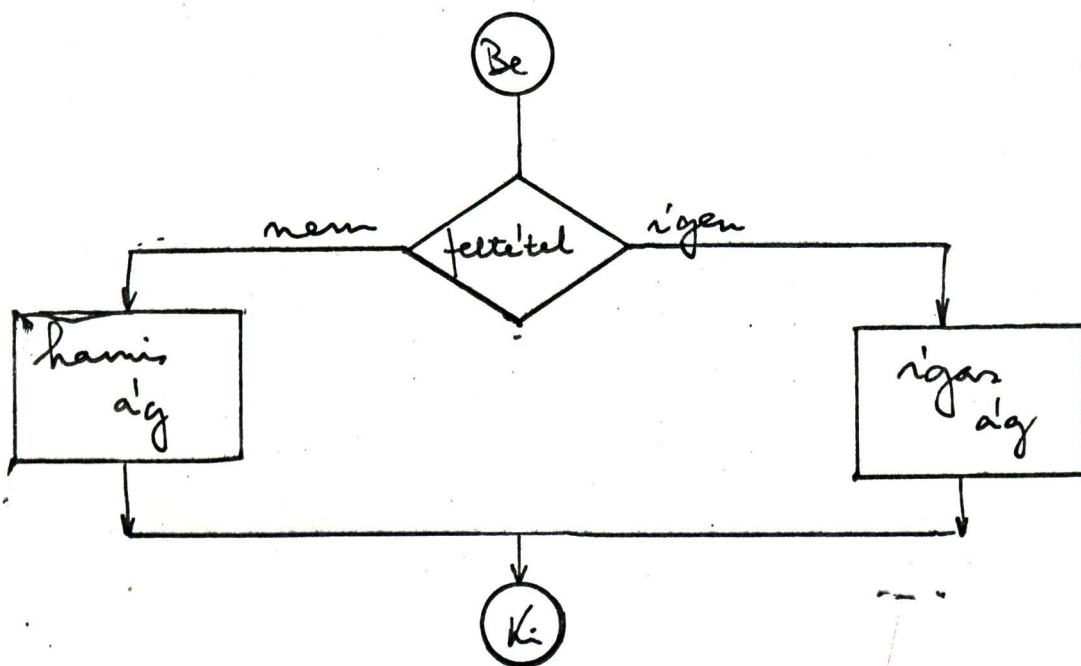
a./ while - endwhile



b./ repeat - until



c./ if - then - else



- A strukturált programozás további megkötése a programok ún. "blokkyszerű" szervezése. Egy blokkhoz egyetlen bemenet és egyetlen kimenet lehetséges csupán.

A strukturált program előnye a "hagyományoshoz" képest, hogy az ágas-bogas programszerkezetből egy skatulyázott szekvenciális szerkezet lesz, ami előnyös a programtervezésnél a bevált top-down módszer követésekor, hasznos a tesztelésben és a dokumentálásban.

3.2. PCL strukturált utasítások

A PCL-ben az IF-THEN-ELSE-WHILE-ENDWHILE és a REPEAT-UNTIL utasítások egyaránt használhatók.

A makrók értelmezéséhez a fordítóprogramnak szüksége van a PCL makrokönyvtár makróira.

A strukturált utasítások minden egyes kulcsszavát a \$ jellel kell kezdeni.

P1:

```
$if (=z) $then
    Load i, store k
$else
    Load k, store i
$endif
$while (#k) $do
    Load i, store j
    Load k, store i
    Load j, store k
$endwhile
$repeat
    Load a, add b
    store a, Load i
    cotl, store i
$until (i,lt,n)
```

A \$if és \$then kulcsszámnak egy sorban kell állnia, hasonlóan a \$while és \$do kulcsszavaknak is. A strukturált kulcsszavak (\$then és \$do kivételével) új sorban kezdődnek.

Példa

Írjunk programszekvenciát, mely a K kapcsoló átváltásáig felfüggeszti a program futását.

Program:

```
program '900
$if (=k) $then
    $while (=k) $do
    $endwhile
$else
    $while (#k) $do
    $endwhile
$endif
end
```

Ugyanez, hagyományos PC assemblerben:

```
program '900
if k, to +3
if k, to +3
to -1, 0
if k, to +0
to 0, 0
end
```


3.3. Feltételek

A strukturált kontrollszerkezetekben az alábbi feltételtípusokat használhatjuk:

- a./ reláció két rekesz között
- b./ reláció Ac és egy rekesz között
- c./ PC skip feltételek
- d./ az előző negáltjai

Példák:

```
$if (a,&t,b) $then  
$while (ac,gt,x) $do  
$until (=z)  
$if (#z) $then
```

A feltételeket zárójelbe kell tenni.

A használható reláció-műveletek:

&t	(kisebb mint)
&e	(kisebb egyenlő)
gt	(nagyobb mint)
ge	(nagyobb egyenlő)
eq	(egyenlő)
ne	(nem egyenlő)

A relációműveletet az operandusoktól vesszővel kell elválasztani.

Ac operandus csak első helyen szerepelhet.

A PC skip feltételei elé az = jelet kell tennünk, a negált skip feltételre való hivatkozásnál pedig a # jel kerül a skip kódja elé.

Az operandusok közt konstansok is szerepelhetnek, literal formában.

P1: \$ if(a,lt,(5))∪\$ then

4. Blokkok

A PCL strukturált utasításait a fordítóprogram először makróként alacsonyabb szintre bontja, majd egy blokk-transzlator kezeli.

A blokk-transzlator által kezelt strukturált séma általánosan is használható, a felhasználó saját blokkokat definiálhat. A sémaleírás azonban elég bonyolult, a felhasznált vezérlőjelek egyáltalán nem olvasmányosak, így valószínű, hogy csak a legigényesebb programozók élnek a blokkszerkezet lehetőségével.

Részükre megadjuk a vonatkozó irodalmat, és itt egy számlált ciklust megvalósító blokkot példaként. A formalizmust a makrók segítségével hozzuk a felhasználóhoz közel. További példák a PCL makrokönyvtárban levő, strukturált utasításokat leíró makrók.

A blokkstruktúra ötlete az alábbi cikkből származik:

S. Kawai:

A Semiblock Structure for Low-level Languages

Software Practice & Experience, Vol.10,11-19(1980)

Az ötlet lényege az, hogy a felhasználó bizonyos jelekkel blokkokat definiálhat. A blokk kezdetén, végén és egy kilépési pontján automatikusan generálódó címkére a blokkon belül a programból hivatkozni lehet, továbbá

lehetőség van blokkon belüli, lokális címkék használata is.

A használt jelek:

[	blokk kezdet
]	blokk vég
!	kilépési pont
\$	cimkeszimbólum

Példa:

Igen gyakori a leszámított ciklusok használata, mint a FORTRAN-ban a DO, PASCAL-ban ALGOL-ban a for.

A PCL-ben is lehetőség van egy for utasítás definiálására, melynek a szintaxisa pl. lehet:

```
$for (<ciklusváltozó>, <kezdőérték>, <végérték>) $do
```

```
[ciklustörzs]
```

```
$endfor.
```

A realizáló makrók:

```
<$for ($1, $2, $3) =  
  num $2, co -1  
  
  st $1, 0  
  [lo $1, co +1  
    st $1, su $3  
    if m, to  
  ]  
>
```

```
<$do = $!>
```

```
<$endfor = to $]>
```

5. Strukturált lista

A korszerű software eszközök közül számos képes strukturált programok automatikus elemzésére és grafikus megjelenítésére /SIEMENS:COLUMBUS, IBM:GREENPRINT; hazánkban a SOMIKA/.

A PCL fordításkor GREENPRINT-szerű ábrát tud adni, mely grafikusán megjeleníti az összetartozó blokkokat és kiváló áttekintést ad a program kontrollszerkezetéről.

Az ábra jellemzői az alábbi példa listájából nyilvánvalóak. A példaprogram csak demonstrációs célra készült, értelme nincs.

Az ábrában az egyes strukturablokkok paragrafuszozás, kezdőblokk-végblokk kiemelés és vízszintes vonal-elválasztó révén jól elkülönítetten láthatók. Jelölve vannak az ábrán a nem strukturált kontrollszerkezetek is. A blokkok kezdetét és végét összekötő vonal karakterei a blokk típusára jellemzőek.

- D - makrodefiníció
- V - vagylagos blokk /IF-THEN-ELSE/
- x - ciklus
- I - modul

A + karakter ill. a belőle kiágazó nyíl a nem strukturált ugrásokat jelzi.

```
*****
*
* ELZO FAZIS TESZTJE
*
*****
```

PROGRAM 100

```
DS      BERKOLTSEG      1
DS      JUTALOM         5
DS      ATALANY         3
```

```
LOAD A,STORE 30:+5
LOAD JUTALOM,STORE ATALANY
LOAD JUTALOM *COMMENTRE PELDA,STORE B
LOAD J,STORE BERKOLTSEG      *MASODIK RESZBEN ALLO COMMENT
```

```
[ ]      <ILOAD=
D          STORE A,LOAD JUTALOM
+--->      TO 2,0
D          STORE X,LOAD 20:
D          STORE 20:+1,STORE X10
[ ]      >
```

```
[ ]      <$IOP($1,$2,$3)=
D          STORE 31:,LOAD $2
D          ADD $3,STORE 30:
+--->      TO +2,0
D          31:0
D          30:0;$1 '0
D          LOAD 31:,0
[ ]      >
```

*

```
[ ]      $IF (JUTALOM,LT,A) $THEN
V          STORE 28:,LOAD 4:
V          ADD 4:+3,ST X2
V          SH /8,ST 30:-4
V
V      [ ]      $IF (AC,GT,ATALANY) $THEN
V          LO JUTALOM,ST B
V          ST ATALANY,0
V      +--->      IF Z,TO 30:
V      [ ]      $ENDIF
```

[] \$ELSE

*UJABB FELTETEL

```
[ ]      $WHILE (JUTALOM,EQ,30:) $DO
V          *          LOAD A,STORE B
V          *          LOAD B,STORE C
V          *
V          *      [ ]      $REPEAT
V          *          *          LOAD C,ST F
V          *          *          SUB K,ADD (5)
V          *      [ ]      $UNTIL (=K)
V          *
V      [ ]      $ENDWHILE
```

```
LO E,ST V
ADD (2),ST X
```


6. Egyéb assembler fejlesztések

6.1. Szimbólumok

A PC-4000 assemblerében egybetűs szimbólumokat, illetve numerikus címkeket használhatunk változók jelölésére. Ez nem túl szerencsés megoldás, hiszen így nincs mód a változók öndokumentálására. Nehezen olvasható például az alábbi programrészlet:

```
$iop (load, 30:, i)
$iop (add, 29:, j)
      store, 35:, to +2
      :
```

Nyilvánvalóbb a programrészlet jelentése az alábbi formából:

```
$iop (load, beralap, i)
$iop (add, jutalék, j)
      store futás, to +2
      :
```

A PCL révén lehetőség nyílik egy betűnél hosszabb szimbólumok használatára.

A maximális hossz 11 karakter.

Mivel a PCL-ben adott a lehetőség szimbolikus címkek és változók használatára, soha ne használjunk numerikus címkeket és hagyományosan /SET révén/ definiált egybetűs szimbólumokat. A PCL fordító nem jelez, ugyan hibát, ha numerikus címkekkel dolgozunk, de könnyen származhat hiba az automatikusan generált numerikus címkek és a felhasználó numerikus címkeinek azonossága esetén.

A PCL-ben tehát az assembler szimbolizmusnak a szokásos assembler nyelveknél fellelhető formáját kell használni.

Szabályok:

- A szimbólum alfabetikus karakterrel kezdődő alfanumerikus string.
- Maximális hossza 11 karakter.
- Minden szimbólumot címkeként definiálni kell!

Példa egy PCL szimbólumu assembler részletre:

Cimkifejezések:

A címzésekben egyszerű cimkifejezéseket használhatunk, aminek formája

szimbólum $\pm$ konstans

lehet.

Pl: store cimkeindex+5, load work-3

A növekmény csak konstans lehet, a változóval indexelt címzéshez már makrókat kell használni.

Tiltott a PC-4000 assemblerben az egybetűs szimbólumok mellett használt

pl: load a5, store x 4

stb. címkifejezés formája. A PCL az a5-öt egyetlen alfanumerikus string-nek, azaz változónak fogja fel, és ennek megfelelően kezeli.

A címkifejezésben nem lehet space.

Címkefeloldások /a RESET utasítás/

A PCL fordító a PC-4000 assemblerére fordít. Az viszont csak 256 címkét enged meg. Egy komoly programrendszerben ennél jóval többet kell használni.

Ezért vezettük be a RESET utasítást.

Formája:

RESET cimke_cimke_cimke_...

Pl.

RESET I_MUNKA_WORK_BER

Funkciója:

A megjelölt szimbólumokat a továbbiakban nem kívánjuk használni. A hozzárendelt szám a fordítás folyamán kiosztható más szimbólum számára.

A PC-4000 assembler a reset utasításoknál **lr** direktívákat értelmez. Az **lr**-eket a PCL fordító generálja.

6.2. Commentek

A PCL-ben a commenteket `*` jel vezeti be, bárhol is forduljanak elő. Ezzel valamivel áttekinthetőbb fejléc- és elválasztó commentezést tesz lehetővé.

6.3. Az INCLUDE utasítás

Egy hosszú programot gyakran részekben írunk. A részek összeolvasztásához hasznos az include utasítás.

Formája:

```
include file-név
```

Például:

```
include sum.pcl
include táblázás.pcl
```

```
⋮
```

Funkciója:

az `include` helyére a PCL fordítás előtt bemásolódik a megadott nevű file.

Az `include` nem szubrutinhívás és nem makro lehetőség, pusztán széttagolt szöveg összeillesztésére szolgál.

7. A PCL fordító futtatása

7.1. A fordító elérése

A futtatás a

ⓐ PCL

procedura-hívással indítható. A fordítóprogram a rendszerkönyvtárban helyezkedik el.

Három fázisban fordít.

7.2. A fordító futtatása

a./ A fordítót hívni a

ⓐ PCL

paranccsal lehet.

b./ Hívás után a fordító lekérdezi az input file nevét.

INPUT FILE:

Üssük be a terminálról az input file nevét,
extension nélkül, bármely nem alfanumerikus karakterrel /pl. return-nel/ lezárva.

Az extension automatikusan PCL lesz.

c./ A következő kérdés arra vonatkozik, kér-e a felhasználó strukturált listát.

KER STRUKTURALT LISTA^T(I/N):

Az I betűvel igent jelzünk, más karakterrel a nem-et.

- d./ Egy listázó és előkészítő fázis után a gép a makro-könyvtár nevére kérdez rá.

MAKROKÖNYVTAR NEVE:

Üssük be a használt könyvtár nevét, extension-nal együtt.

A standard makrók a MAC.LIB könyvtárban találhatók.
(a [305,307] -ben)

- e./ A PCL lefutása után az

input filenév.PCL változatlan

input filenév.LST file a listát tartalmazza

input filenév.PCA az assembler program

input filenév.STL a szimbólumtábla.

A hibaüzenetek a konzolon íródnak ki.

7.3. Példa:

A következőkben egy felhasználó teendőit szemléltetjük, egy futtatás menetének és eredményeinek bemutatásával.

IAS PROGRAM DEVELOPMENT SYSTEM VERSION 2.0
12:44:02 28-SEP-81

← BE JELENTKEZÉS

PDS> LOGIN WALTER
PASSWORD?

User WALTER UIC [305,307] TT15: Job-id 65 12:44:11 28-SEP-81
***** KOSZONTJUK A SZAMOK IAS V2.1 RENDSZEREKEN! *****
MAGYAR NYELVU UJSAG A "HIREK" PARANCCSAL KAPHATO.
IN ENGLISH YOU CAN GET THE NEWS VIA PDS>NEWS COMMAND.

LEGUJABB HIR 1980.OKT.23

KERJUK, HOGY JELSZAVAT VALTOZTASSA IDONKENT A "PDS>SET PAS" PARANCCSAL
--- TARTSUK A TERMINALTERMET TISZTAN! ---

PDS> TYPE T3.PCL

* SKALARIS SZORZAT, RUTINOKKAL
*

PROGRAM '200
ORG '1LO '100
SET A B C W1 P
PC P

* SZAMJEGY BEOLVASASA

O *KILEPES
2:TA 100,AN ('70)

ST C,SU ('30)

IF N,TO 2:-1

NU '39,SU C

IF N,TO 2:-1

LO C,TO 2:-1

*DECIMALIS BEOLVASAS

O *KILEPES

8:RE 2:-1,TO 2:

IF N,TO -1

AN ('0),ST A

RE 2:-1,TO 2:

IF N,TO 8:-1

AN ('0),ST W

LO A,SH +4

MU ('5000,0),AD W

ST A,TO -5

*DECIMALIS KIRO

O *KILEPES

7:LO B,SH /17

ST W,LO A

DI W,SH -16

SH /16,ST W1

AD (LO 0:,TA 000),ST +1

O

LO W1,MU W

NE 2,AD A

ST A,0

LO B, DIV (5,0)

← A PCL SOURCE
KIIRATASA

'K100,0

'K200,0
'3300,0
'K400,0
'3500,0
'3600,0
'K700,0
'K800,0
'3900,0

SET S N X10 Y10 I M Z Q
PC Q

<\$CLEAR(\$1)=
NUM 0,ST \$1

<\$SELECT(\$1)=
LOAD (\$1),PE A00
SH /8,PE A00

<\$READINT(\$1)=
RE 8:-1,TO 8:
LO A,ST \$1

<\$PRINTINT(\$1)=
LO \$1,ST A
LO (100),ST B
RE 7:-1, TO 7:

<\$IOP(\$1,\$2,\$3)=
ST Z,LO (LO Z,\$1 \$2)
AD \$3,ST +1

<\$NEWLINE=
NU '8M,SH /8
TA 000,0

<\$STOP=
ME 000,TO '140

*
*
* FOPROGRAM
*
*

10:NOTE START
\$SELECT('14000093)
\$READINT(N)
\$FOR(I,1,N) \$DO
\$READINT(M)
\$IOP(STORE,X,I)
\$ENDFOR
\$FOR(I,1,N) \$DO
\$READINT(M)
\$IOP(ST,Y,I)
\$ENDFOR
\$CLEAR(S)
\$FOR(I,1,N) \$DO
\$IOP(LO,X,I)
SH /16,ST M
\$IOP(LO,Y,I)
SH /17,MU M
AD S,ST S
\$ENDFOR
\$SELECT('14000093)

*Is a template
program!*

A PCL FORDITO
FWTATAS

@PCL
CL1 INPUT FILE NEVE: T3
KER STRUKTURALT LISTAT?(I/N):I
CL2
FIX MACRO FILE NEVE:MAC.LIB
NO ERROR

DEL/KEEP STD.XXX

PDS> TYPE T3.LST

← A LISTA KIIRAT
SA

*
* SKALARIS SZORZAT, RUTINOKKAL
*

[C] PROGRAM '200
I ORG '1LO '100
I SET A B C W1 P
I PC P
I *****
I * SZAMJEGY BEOLVASASA
I O *KILEPES
I 2:TA 100,AN ('70)
I ST C,SU ('30)
+----> IF N,TO 2:-1
I NU '39,SU C
+----> IF N,TO 2:-1
+----> LO C,TO 2:-1
I *****
I *DECIMALIS BEOLVASAS
I *****
I O *KILEPES
+----> 8:RE 2:-1,TO 2:
+----> IF N,TO -1
I AN ('0),ST A
+----> RE 2:-1,TO 2:
+----> IF N,TO 8:-1
I AN ('0),ST W
I LO A,SH +4
I MU ('5000,0),AD W
+----> ST A,TO -5
I *****
I *DECIMALIS KIIRO
I *****
I O *KILEPES
I 7:LO B,SH /17
I ST W,LO A
I DI W,SH -16
I SH /16,ST W1
I AD (LO 0:,TA 000),ST +1
I O
I LO W1,MU W
I NE 2,AD A
I ST A,0
I LO B,DIV (5,0)
I SH -16,ST B
+----> IF Z,TO 7:-1
+----> TO 7:,0
I *****
I O:'3000,0
I 'K100,0

```

      'K800,0
      '3900,0
      *-----
      SET S N X10 Y10 I M Z Q
      PC Q

```

```

[ ]    <$CLEAR($1)=
D      NUM 0,ST $1
[ ]    >

```

```

[ ]    <$SELECT($1)=
D      LOAD ($1),PE A00
D      SH /8,PE A00
[ ]    >

```

```

[ ]    <$READINT($1)=
+----> RE 8:-1,TO 8:
D      LO A,ST $1
[ ]    >

```

```

[ ]    <$PRINTINT($1)=
D      LO $1,ST A
D      LO (100),ST B
D      RE 7:-1, TO 7:
[ ]    >

```

```

[ ]    <$IOP($1,$2,$3)=
D      ST Z,LO (LO Z,$1 $2)
D      AD $3,ST +1
D      0
[ ]    >

```

```

[ ]    <$NEWLINE=
D      NU '8M,SH /8
D      TA 000,0
[ ]    >

```

```

[ ]    <$STOP=
+----> ME U00,TO '140
[ ]    >

```

```

      *-----
      *
      * FOPROGRAM
      *
      *-----
      10:NOTE START
      $SELECT('14000093)
      $READINT(N)

```

```

[ ]    $FOR(I,1,N) $DO
*      $READINT(M)
*      $IOP(STORE,X,I)
[ ]    $ENDFOR

```

```

[ ]    $FOR(I,1,N) $DO
*      $READINT(M)
*      $IOP(ST,Y,I)
[ ]    $ENDFOR

```

*A tinylog
program
structure
list*

```

I      *      $IOP(LO,Y,I)
I      *      SH /17,MU M
I      *      AD S,ST S
I      []      $ENDFOR
I      -----
I      $SELECT('14000093)
I      $NEWLINE
I      $PRINTINT(S)
I      $STOP
[]      END
-----

```

***** HOSSZU SZIMBOLUMOK *****

PDS> TYPE T3.PCA

```

NOTE *****
NOTE *
NOTE *   SKALARIS SZORZAT, RUTINOKKAL
NOTE *
NOTE *****
PROGRAM '200
ORG '1LO '100
SET A B C W1 P
PC P
NOTE *****
NOTE * SZAMJEGY BEOLVASASA
O NOTE KILEPES
2:TA 100,AN ('70)
SP C,SU ('30)
IF N,TO 2:-1
DJ 39,SU C
IF N,TO 2:-1
LO C,TO 2:-1
NOTE *****
NOTE *DECIMALIS BEOLVASAS
NOTE *****
O NOTE KILEPES
8:RE 2:-1,TO 2:
IF N,TO -1
AN ('0),ST A
RE 2:-1,TO 2:
IF N,TO 8:-1
AN ('0),ST W
LO A,SH +4
MU ('5000,0),AD W
ST A,TO -5
NOTE *****
NOTE *DECIMALIS KIIRO
NOTE *****
O NOTE KILEPES
7:LO B,SH /17
ST W,LO A
DI W,SH -16
SH /16,ST W1

```

→ A2 OUTPUT
SOURCE
KIRALY


```

TO 7:0
NOTE *****
O:3000,0
'K100,0
'K200,0
'3300,0
'K400,0
'3500,0
'3600,0
'K700,0
'K800,0
'3900,0
NOTE *-----
SET S N X10 Y10 I M Z Q
PC Q

```

```

NOTE *-----
NOTE *
NOTE * FOPROGRAM
NOTE *
NOTE *-----
10:NOTE START

```

```

LOAD ('14000093),PE A00
SH /8,PE A00

```

```

NOTE -----SELECT('14000093)

```

```

RE 8:-1,TO 8:
LD A,ST N

```

```

NOTE -----READINT(N)
NOTE -----FOR(I,1,N) DO
NUM 1,CO -1
      ST I,0

```

```

60: LD I,CO +1
      ST I,SU N
      IF Z,TO +2
      IF P,TO 62:

```

```

61:
RE 8:-1,TO 8:
LD A,ST M

```

```

NOTE -----READINT(M)

```

```

ST Z,LO (LO Z,STORE X)
AD I,ST +1
O

```

```

NOTE -----IOP(STORE,X,I)
NOTE -----ENDFOR
      O,TO 60:
62:   LR 60 62
NOTE -----FOR(I,1,N) DO
NUM 1,CO -1
      ST I,0

```

```

60: LO I,CO +1
60: LD I,CO +1

```

NOTE -----READINT(M)

ST Z,LO (LO Z,ST Y)
AD I,ST +1
0

NOTE -----IOP(ST,Y,I)
NOTE -----ENDFOR

0,TO 60:
62: LR 60 62

NUM 0,ST S

NOTE -----CLEAR(S)
NOTE -----FOR(I,1,N) DO

NUM 1,CO -1
ST I,0

60: LO I,CO +1
ST I,SU N
IF Z,TO +2
IF P,TO 62:

61:
ST Z,LO (LO Z,LO X)
AD I,ST +1
0

NOTE -----IOP(LO,X,I)
SH /16,ST M

ST Z,LO (LO Z,LO Y)
AD I,ST +1
0

NOTE -----IOP(LO,Y,I)
SH /17,MU M

AD S,ST S
NOTE -----ENDFOR

0,TO 60:
62: LR 60 62

LOAD ('14000093),PE A00
SH /8,PE A00

NOTE -----SELECT('14000093)

NU '8M,SH /8
TA 000,0
NOTE -----NEWLINE

LO S,ST A
LO (100),ST B
RE 7:-1, TO 7:

NOTE -----PRINTINT(S)

ME U00,TO '140
NOTE -----STOP
END

PDS> LGOGUT
Illegal command.

← RIFELENTKE

STANDARD MAKROK

```

TYPE MAC.LIB
<$IF= >
<$THEN=$C>
<$ELSE= 0,TO $I>
<$ENDIF=J >
<$WHILE=$C>
<$DO=$I>
<$ENDWHILE=      0,TO $J>
<$REPEAT=$C>
<$UNTIL=$J >
<$FOR=NUM $2,CO -1
      ST $1,0
      C LO $1,CO +1
      ST $1,SU $3
      IF Z,TO +2
      IF P,TO>
<$ENDFOR=      0,TO $J >
<$EQ=  LO $1,SU $2
      IF Z, TO +2
      0,TO>
<$NE=  LO $1,SU $2
      IF Z,TO>
<$LT=  LO $1,SU $2
      IF N,TO +2
      0,TO>
<$GE=  LO $1,SU $2
      IF N,TO>
<$LE=  LO $1,SU $2
      IF N,TO +3
      IF Z,TO +2
      0,TO>
<$GT=  LO $1,SU $2
      IF Z,TO +2
      IF P,TO +2
      0,TO>
<$AEQ=  0,SU $2
      IF Z,TO +2
      0,TO>
<$ANE=  0,SU $2
      IF Z,TO>
<$ALT=  0,SU $2
      IF N,TO +2
      0,TO>
<$AGT=  0,SU $2
      IF Z,TO +2
      IF P,TO +2
      0,TO>
<$ALE=  0,SU $2
      IF N,TO +3
      IF Z,TO +2
      0,TO>
<$AGE=  0,SU $2
      IF P,TO +2
      0,TO>
<$EQSK= IF $1,TO +2
      0,TO>
<$NESK= IF $1,TO>

```

PDS>

Simulator

Felhasználói Kézikönyv

1981. augusztus

Bevezetés

A PCSIM szimulátor lehetővé teszi PC 4000-es gépre írt programok működtetését PDP 11/70-es gépen. Használata elősegíti a programok belövését, megkönnyíti tesztelését, egyszerűsíti a program javítást. A PDP számítógép nagy központi sebessége és gyors perifériái megengedik azt is, hogy a szimuláláskor ne következzen be számottevő sebesség csökkenés. Így a belőtt programok futtatása is megfelelő hatékonysággal történhet a helyettesítő számítógépen.

'Milyen előnyökkel bír a Simulator?'

A kiváltó számítógépen üzemelő IAS operációs rendszer korszerű timesharing rendszer, amely jelenleg egyidejűleg 16 terminálon elégíti ki a párhuzamos felhasználói igényeket. Lehetőség van tehát akár az összes terminálon egyidejűleg PC 4000-es fejlesztéseket végezni. A programozók ugyanakkor élhetnek az operációs rendszer egyéb segédprogramjaival is /szövegszerkesztő program használata, file-kezelő segédprogramok stb/.

A Simulator beépített dinamikus programjavító / lekérdező rendszert is tartalmaz /Ez természetesen normál futáskor kikapcsolható/. A programjavító rendszer segítségével a program futást tetszőleges helyeken megállíthatjuk,

program és adatrészeket lekérdezhetünk és módosíthatunk, program szakaszokat átléphetünk, ill. különböző adatokkal többször ismételhetünk. Természetesen a panelkapcsolók állapotát is változtathatjuk ill. lekérdezhetjük.

A Simulator ugyanakkor rendelkezik a statikus, lépésenkénti nyomkövetés listázási lehetőségével is.

'A Simulator illeszkedése a teljes rendszerbe.'

1/ A PCL nyelven megírt programot az IAS editorával egy lemezfile-ba helyezzük.

2/ A PCL fordítóprogrammal ezt a file-t a PC gép Assembler nyelvére fordítjuk.

3/ A PCSIM szimulátor rendszerrel futtatjuk az Assemblert, amely előállítja a kívánt tárgyprogramot.

4/ A PCSIM ismételt futásával ezen tárgyprogramot teszteljük, javítjuk.

5/ A feltárt hibákat az 1/-ben leírt editorral javítjuk és ismételjük az előző négy fázist.

6/ A hibátlan tárgyprogramot átvisszük a PC-4000-es gépre.

A Simulator rendszer nyelve PASCAL és MACRO-11.

A kézikönyv 1. fejezetében vázlatosan ismertetjük a Simulatort, a 2. fejezetében a használatához szükséges tudnivalókat részletezzük, a 3. fejezetében az eredmények PC gépre történő átviteléről szólunk, végül az utolsó fejezetben egy példán szemléltetjük a Simulator működését és kezelését.

1. A Simulator vázlatos ismertetése

A Simulator feladata a PC 4000-es számítógép Assembler nyelvén írt programok PDP 11/70-es IAS op sys alatt történő szimulálása, breakpoint-ozási lehetőség használata és az utasításonkénti nyomonkövetés. A fenti tevékenységek biztosítják a PC 4000 számítógép kiváltását, a programok kényelmes tesztelését - javítását és a futás lépésenkénti nyomonkövetését.

A Simulator inputja egy egész típusu file, amelyet az előző fázis /Cross-Assembler/ állít elő. Tevékenysége három fő részre osztható:

- Az assembler programok szimulálása
- Breakpoint
- Lépésenkénti nyomonkövetés

1.1 Az assembler programok szimulálása

Az input file-t /amely a végrehajtandó/tesztelendő programot tartalmazza/, amelynek a felhasználó bármilyen nevet adhat, a Simulator a tárba helyezi. Szimulálja a PC 4000 számítógép regisztereit:

- az akkumulátort /AC/
- a cím regisztert /REG/
- az utasításszámlálót /PC/
- az I/O regisztert /KAR/
- és a disc status szavakat /DISCST/

Kijelöli a lemez puffereket is /PUFF/.

Induláskor a terminálon kérdéseket tesz fel a felhasználónak /lásd következő fejezet/, amelyekre adott válaszokkal beállítja a kezdeti értékeket és állapotokat. A megnevezésekkel a PC 4000-es gép perifériái azonosíthatók. Az egyes perifériák a PDP 11/70-es gépen a következő módon kerültek kijelölésre

- lyukszalagolvasó - /egész típusu/ diszk file
- lyukszalaglyukasztó - /egész típusu/ diszk file
- consol írógép - terminál
- sornyomtató - lemezfile /karakter típusu elemekkel/
- diszk - diszk file /fix névvel/.

A későbbiek során lehetőség lesz a floppy disc-ek diszken történő szimulálására is. Mindezen kezdeti értékadások és kijelölések után a megadott kezdőcímtől kezdődően a Simulator elkezdi a tárba helyezett program futtatását.

A futtatást a breakpoint-on keresztül lehet vezérelni /lásd 1.2 alfejezet/. Ha a végrehajtás pontos utasításonkénti sorrendjére vagyunk kíváncsiak, akkor a nyomkövetés

kérésével a sornyomtatón ezt is megkaphatjuk. A nyomkövetést a breakpoint-tal kombinálva elérhetjük, hogy a program tetszőleges részét kijelölhessük nyomkövetésre. A szimulálás befejezésekor a rendszer megkérdezi a felhasználót, hogy a tárban lévő /javított/ programot kéri-e megőrzésre. Ha igen, akkor a Simulator kiírja a tár tartalmát /a megadott kezdő és végcímek között/ a lyukszalaglyukasztóhoz rendelt file-ra. 2

1.2 Breakpoint

A "Breakpoint" gyűjtőnév alatt itt egy, a szimulátorba épített, tesztelő-hibakereső rendszert értünk, amelynek segítségével a programot tetszőleges helyeken megállíthatjuk /töréspontok/ és a teljes programkörnyezetet /ill. annak bennünket érdeklő részeit/ lekérdezhetjük és módosíthatjuk. Amikor a program egy törésponthoz ér, a vezérlést a breakpoint rendszer kapja. A töréspontokon a program várakozik és egy egyszerű parancsnyelv segítségével kommunikál a felhasználóval mindaddig, amíg az nem indítja tovább a programot. /A lehetséges parancsokat a 2.2 fejezetben ismertetjük./

A program kezdő párbeszédsorozata után azonnal a breakpoint rendszer kapja a vezérlést. Ekkor lehet a kívánt programhelyekre töréspontokat beállítani, a panel K,L,M kapcsolóinak új értékeket ill. egyes változóknak teszt

értéket adni. Itt is kapcsolhatjuk a statikus nyomkövetést, amely a breakpoint nyújtotta interaktív tesztelési móddal ellentétben minden utasítárról egy sort nyomtat, tehát az utólagos elemzéshez/hibakereséshez nyújt segítséget. Ez a következő töréspontig működik, ahol újra dönthetünk ki-vagy bekapcsolásáról. Az újabb töréspont alkalmával ismét a teljes utasításkészlet rendelkezésre áll, tehát a tártartalmat/regisztereket módosíthatjuk, panelkapcsolókat állíthatjuk és töréspontokat törölhetünk ill. újabbakat állíthatunk be.

1.3 Utasításonkénti nyomkövetés

A Simulator első fázisában megkérdezi a felhasználótól, hogy szüksége van-e utasításonkénti nyomkövetésre. Ha az erre adott válasz 1 /igen/, akkor a Simulator minden utasítás végrehajtása során a következő információkat írja ki a terminálra./Minden szám hexadecimálisan értendő/.

- az utasítás sorszáma
- az utasítás neve
- az utasítás további pontosítása /ha kell/
- cím, tártartalom
- az akkumulátor tartalma

Lehetőség van továbbá bármely programrész önálló nyomkövetésére /lásd 2.2/.

2. A Simulator kezelése

A Simulator a PDP terminálra való bejelentkezés után PDS> PCS paranccsal kell indítani. Ezután a program a terminálon kérdéseket tesz fel, amelyre a válaszokat a kérdés mellé kell beütni. Helytelen válasz esetén a kérdés megismétlődik. Breakpoint fázisban nem kérdések jelennek meg, hanem a felhasználó lekérdezheti az egyes regiszterek, rekeszek stb. állapotát. A Simulator felhasználóval kapcsolatos kommunikálását két részre bontjuk.

- A szimulálás és a nyomkövetéssel kapcsolatos kérdések kezelése
- a breakpoint kezelése.

2.1 A szimulálással és nyomkövetéssel kapcsolatos kérdések kezelése

Az alábbiakban tételesen felsoroljuk az egyes kérdéseket a megjelenés sorrendjében és a lehetséges válaszokat. Minden választ RETURN billentyűvel kell elküldeni.

'Ø-S BLOKKBA KERULO FILE:'

válasz: Az IAS filenév konvencióknak megfelelő név /betüből, számból és tizedespontból álló string/,

amelybe előzetesen elhelyeztük a program kezdő és végcímét /ezek az Assembler fordító használata során a program kiíratásakor, a file-on kiírásra kerülnek/ és a programot. Könyvtárunkban létező file-t kell megadni, különben a rendszer abortál.

A kérdés addig jelenik meg, amíg válasz nélkül a RETURN billentyű beütésével el nem engedjük. Ezzel a lehetőséggel több programtárba olvasását érhetjük el.

'1-S BLOKKBA KERULO FILE:'

válasz: lsd az előző kérdést.

'PRINTER FILE:'

válasz: A beütendő file névre vonatkozó szabályt lásd az 1. kérdésnél. Hatására a sornyomtatóra kiirandók számára egy file nyílik meg, amelyet a Simulator lefutása után egy >PRINT filenév paranccsal kiirattathatunk a sornyomtatóra. Egyszerű RETURN hatására a file nem képződik.

'MEMORIA / ϕ ,1/:'

válasz: azt a számot / ϕ ,1/ kell beütni, amely blokkban indul a program végrehajtása.

'START CIM:'

válasz: hexadecimális szám, ahonnan a program indul. Az utolsó két kérdés az első breakpoint megjelenése során

még időben javítható!

'LYUK-I FILE:'

válasz: A lyukszalagolvasóhoz tartozó file nevét kell beírni. Egyszerű RETURN-re a file nem képződik.

'LYUK-O FILE:'

válasz: A lyukszalaglyukasztóhoz rendelendő file nevét kell beírni. Egyszerű RETURN-re a file nem képződik.

A fenti kérdések után a Simulator elkezd a program végrehajtását. Ennek első lépéseként a Breakpoint fázis indul el.

A Simulator a program befejezésekor /'STOP'/ ismét meghívja a Breakpoint rendszert /ellenőrizhetjük a tárbeli eredményeket/, majd az alábbi kérdéseket teszi fel:

'Ø-SBOL HOVA:'

válasz: azt a filenevet kell beírni, ahová kérjük az eredményt. Szintaktikájára lásd A Ø-AS BLOKKBELI... A kérdés addig jelenik meg, amíg egy egyszerű RETURN-nel el nem engedjük

['KEZDOCIM:']

válasz: hexadecimális szám /1Ø-15-ig a J,K,L,M,N,O betűket kell beírni/

['VEGCIM:']

válasz: hexadecimális szám

'1-SBOL HOVA:'

válasz: a képezendő output file. Szintaktikájára
lásd 'A $\emptyset$ -SBOL HOVA...

'KEZDOCIM:'

válasz: hexadecimális szám

['VEGCIM:']

válasz: hexadecimális szám

A Simulator befejezésekor a terminálon megjelenik a

'BYE' üzenet.

2.2 A Breakpoint kezelése

A Breakpoint rendszer mindig a BN:C,PC formában jelentkezik be, ahol N $/0 \leq N \leq 8/$ az illető töréspont száma, C a memoria blokk azonosító. $N=0$ ha a program végrehajtása lépésenként történik /lásd S parancs/, ill. a rendszer 1. hívásakor. PC helyén az aktuális utasításslámláló tartalma jelenik meg. Ezután az alulvonás $/_/$ karakterrel jelzi, hogy felhasználói parancsra vár. A parancsok egy betűsek, ezekhez kiegészítő karakterek járulhatnak. Érvénytelen parancs vagy karakter megadásakor '?' után ismét új parancsra vár.

A Breakpont rendszer utasításait 4 csoportba osztjuk.

2.2.1 Tártartalom lekérdező/módosító utasítások

A./ [.] cim { :|.} tartalom [ujtartalom] zárókarakter

/A metaszimbolumok jelentése a szokásos, az opcionális részeket [] szögletes zárójelek közé helyezzük, a kötelezően választható alternatívákat { } figuratív zárójelbe, egymástól | karakterrel elválasztva. A gép által visszairt információt aláhúztuk./

B./ [.] cimkezdő, zárócim { :|.} tartalom1 tartalom2 ...

Értelmezése:

A./ Cim: a kiirandó társzó hexadecimális cime.

/Ha nem létező tárcimre hivatkozunk, '?' a válasz./

ha az opcionális '.' karakterrel kezdünk, akkor a cimet decimális formában várja.

a tartalom megjelenési formája a megelőző karaktertől függ.

' :' esetében hexadecimális számként iródik ki

' .' " decimális " " "

az ujtartalom ugyanebben a formában gépelhető be, amennyiben a korábbi tartalmat módosítani akarjuk. Ha a begépelte számsorozat rövidebb, úgy balról nullákkal töltődik fel a szó.

zárókarakter lehet

CR /return/ - hatására az aktuális megjelenítés módosítás befejeződik, a rendszer a '_' karakterrel visszajelentkezik; új parancsra vár.

LF /line feed/ - hatására a művelet befejeződik, és automatikusan a következő szó tartalma jelenik meg, az előző sorban megadott formátumnak megfelelően.

/Ezáltal hosszabb, folyamatos tárterületet módosíthatunk, minimális gépeléssel./

BS /backspace/ - az LF-el azonos formájú, de a megelőző szó tartalmára vonatkozik.

/A tárban visszafelé haladva kérdezhetünk le ill. módosíthatunk./

B./ Hosszabb tárintervallum folyamatos kilistázására alkalmas utasítás.

cimkezdő ill. zárócim adja az intervallum kezdő ill. végpontjait /a '.' megadásától függően decimális/hexás címként/ és a kilistázás ':' ill. '.' hatására hexás ill. decimális lesz. Egy sorba 7 szó tartalma kerül, módosításra lehetőség ekkor nincs.

2.2.2 Töréspontok kezelése

Ezekben az utasításokban zárókaraktert nem szabad adnunk!

A./ A program tetszőleges helyére /szókezdetre!/ töréspontokat szurhatunk be.

Szintaktikája:

[.] cim;B[N]

Ahol a cim hexás ill. decimális /ha '.' előzi meg/.

Egyidőben maximálisan 8 töréspont létezhet, ezek egytől nyolcig számozódnak. N egy szám, amely azt jelenti, hogy a Breakpoint N-ik előfordulásakor jelenik meg először.

B./ A töréspontokat kilistázhatjuk.

Szintaktikája:

\$[N]{:|.}

Hatására a N. töréspont ($1 \leq N \leq 8$) helyének címe megjelenik hexadecimális (':') ill. decimális ('.') formában.

Ha N elmarad, akkor az összes töréspont kilistázódik.

C./ Feleslegessé vált töréspontokat törölhetünk.

Szintaktikája:

[N]B

Az N. töréspont megszűnik, a programfutás továbbiakban nem szakad meg az ehhez tartozó címen.

Ha N-t elhagyjuk, akkor az összes töréspont törlődik.

2.2.3 Regiszterek, kapcsolók állapotának lekérdezése, módosítása

Ezek az utasítások mind a '#' karakterrel kezdődnek.

Zárókarakterként csak a CR megengedett!

A./ # {K|L|M}:állapot [ujállapot] zárókarakter

ahol K,L,M a megfelelő panelkapcsolókat jelenti.

az állapot $\emptyset$: ha kikapcsolva

1 : ha bekapcsolva van,

az ujállapot is ez lehet.

B./ #A:tartalom [ujtart] zárókarakter az akkumulátor tartalmát jeleníti meg; csak hexás forma megengedett.

C./ #P{:|.}cim [ujcim] zárókarakter

A PC utasításszámláló jelenlegi tartalmának megjelölése ill. módosítása /hexás ill. decimális formában/.

Módosítás esetében a program továbbindítása az ujcimen kezdődik.

D./ #E|O:tart [ujtart] zárókar

Az E parancs az akkumulátor előjelét kérdezi le;

$\emptyset$ ha pozitív, ill. 1 ha negatív.

Az O parancs a tulcsordulásjelző, amely a tényleges PC hardware-től különbözően lett implementálva.

Tartalma itt $\emptyset$, ha nincs tulcsordulás és
1 ha van.

E./ #D{ $\emptyset$ |1}:tartalom [ujtartalom] zárókarakter.

A $\emptyset$ ill. 1 lemezegység állapotregiszter tartalmát jeleníti meg ill. ez módosítható.

#T{ $\emptyset$ |1}{:|.}tartalom...

A $\emptyset$ ill. 1 lemezegység pufferének tartalmát /64 szó/
írja ki hexas ill. decimális formában.

Nem módosítható.

F./ #C:tart [ujtart] zárókarakter

Az aktuális memória blokk számát adja, illetve
ez bírálható felül. /Jelenleg $\emptyset$ -3 lehet/.

G./ #N:tart [ujtart]

tart $\emptyset$: utasításonkénti nyomkövetés kikapcsolása

ujtart 1: " " bekapcsolása

2.2.4 A Breakpoint rendszerből visszatérő utasítások.

a/ A G parancs hatására a program végrehajtása a következő töréspontig folytatódik.

b/ Az S paranccsal a program lépésenként hajtható végre. /A következő szó elején áll meg, tehát 2 utasítást hajtott végre - függetlenül az előre beállított töréspontok helyétől/.

c/ Az X parancs hatására a szimulálás befejeződik, és a záró kérdések jelennek meg. Ez egy normális, azonnali terminálás végrehajtására alkalmas, így a használt lemez állományok nem sérülnek meg /mint az ABORT esetén előfordulhat/. A záráskor ismét meghívódik a breakpoint rendszer; ezt ismét X paranccsal engedjük tovább.

2.3 Assembler program fordítása

Egy Assembler program fordítása során a kérdések száma tovább csökken minthogy a fordító program automatikusan töltődik be. Ebben az esetben a Simulator-t a >PCA paranccsal kell meghívni és a következő kérdések jelennek meg.

'A FORRAS FILE NEVE:'

válasz: filenév. Annak a file-nak a nevét kell beütni, amely a lefordítandó programot tartalmazza.

'A LYUKSZALAGLYUKASZTO NEVE:'

válasz: filenév. Lyukszalagralyukasztás esetén kell használni.

'PROGRAM STOP'

válasz: nincs, a fordítás végét jelzi.

'Ø-S BLOKKBOL HOVA MENT:'

válasz: filenév. Ha a programunk a Ø-s blokkban fordítódott, akkor a file-nevet itt kell megadni.

['KEZDOCIME:']

válasz: hexadecimális szám.

['VEGCIME:']

válasz: hexadecimális szám.

'1-S BLOKKBOL HOVA MENT:'

válasz: filenév. Akkor használjuk, ha programunk az 1-s blokkba fordítódott .

['KEZDOCIME:']

válasz: hexadecimális szám

['VEGCIME:']

válasz: hexadecimális szám.

'BYE'

A fordítás befejezését jelentő üzenet.

A fordítási lista az ASS.LST-ben képződik, amelyet egy PRINT ASS.LST paranccsal sornyomtatóra kilistázhatunk. Az outputként előállt file pedig a > PCS paranccsal futtatható!

3. A Simulator segítségével létrehozott hibátlan object program átvitele PC 4000-es számítógépre

A Simulator futása során a programunkon eszközölt javításokat/módosításokat vissza kell vezetni a forrásnyelvi programba és az így kijavított forrásnyelvi programot újra assemblálni. Az assemblálás eredményeként előálló tárgyprogram egy lemezfile-on képződik, ennek átvitelére a PC 4000-es gépre a következő lehetőség kínálkozik:

Az object programot mágnesszalagra másoljuk és azt valamilyen lyukszalaglyukasztós perifériával rendelkező géppel /pl R-gépek/ kilyukasztatjuk a PC 4000-es gép által kívánt formátumban.

Ezt a megoldást természetesen a forrásnyelvi programra is alkalmazhatjuk, ill. gép hiányában a kinyomtatott lista alapján a PC 4000-es gépen kézzel is bevihetjük /Ez a megoldás fáradságos és sok hibalehetőséget takar, viszont nem igényli egy közbülső gép beiktatását/.

4. Példa

Minthogy a Simulator kérdései szükségszerűen megjelennek, /csupán néhány kérdés opcionális/ a rendszer működését egyetlen példán keresztül mutatjuk be.

A PR3.DAT nevű file egy egyszerű, a $\emptyset$ -ás diszkrét kiíró és onnan olvasó programot tartalmaz. Az 1-es mellékletben megadott listán lépésenként kommentáltuk a tevékenységeket.

OS - 4000

operációs rendszer

- Feladatterv -

MKKE

Matematikai és Szám.tud. Intézet

1981.

OS - 4000

Tartalomjegyzék

1. A rendszermag komponensei
2. A file rendszer alapjai
 - 2.1. File fogalmak és típusok
 - 2.2. A File Directory
 - 2.3. A file-ok felépítése
 - 2.4. Directory kezelés
3. Perifériakezelés
 - 3.1. File Control System
 - 3.2. Device Handler-ek
 - 3.3. Driver-ek
 - 3.4. Device és file táblák
4. Keyboard Monitor
5. User Service Routines
6. System Utilities
 - 6.1. INIT
 - 6.2. GEN
 - 6.3. BUILD
 - 6.4. BOOT
7. Tárkezelés
8. Implementációs megfontolások
 - 8.1. Kidolgozási terv
 - 8.2. Dokumentáció és programstílus

1. A rendszermag komponensei

Az OS-4000 a PC-4000 kishszámítógép diszorientált operációs rendszere. A rendszer magját az alábbi programok alkotják.

a./ Input-Output Control System. /^{CS}IOSC/

A rendszer input/output folyamatait irányítja, három szinten. A File Control System a logikai szintű bevitelt és kivitelt, a Device Handler-ek a már numerikusan paraméterezett, de még bizonyos mértékig eszközfüggetlen bevitelt-kivitelt, a Driver-ek az egyes eszközök specifikumait figyelembe vevő fizikai input-output-ot valósítják meg.

b./ Keyboard Monitor /KBM/

Az operátor /programozó/ a géppel a Keyboard Monitor-on keresztül kommunikál. A KBM a felhasználó konzol-on beütött parancsait hajtja végre. Ezen parancsokkal lehet programokat indítani, perifériahezrendeléseket végezni stb.

c./ User Service Routines /USR/

Elemi szintű file kezelő funkciók és KBM funkciók programból való hívására ill. néhány rendszer-információ programból történő lekérdezésére ad módot.

d./ System Utilities /SU/

A rendszer generálásához, betöltéséhez szükséges programok. /GEN, BUILD, BOOT, INIT/.

A rendszermag részben rezidens, részben automatikusan mozog a diszkről be és ki. Az IOCS, a KBM és az USR elemei a $\emptyset$ -s memóriablokkot, az un. rendszerterületet foglalják, az SU programok felhasználói területen futnak.

A rendszermag működésének megértéséhez szükséges a file koncepció ismerete. Az OS-4000 folytonos file-kijelölő algoritmus és a szegmentált file katalógus jellemzi. A megoldások messzemenően figyelembe veszik a gép kapacitás- és sebesség adottságait.

2. A file rendszer alapjai

2.1. File fogalmak és típusok

Az OS-4000 alapegységei a file-ok. Belső felépítésük lényegtelen, a rendszer szavas ill. karakteres alapegységeket kezel.

File név és típusjelölés:

Az egyes file-okat eszköz azonosítóval, névvel és típusjelöléssel azonosítjuk. A file neve maximum 6 alfanumerikus karakter. A típusjelölés az adatok formátumát jelölheti; mindig két karakteres. A nevet és a típusjelölést egy pont karakter választja el. Az eszköz azonosítója periféria standard neve a filenévétől ':' karakter választja el. Ha elmarad, úgy a rendszerdiszk lesz feltételezve. Pl.

FORTR.FT

D1:MINTA.PL

A felhasználói programok a file-okat a "logical unit number" /LUN/ segítségével azonosítják. A LUN-t és a file szimbolikus azonosítóját a KBM ASSIGN parancsa vagy a programból hívható USR/ASSIGN funkció és az OPEN utasítás együttesen felelteti meg.

File-ok felépítése:

A file orientált eszközök 64 szavas blokkokra oszthatók. Ezek a fizikai átvitel egységei. A blokkban az első 62 szó értékes információkat /adatokat/ tartalmaz; a blokk végén ellenőrzési információk helyezkednek el.

A file-ok tartalmazhatnak:

- karaktereket
- szavas egységeket

Ennek megfelelően beszélünk

karakteres vagy bináris file típusokról.

A bináris file-ok külön osztályát képezik a memóriakép struktúrája /core image/ file-ok; melyek tulajdonképpen file-ban tárolt programok. Itt az adatokon kívül betöltési és egyéb információk is megőrzésre kerülnek.

File-ok kijelölése:

Az OS-4000 csak folytonos file-okat kezel. Új file-ok esetében; amennyiben megadjuk a file hosszát, akkor az eszközön levő "üres" helyek közül a legkisebb, file hosszát már meghaladót rendeljük hozzá a file-hoz. Ha nincs megfelelő nagyságu "lyuk", akkor a szabad területből vágunk le egy szükséges nagyságu darabot.

Nem definiált hosszúságu file-hoz a teljes szabad terület rendelődik hozzá.

Ebből következik, hogy egy eszközön egyszerre csak egy definiálatlan méretű file helyezkedhet el.

File-ok törlésekor a file neve törlődik, de a file fizikailag a helyén marad. A terület a továbbiakban szabadnak minősül; ide a rendszer új file-t kreálhat.

Összefüggő törölt területek egyesítéséről a rendszer automatikusan gondoskodik.

Tul sok kisméretű elhagyott file esetén előfordulhat, hogy új file-unk nem fér rá az eszközre vagy a katalógus van már tele, holott az üres helyeket egyesítve még lenne hely. Ilyenkor az eszközt a COPY program COMPRESS funkciójának hívásával reorganizálni kell. /A hívás a KBM-on keresztül történhet./

2.2. A File Directory

Bármely file orientált eszközön a rajta elhelyezkedő file-okat az eszköz 1.-6. blokkjain található file katalógus /FD/ tartja nyilván.

Az FD olvasásának ill. írásának időigényét az FD szervezésével csökkentjük.

Az FD minden egyes blokkja egy alkatalógust, egy ún. szegmenst tartalmaz. A szegmens felépítése:

A szegmensben elhelyezkedő bejegyzések száma	}	szegmensfej /1 szó/
Az első szegmensbeli file kezdő blokkszáma		
mutató a köv. szegmensre 0; ha nincs, 1; ha van		
file bejegyzés		
file bejegyzés		
...		
file bejegyzés		
tartalék hely /1 bejegyzésnyi/		

A tartalék hely azt jelenti, hogy katalógusfeltöltéskor az utolsóelőtti bejegyzés-helynél történik a szegmensváltás.

A file-bejegyzések

Létező file-oknál:

file neve	}	2.szó
tipusjelzése		
kreálási dátum	}	3.szó
end-of-file mutató /blokk, szó, kar./		
file hossza blokkban		
	}	4.szó

Üres file-nál:

xx	}	1.szó
file hossza blokkban		

Új file-oknál, amennyiben nincs hossz-adat; egy $\emptyset$ hosszúságú üres file is szerepel. Ez csak az FD utolsó eleme lehet.

2.3. A file-ok felépítése

Karakteres és bináris adatfile-ok

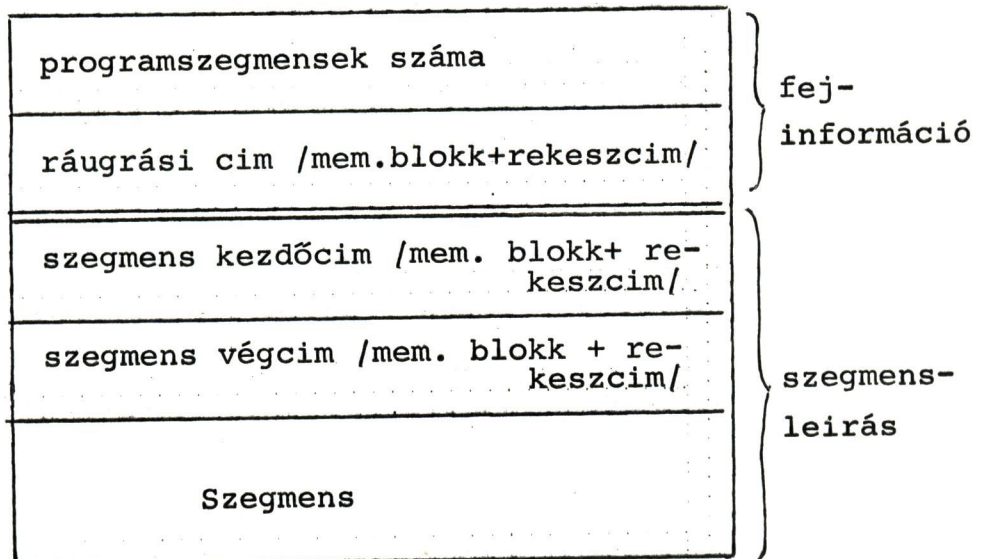
Az adatterületen csak adatok szerepelnek /+ a kontrollszummák/. Ténylegesen kitett EOF jel nincs; /pozicionális

figyelés van, azaz az EOF helyét figyeli a FCS/.

Core image file-ok

Az adatterület felépítése:

1. blokk:



A szegmensleírás minden szegmensre megismétlődik.

A core image file-okat GET, RUN, SAVE monitor parancsokkal, a CHAIN USR funkcióval ill. a COPY programmal mozgathatjuk. Közvetlen I/O a felhasználói programokból általában nem adódik ki core image file-okra.

2.4. Directory kezelés:

A directory-ből elhagyás történik:

close delete -nél; /FCS utasítás/

delete-nél; /KBM parancs ill. USR funkció/

Elhagyáskor a bejegyzés törlődik, s helyére üres file bejegyzése kerül. Ha előtte vagy utána üres file volt, azzal összevonódik. A bejegyzés utáni szavak áttöltődnek úgy, hogy az FD-szegmens mindig folyamatos.

Szegmens határán lévő empty file-ok csak a COMPRESS-nél és un. "átgyűrűző" beillesztésnél vonódnak össze /azaz szűnnek meg!/.

A directory bővül, ha:

"open" FCS utasítás hajtódik végre, WRITE-ra.

Megkeresi a megfelelő üres file-t, s szétnyomja a directory-t, amennyiben az üres file bejegyzése helyére egy tényleges bejegyzést és egy üres file-t tesz. Az üres file $\emptyset$ hosszal elmarad, ha nem a legutolsó file-ra vonatkozik.

A szétnyomás csak akkor gyűrűzik tovább több szegmensen, ha a szegmens tulcsordul. Ilyenkor sem valószínű, hogy több szegmensen átgyűrűzik.

3. Perifériakezelés

Három szinten történik.

File control system /FCS/

felhasználói szintű IO + ellenőrzések biztosítása

Device handler-ek:

Az FCS által kért, konkrétan paraméterezett IO igények kielégítése.

Driver-ek:

Elemi szintű eszközmozgató rutinok.

3.1. File Control System

Feladata:

Kapcsolatteremtés a felhasználói program felé.

A felhasználó az FCS funkcióit szubrutin hívásokkal aktivizálja.

Funkciók:

a./ File megnyitás

Paraméterek: LUN

file DCB kezdőcíme /blokk+rekesz-
cim/

tipus

hozzáférés módja

hossz /blokkban/

file-név

Algoritmus:

A Keyboard Monitor ASSIGN parancsai ill. az automatikus assign-ok révén adott egy tábla arról, hogy az illető LUN-ok milyen eszközre vonatkoznak.

ASSIGN SD1 2

A hozzárendeléssel minden LUN-nak egy perifériát feleltetünk meg.

A hozzárendelési tábla: /LUB/

Eszköz- pointer

A LUN-t a táblaelem sorszáma adja. /Vagyis a LUN-t érdemes folyamatosan kiadni, mert maximálva van!/.

A programokban LUN-t, azaz számokat használunk a perifériák /file-ok/ mozgatásánál.

Nyitáskor az FCS a hozzárendelési tábla alapján megállapítja, milyen eszközről van szó és ennek megfelelően jelöli ki a végrehajtandó FCS funkciókat.

Diszken: Read és update esetén File Directory /FD/ keresés. Hibajelzés, ha nincs. WRITE esetén kreálás. Létezés esetén hibajelzés. Egy, a file-hoz hozzárendelt file control blokk /FBC/-be /többek között/ a következő információk kerülnek:

file kezdete	/blokksorszám/
hossza	/blokkok száma - szavak száma/
EOF helye	/blokk, szó, kar./
file típusa:	/bin/char/
hozzáférés módja	/R/W vagy update/

Ezen kívüli FCB tartalmakat ld. egy későbbi pontban!

b./ File írás/olvasás

Paraméterek: LUN, $\left\{ \begin{array}{l} \text{hová} \\ \text{honnan} \end{array} \right\}$, mennyit

- ellenőrzi az I/O jogosságát
/mire van nyitva/
- ellenőrzi a limit túllépést
- a tényleges I/O lebonyolításához továbbadja a szükséges információkat az aktuális device handlernek. /Pl. a Buffer Control Block címét!/.

c./ Pozicionálás

find /lun, pozíció/

A pozícióban kijelölt szóhoz ill. karakterhez állítja a handler aktuális mutatóit és a handler bufferbe betölti a megfelelő blokkot.

d./ File close

A file directory-t update-eli /diszk esetén/, esetleges lezárási tevékenységeket /EOF írás/ végez.

Adatstruktúra

A File Control Block tartalma

file neve	- assign
tipusa	- open par.
eszköz tipusa	- assign
lun	- assign
file kezdet	- open veszi a dir.-ből
hossza	- " -
EOF helye	- " -
mire van nyitva	- open par.
BCB címe	- automatikusan képződik

A file control block-ot, a BCB-t és a handler bufferét a felhasználó jelöli ki a központi memóriában. Ez az ún. Data Control Block /DCB/ a felhasználótól függetlenül mozog; de ellenőrzési célokból bármikor ki lehet szedni belőle az aktuális állapotokra vonatkozó információkat.

A DCB mérete fix.

3.2. Device handler

Feladata:

Kapcsolat teremtése az FCS és az eszközdrivererek között; blokkolás, deblokkolás elvégzése; szükség esetén fizikai I/O végrehajtása driveren keresztül.

Funkciói

a./ A Buffer Control Block kezelése.

Itt helyezkednek el a fizikailag egyszerre átvitt információmennyiség és az annak kezeléséhez szükséges adatok. /ld. később/.

b./ A kívánt számú szó/kar. mozgatása a felhasználói buffer és a belső buffer között.

c./ Ha a belső buffer üres /Read-nél/
vagy tele /Write-nál/
vagy a blokkszám változott az
utolsó híváshoz képest, akkor
a driver hívása.

d./ Ha a kívánt számú szó/kar. mozgatása közben end-of-record-ot észlel, akkor ez felülbírálja a hossz-megadást, és az I/O befejeződik.

Adatszerkezet:

A BCB /Buffer Control Blokk/:

a legutolsó híváskor mozgatót blokk száma
belső buffer címe
- " - hossza
offset pointer blokkon belül /szó+kar/
még fel nem használt { karakterek szavak } száma
szó/char. típus

/binárisnál szavak; karakteresnél karakterek mozognak/

Paraméterek:

- felhasználói buffer címe /memo-blokkszám+cim/
- direkt blokkos eléréskor a blokk sorszáma,+
+ az offset száma, egyébként ? jel
- hányat!

Példa:

Egy szekvenciális file olvasása a lemezeiről.

A file neve: X.PA

Rekordhossza: 10 szó

A file bináris!

Volt egy ASSIGN DSK1 2

```
open (2, bináris, read,X.PA,XBUF)
1 : read (2,ibuf, 1, 10)
    to eof, 0
    feldolgozás
    to 1: , 0
```

Az open hatására

- 1./ Melyik eszközön van? - diszken!
ellenőrzés; lehet-e bin.
és read
- 2./ LOOKUP bejön a directory-ból a blokkszám,
 ;
 EOF mutató
- 3./ FCB feltöltés
- 4./ BCB hozzárendelés ; feltöltés kezdőértékekkel

A read hatására

- 1./ Ellenőrzi, szabad-e olvasni.
- 2./ Megnézi /a BCB-hez nyulva/, van-e még elég hely. /Nem megyünk-e idegen file-ba!/.
3./ EOF vizsgálat.
- 4./ Indítja a device handler-t; ami behoz 10 szót az 1.memóriablokk ibuf-tól kezdődő szavaira.

Ha EOF; azt az FCS veszi észre /tulolvasás!/ és az eof kimeneten küldi vissza a vezérlést.

A rekordvéget a device handler-ek érzékelik
/karakteres feldolgozásnál/.

3.3. Device driver-ek

A fizikai szintű I/O-ot végző rutinok.

Paraméterek:

- melyik blokkból
- hová kell olvasni

A blokkszámnak jel-orientált perifériáknál nincs
értelme.

3.4. Device és file táblák

A perifériák jellemzőit ill. a logikai periféria-
kezelést az alábbi rezidens táblák írják le.

a./ Physical Unit Block /PUB/

A standard periférianévek és fizikai megjelölé-
sek táblázata.

csatornaszám	periférianév	mutató a periféria- handleres táblájába
--------------	--------------	--

b./ Handler információs tábla

handler elhelyezkedése a diszken	handler kcime ($\emptyset$, ha nem rezidens)
--	--

c./ Perifériaajellemzők táblája

file-struk- turált, vagy nem	read only write only	bin. v. karakteres
------------------------------------	-------------------------	-----------------------

b./ és c./ összevonható

d./ ASSIGN tábla

pointer a PUB-ra

A lun. az elem sorszáma

A LUB, PUB, ASSGN stb. táblák nem fix hosszúságú táblák. A konkrét méreteket egy, a generáláskor készülő tábla /SYSCOM/ tartalmazza.

A SYSCOM helye és mérete fix.

e./ SYSCOM tábla

LUB kezdőcíme	
PUB kezdőcíme	
perifériatábla kezdőcíme	
ASSGN kezdőcíme	mérete
dátum	

4. Keyboard Monitor

Bevezetés

A Keyboard Monitor a PC 4000-es számítógépre készítette operációs rendszer kezelői funkcióit végzi. A számítógép bekapcsolása után a bootstrap eljárás segítségével az operációs rendszer rezidens része betöltődik a 0-as blokkba és a consolon bejelentkezik a Keyboard Monitor. A Keyboard Monitor a consolról parancsokat fogad megteremtve a kapcsolatot a felhasználó és az operációs rendszer ill. felhasználó rutinjai között.

4.1. A Keyboard Monitor globális leírása

A Keyboard Monitor a PC 4000-es számítógépre írt operációs rendszer kezelői funkcióinak elvégzésére képes. A consolról parancsokat fogad el, megteremtve ezzel a kapcsolatot a felhasználó és az operációs rendszer ill. a felhasználó rutinjai között. Lehetővé teszi, hogy a felhasználó logikai számokat rendelhessen az egyes perifériákhoz, futtathasson saját és rendszerprogramokat, tárolhasson programokat, valamint hívhassa a debug kezelő rendszert.

A Keyboard Monitor a bootstrap eljárás befejezésekor a consolon a következő sor bal szélére > jelet gépel és várja a felhasználó parancsát.

A parancsok a következők lehetnek

ASGN	par. lista	/logikai számot rendel szabvány perifériához/
GET	- " -	/tárba tölt egy core image file-t/
START	- " -	/futtatás indítása/
SAVE	- " -	/memória részek kimentése core image file-ba/
HEDT	- " -	/futtatás közbeni debug/
DATE	- " -	/dátum bevitele az op. rendszerbe/
RUN	- " -	/betöltés és indítás/

Amennyiben a felhasználó nem a fenti parancsok egyikét adja, úgy azt file-névnek értelmezi, és megkeresi a File Directory-ban. Ha megtalálta, akkor betölti a tárba és elindítja. Ha egyik eset sem áll fenn a parancs megismétlődik és utána megjelenik egy ? jel.

A parancsok szintaktikája kötött. Általános alakja a következő

> parancsszó 1 üres köz összefüggő paraméter 1 üres stb.
vagy

> filenév.extension
ami egyenértékű a

> RUN filenév.extension paranccsal.

/Itt az extension elhagyása esetén .CI a feltételezés/.

A fenti parancsoknak megfelelően a Keyboard Monitor felépítése a következő részekre bomlik:

a./ a consolon beírt parancs értelmezése, hiba esetén visszajelzés;

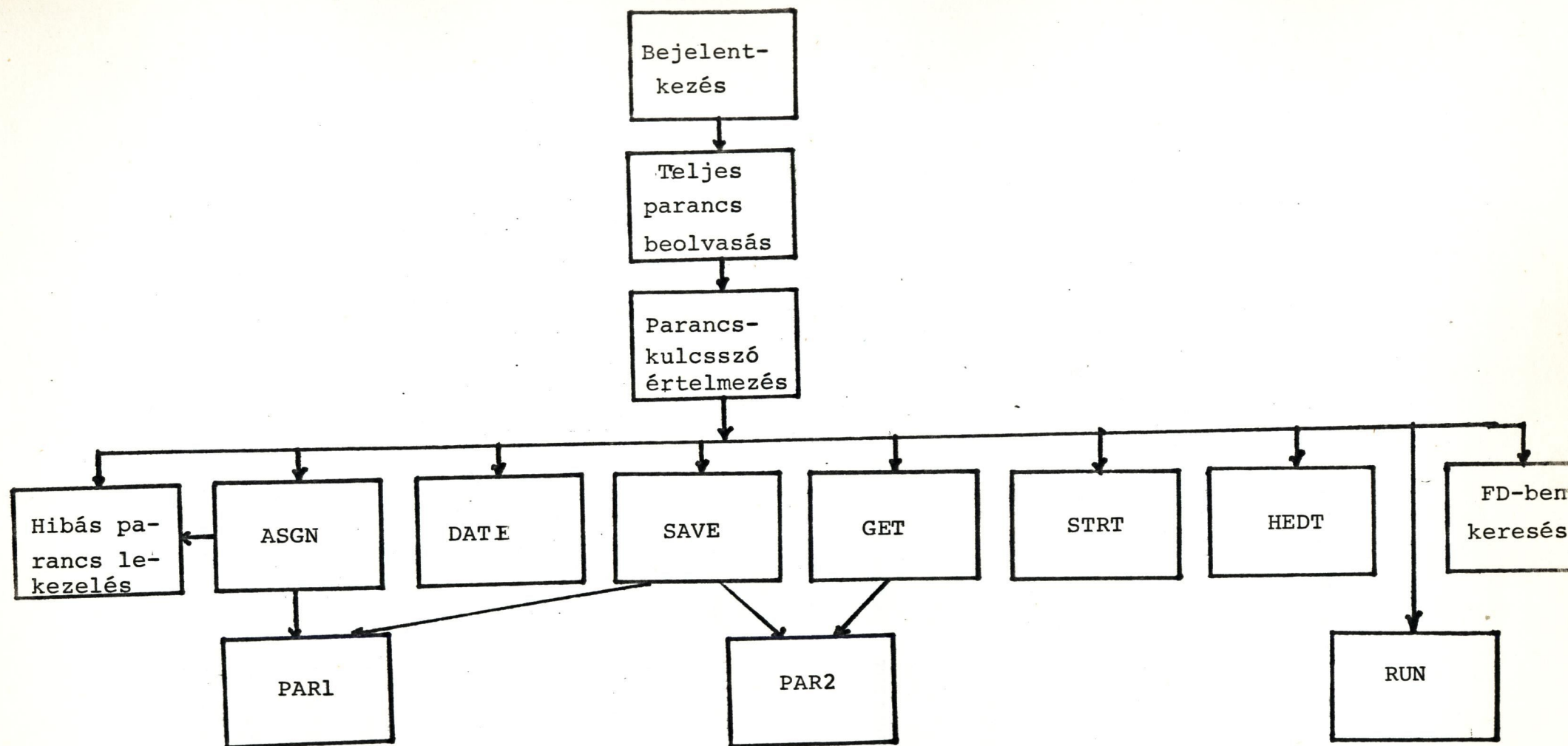
b./ az egyes parancsok feldolgozása.

A parancs értelmezés után a Keyboard Monitor értelmezi a paramétereket, végrehajtja a szintaktikus ellenőrzéseket a paraméterekre vonatkozóan és hibátlan paraméterlista esetén módosításokat hajt végre a rendszertáblákon majd végrehajtja a felhasználó által meghatározott parancsot.

Ennek megfelelően a Keyboard Monitor modulszerkezete a következő oldalon található.

A Keyboard Monitor tevékenysége során a következő rendszertáblázat és directory-kat használja.

- 1./ PUB /Physical Unit Block/- az ASGN parancs feldolgozáshoz
- 2./ ASSIGN tábla /LUB/ - az ASGN parancshoz
- 3./ FCB, BCB táblák core image file-ok kezeléséhez
/GET, SAVE, RUN/ FCS-en keresztül!



Modulszerkezet tábla

- 4./ Futáscím /1 szó/ /MCB-ben/ - a futás elindításához /JUMP, GET, RUN/
- 5./ SYSCOM tábla - amelybe a DATE paranccsal bevisz-
szük az aktuális dátumot/
- 6./ Parancstábla 7 szó, amely a Keyboard Monitor
parancsait tartalmazza.

ASGN - assign parancs

GET - core image file beolvasása

SAVE - core image file létrehozás

HEDT - debug

DATE - dátum bevitel

STRT - program végrehajtás

RUN - betöltés és futtatás

A rendszer fontosabb változói:

inp - tömb a consol puffer-ja

parancs - 1 szó a parancs kódja, ha $\emptyset$ hibás a
parancsszó

hibakód - a hibaüzenet kódszámát tartalmazó egész
szó

posi - inp. puffer pozíciója

paraméter- 1 szó a leválasztott paraméter

filenév - 2 szó a filenév.extension formátumozott
tárolására

4.2. A Keyboard Monitor moduljainak leírása

Az alábbiakban az előző fejezet modul szerkezet ábrájának elemeit írjuk le rendszerterv szinten.

Modul neve: Bejelentkezés

Funkciója: Soremelés a consolon és az első pozícióra a > jel kiírása

Algoritmus: triviális

Speciális adatok: a soremelés kódja és a '>' jel kódja

Modul neve: Teljes parancs beolvasás

Funkciója: A felhasználó consolra irt parancsának tárbá töltése

Algoritmus: consol handler

Speciális adatai: Az inp nevű 15 szóból álló tömb, amely a beolvasott parancsot tartalmazza. Inpmax, amely az inp tömb pointere.

Modul neve: Parancskulcsszó értelmezés

Funkciója: A felhasználó által meghatározott parancs értelmezése.

Algoritmus: Az inp tömb 1. szavát ciklikusan összehasonlitja a parancstábla elemeivel. Az összehasonlitás során feltölt egy parancs szót, amely tartalma a

következő

∅ - ha nem parancs érkezett,

i - ha az i-ik parancs érkezett,

majd ugrik-e megfelelő modulra.

Speciális adatai:

input_adatok: Inp(1)

output_adatok: parancs, hibakód,

belső_adatok: parancstábla, hibakód=1

Modul neve: Hibás parancs lekezelés

Funkciója: A hibakód tartalmának soremelés után történő kiírása.

Algoritmus: A modul soremelést hajt végre a consolon, kiírja a hibakód tartalmát majd ugrik a 'Bejelentkezés' modulra.

Speciális adatai:

input_adatok hibakód

output_adatok -

belső_adatok -

Modulnév: ASGN

Funkciója: Az ASGN parancs paramétereinek szintaktikus ellenőrzése. Az ASGN parancs formája a következő:

> ASGN_dev_lun

ahol

dev: standard periféria azonosító /lsd.Periféria azonosító tábla/

lun hexadecimális (?) max 1 jegyű szám.

A periféria logikai perifériaszáma.

Algoritmus: A PAR1 nevű egyszavas paraméter leválasztó rutin kétszeri meghívásával leválasztja a két paramétert. Az 1. paraméter tartalmát összeveti a P és B tábla 1. oszlopával. Ha nem találja, hívja a 'Hibás parancs lekezelés' rutint a megfelelő hibakóddal (2).

Megnézi szám-e a 2. paraméter (3-as hiba).

Összehasonlitja a 2. paraméter tartalmát a Periféria azonosító 2. oszlopával.

Ha a parancs hibátlan, a 2. paraméter tartalmát beviszi az ABSIGN tábla 1. oszlopának megfelelő sorába, majd ugrik a 'Bejelentkezés' modulra.

Speciális adatok

input_adatok: inp

output_adatok: -

belső_adatok: Periféria azonosító tábla, hibakód

Modul neve: PAR1

Funkciója: Maximum négy karakter elhelyezése egy szóba.

Algoritmus: Az adott puffer tömb megadott pozíciójától kezdődően blankig/4 karakterig betölti a karaktereket egy szóba.

Speciális adatok:

input_adatok: inp, posi
output_adatok: Paraméter
belső adatok: -

Modul neve: DATE

Funkciója: Dátum rendszerbe vitele. A DATE parancs formátuma

> DATE_éé-hh-nn

Algoritmus: A dátum átírása a dátumtáblába, ugyanilyen sorrendben balszélre igazítva. Két parl hívással megoldható. Ellenőrzés van! Utána visszatér a 'Bejelentkezés' modulra.

Speciális adatok:

input_adatok: inp
output_adatok: datumtábla
belső adatok: -

Modul neve: HEDT

Funkciója: A HEDT parancs lehetővé teszi, hogy a felhasználó a consolon gépelt utasítások segítségével

futtathassa a programját, ellenőrizhesse a működését és a programot módosíthassa is.

Algoritmus. A HEDT algoritmus megegyezik a Simulator algoritmusának az algoritmusával. Az egyszerűsítés az, hogy a mi esetünkben csupán egy breakpoint-ot engedünk beállítani.

A HEDT parancs futtatására a HEDT eljárás betöltődik a memóriába és inputot vár a consolról.

Speciális adatok:

input_adatok: HEDT opciók
output_adatok: HEDT válasz
belső adatok: -

Modul neve: GET

Funkciója: Egy adott device-ről /diszk, floppy diszk/ core image file tárbá töltése. A parancs formátuma

>GET_lun_filenév.ex

filenév: max 6 karakteres file-név

ex: extension. Célszerű core image file esetén a CI-t választani /ha elhagyjuk, ez a feltételezés/

Algoritmus: A PAR2 modul segítségével a filenevet és extens.-t két szóban helyezi balra igazítva

filenév extension

6 ch	2 ch
------	------

Ha a lun-hoz rendelt dev 1. karaktere $\neq$ D vagy F /hibakód=9/. A look up USR szintű hívással megkeresi az adott device-n a file-t a directory-val a memória kontroll blokkba helyezi.

Elindít egy 'szegmensek száma' lépésű ciklust, amelyben a következő tevékenységeket végzi:

- beolvassa a blokkazonosító-kezdőcim-végcim szóhármast az i.-ik szegmens elejéről
- ellenőrzi, hogy a kezdőcim protected terület-e. $\emptyset$ -s blokk esetén /Rezidens tábla/.
- betölti a szegmenst a kezdőcimtől a végcímig

Minthogy a core image file a SAVE parancs hatására épült fel hiba nem lehet.

A betöltés után visszatér a 'Bejelentkezés' modulhoz.

Speciális adatok:

input adatok: inp

output adatok: -

belső adatok: filenév /2 szó/

Modulnév: STRT

Funkciója: A GET paranccsal tárba olvasott program indítása

Algoritmus: A memória Control táblából kiemeli a futás kezdőcímét és oda ugrik.

Speciális adatok:

input_adatok: -

output_adatok: -

belső_adatok: memória control tábla

Modulnév: SAVE

Funkciója: A tár nem rezidens, a felhasználó által definiált részének core image file formátumu tárolása diszken vagy floppy diszken.

Az utasítás formátuma a következő:

>SAVE_dev_filenév.extension_blokkszám ^{max:}

szegmensek száma: hexa szám

memória: 1 jegyű szám ($\emptyset$,1)

kezdőcim: 3 jegyű hexa szám

végcim: 3 jegyű hexa szám

futás kezdőcime: - " -

Algoritmus: Az algoritmus lépései:

1./ Paraméterek leválasztása /paraméter, filenév, blokkszám/ /PAR1, PAR2/

- 2./ USR create funkciójának hívása /OPEN/
- 3./ 'memória:' kiírása a válasz ellenőrzése
/hibakód=5/
- 4./ 'kezdőcim:' kiírása, válasz beolvasása
- protection ellenőrzés ϕ -s blokk esetén /rezidens tábla/
/hibakód=6/
- 5./ 'végcim:' kiírása a beolvasott válasz összeve-
tése a kezdőcim-mel /hibakód=7/
- 6./ kiírás a core image file-ra
 - blokkszám, kezdőcim, végcim
 - a szegmens kiírása
 - a memória controll blokkban a szegmensek
száma 1-el nő.

- ugrás 3-ra

Minden 3-5-ben előforduló hibaüzenet után 3-re
való visszaugrás történik.

- 7./ 'futás kezdőcime:' kiírása, válasz beolvasása
és a memória controll táblába töltése
- 8./ 'szegmensek száma:' kiírása, válasz beolvasása
- 9./ az FCS close funkciójának hívásával a file le-
zárása és ugrás a 'Bejelentkezés' modulra.

Speciális adatok

input adatok: paraméter, filenév, blokkszám

output_adatok: hibakód

belső adatok: memória controll blokk, rezidens
tábla

Modulnév: PAR2

Funkciója: Az inp puffer adott pozíciójától kezdve
a file-név.extension leválasztása a filenévre előírt
formátum szerint.

Algoritmus: Az inp adott pozíciójától kezdődően
max. 6 karakterig bezárólag '.' jelet keres. Ha nincs
/hibakód=8/. A karaktereket a filenév szóba helyezi a meg-
felelő formátum szerint.

A tizedespont utáni 2 karaktert elhelyezi a file-név
2. szavának hátsó 2 karakterébe.

Speciális adatok:

input adatok: inp, posi

output_adatok: filenév

belső adatok: hibakód

Modulnév: RUN

Funkciója: A GET és STRT parancsok együttes végre-
hajtása

Az utasítás formátuma:

>RUN_dev_filenév.ex.

dev: lun

filenév: max 6 karakter file-nev

ex: extension, max 2 kar

Algoritmus: Meghívja a GET majd az STRT parancsokat.

Speciális adatok:

input adatok: inp

output adatok: -

belső adatok: -

Modulnév: FD keresés

Funkciója: Ha a parancsszó = $\emptyset$ akkor meghívódik a rutin és megnézi, hogy a beírt karaktersorozat nem file-név-e. Ha igen a BCB táblából kiveszi a file-hoz tartozó lun-t és elindítja a RUN parancsot. Ha nem, hibaüzenet /1/.

Algoritmus: -

Speciális adatok:

input adatok: inp

output adatok: -

belső adatok: -

4.3. Kezelési rend

Mielőtt a Keyboard Monitor kezelési rendjére rátérnénk megjegyezzük, hogy a felhasználónak olyan programot kell írnia, amely végén egy ugrás történik a Keyboard Monitor 'Bejelentkezés' nevű eljárására. Így elérhetjük, hogy a felhasználó mindig a Monitorba térjen vissza.

A BOOTSTRAP eljárás végrehajtása után a Monitor automatikusan indul és a > jellel bejelentkezik a consolon. A felhasználó által beírt parancsokat a paraméterekkel együtt a RETURN billentyű leütésével kell elküldeni. Ha a parancs vagy bármelyik paramétere hibás, akkor a MONITOR egy hibakódot ír ki, amely a hibára utal. A lehetséges hibakódokat a jelentésükkel együtt tartalmazza a Monitor hibakód táblázata.

Monitor hibakód táblázat.

- 1 - hibás parancsmegadás
- 2 - hibás periféria azonosító
- 3 - a logikai periféria azonosító nem szám
- 5 - helytelen blokkszám
- 6 - a cím védett
- 7 - a végcím a kezdőcímnél kisebb
- 8 - túl hosszú filenév
- 9 - hibás device /D vagy F lehet/

Minden parancs formátuma kötött. Minden esetben igaz az, hogy a parancsot és a paramétereket 1 üres köz-
zel kell elválasztani.

A Monitor parancskészleteit és formátumát az előző
fejezet modul leírásainak funkcióinál leírtuk, most ezt
foglaljuk össze.

ASGN_dev_lun

Paraméterei:

dev: standard periféria azonosító /lsd Perifé-
ria azonosító tábla/

lun: logikai periféria szám LUB /hexadecimá-
lis/

Jelentése: Logikai perifériaszám rendelés [$\emptyset$,1 $\emptyset$]
a standard perifériákhoz.

DATE_éé-hh-nn

Paramétere:

éé-hh-nn : napi dátum

Jelentése: napi dátum operációs rendszerbe helyezése

HEDT

Jelentése: A debug kezelő eljárás elindítása.

Első bejelentkezés után a STRT paranccsal a
programunkat el kell indítani. További bejelent-
kezés után a program futása automatikus.

GET_dev_filenév.extension

Paraméterei:

dev: logikai /standard disk/ floppy diszk
periféria azonosító

filenév: max 6 ch-es név

. : kiírása kötelező, előtte utána nem
lehet blank

extension: max. 2 karakter, itt CI használata
ajánlott

Jelentése: Egy core image file tartalmának tárba
vitele. Rezidens területre történő töltés
esetén a file nem töltődik.

STRT

Jelentése: A betöltött program indítása.

SAVE_dev_filenév.extension_max.blokkszám

Paraméterei:

Az első 3 paramétert lsd GET

max blokkszám: max 3 jegyű decimális szám
a file számára lefoglalandó
terület 64 szavas egységekben.

.MEMORIA:_par

Paraméterei:

par: 1 jegyű szám

Jelentése: a kiírandó szegmens memória blokkja

.KEZDŐCIM: _par

Paramétere:

par: max 3 jegyű hexa szám

Jelentése: a memoria kiírást ettől a címtől kell kezdeni.

.VEGCIM: _par

Paramétere:

par: max 3 jegyű hexa szám

Jelentése: a memória kimentés eddig a címig történik. Ha a kezdőcím helytelenül lett megadva, itt annál kisebb szám beütése hibaüzenet után a .MEMORIA utasításra való visszalépést eredményezi, így a kezdőcím újra beállítható.

.FUTAS KEZDOCIME: _par

Paramétere:

par: max 3 jegyű hexa szám

Jelentése: A core image file kiírása után a GET és STRT parancsokkal a visszatöltött program végrehajtása ezen a címen kezdődik.

.SZEGMENSEK SZAMA: _par

Paramétere:

par: max 3 jegyű hexa szám

Jelentése: A core image file-ba töltendő önálló, folytonos programegységek száma.

RUN_dev_filenév.extension

Paraméterei: lsd GET

Jelentése: GET és STRT együttes végrehajtása.

4.4. Tesztelés és implementálás

A Monitor tesztelésénél fel kell használni a Simulator lehetőségeit. A tesztelés 1. fázisában az USSR rutinok felhasználása nélkül az egyes parancsokat elemenként kell tesztelni. A HEDT elkészítése után a többi parancs akár a HEDT segédlettel is tesztelhető a Simulator alatt. Ez azért célszerű, mert így a HEDT-t is párhuzamosan, alaposan újra tesztelhetjük.

Az egyes parancsok tesztelési rendje igen egyszerű:

- összes hibaüzenet kipróbálása
- helyes parancsmegadás és a megfelelő rendszer-tábla ellenőrzése.

A tesztelés 2. fázisában be kell építeni a kész USSR rutinokat és a consol ill. device handler-t. Ezután a tesztelés menete ugyanaz kell legyen, mint az 1. fázisban.

A PC 4000-es gépre való implementálás igen egyszerű. Minthogy ez a BOOTSTRAP feladata, ezért a következő tevékenységeket elég elvégezni:

- A Monitor object kódját be kell írni a tárba.
- A tár tartalmát ki kell menteni a $\emptyset$ -s diszk rendszer file-ok számára fenntartott területre. /GEN/

6. System utilities

6.1. INIT

Mágneselem inicializáló program

Egy üres mágneslemezre felviszi a file-szervezettséghez szükséges strukturákat. Az INIT-et a BUILD program hívja a rendszer építésekor; egyébként önállóan futtatandó, minden új lemez felhasználása előtt.

Globális leírás

Funkciója: A mágneslemezekre egy VOLUME LABEL-t és egy üres file-t tartalmazó directory-t rá kell írni.

A VOLUME LABEL formátuma:

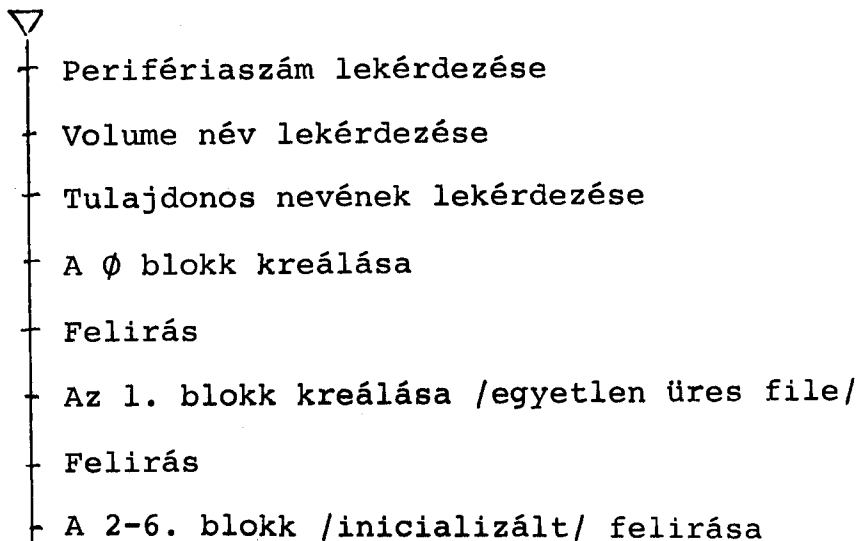
A $\emptyset$ blokk-ban helyezkedik el.

Volume név: 6 karakter

Tulajdonos neve: 18 karakter, /ha a volume név
nem SYSTEM/!

Megnyitás dátuma: 1 szó

Algoritmus:



Kezelés:

A párbeszéd a konzolról történik.

A párbeszéd menete:

periféria: $\left\{ \begin{array}{l} D\emptyset = \text{SYS} \\ D1 \\ F\emptyset \\ F1 \end{array} \right\}$

Volume id.:

Tulajdonos:

OK.

A program meghívása a

RUN INIT

6.2. GEN

A GEN program feladata a $D\emptyset$:lemezegységre előállítani egy futóképes, kezdő rendszert, amely alatt a BUILD program segítségével már a felhasználó a saját igényeinek megfelelő rezidenst tud készíteni.

A HW lehetőségek /nagy-tömegű input hordozója papírszalag lehet, a $D\emptyset.D1$ Mom diszk nem cserélhető/ következtében a teljes GEN program egy lyukszalagon elhelyezkedő, önbetöltő, stand-alone program lesz. Tartalmaznia kell

/feltehetőleg ebben a sorrendben/

- Boot betöltő program
- Init program /stand-alone változata/
- Boot l-t Vol 0-ba író program
- COPY program stand-alone, speciális verziója, mely lyukszalag → lemez másolást végez, a normál file-kezelő rendszer felhasználásával /szimulálásával?/. A lemezen a file-directory minden egyes file felvitele után módosítódik és az aktuális állapotot tartalmazza. A másolandó file-ok között célszerű teszt-pontokat beépíteni /amelyekre pozicionálni tudunk/, így esetleges HW hiba esetén nem kell a teljes betöltő futást megismételni.
- A PCSYS.00 tárhoz töltése, normál BOOT, amelynek hatására a KBM bejelentkezik, s a rendszer működőképes. /BUILD futtatható/.

Megjegyzés: 1. A program nem ellenőrzi a D0: korábbi tartalmát; az tehát megsemmisül.

2. A GEN szalag előállítása a korábbi /jelenleg élő/ programok segítségével történik, s mivel tartalmazza az új OS minden elemét, tehát végső formájában azok elkészülte után alakítható ki.

6.3. BUILD program

6.3.1. Bevezetés

A BUILD program KBM paranccsal indítható, futó rendszer alatt. A BUILD által feltett kérdésekre adott válaszok alapján elkészíti a kívánt 'nn' verziószámu PCSYS.nn fílet, és azt a lemezen tárolja. Ez a program teszi lehetővé, hogy a felhasználó az induló /gyártó által szállított/ PCSYS.00 operációs rendszer segítségével, az alatt újabb, a saját igényeinek, perifériáinak megfelelő rezidenst állítson elő. Meghatározhatja a tárrezidens handlereit és programjait is.

6.3.2. A BUILD program részletes leírása

Funkciója:

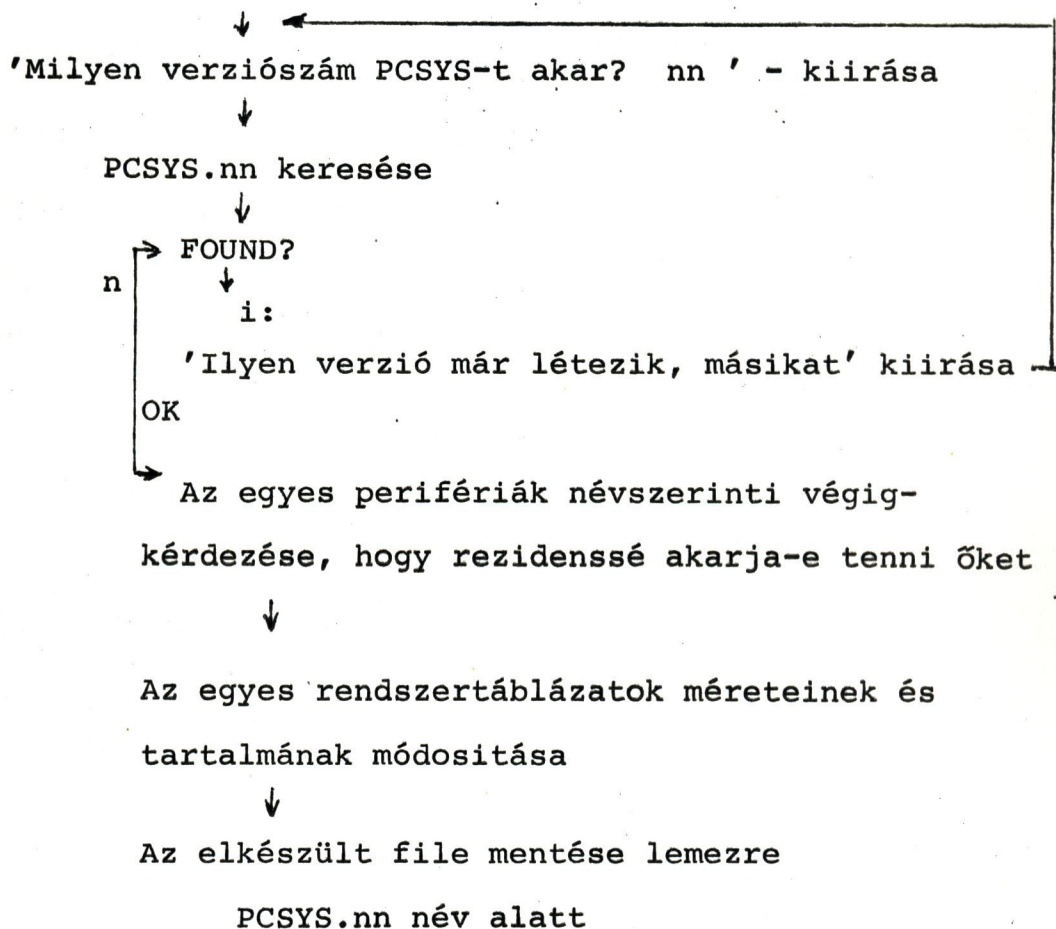
Eszközt ad a felhasználó kezébe új OS rezidens program előállítására. A starter rendszer felhasználásával (.00) ettől eltérő verziókat állíthat elő, esetlegesen speciális felhasználások részére többfélét is. /Ezek közül bármelyik indítható a rendszer betöltésekor./ Az egyes rezidens rendszerrészek különböző perifériahandlereket tehetnek tárrezidenssé, meghatározhatják a tárba már rendszerindításkor bekerülő programokat. Mindez interaktív módon történik, s az eredményként előálló file PCSYS.nn néven kerül tárolásra a rendszerlemezen.

/'nn' értékét szintén a felhasználó specifikálhatja./

A BUILD program a KBM RUN parancsával hívható.

Nem pótolhatja tehát a stand-alone GEN programot, amely lyukszalagon tartalmazza a teljes induló operációs rendszert, annak minden komponensével, s lemezre tölti azt.

Algoritmus



Speciális adatai:

Az operációs rendszer rezidens részének felépítését, adatait és ezek tartalmát tárolja, ill. ezen információkhoz hozzáfér.

6.3.3. Kezelési rend

Egyszerű KBM paranccsal hívható. Az interaktív működés lehetővé teszi, hogy minden adat hibátlanul kerüljön feldolgozásra. Futás után ismét a KBM kapja a vezérlést.

6.3.4. Tesztelés és implementálás

Az előállított file a file directory-ba kerül /a file rendszer karbantartó program-kommunikációt szimulálni kell/.

A keletkezett file használhatóságát a BOOT program ellenőrzi /ennek képezi inputját/, tehát végső tesztelésére csak annak elkészülte után kerülhet sor.

6.4. BOOT/IPL program

6.4.1. Bevezetés

A BOOT program a rendszer betöltésére szolgál; a rendszer legelőször működő egysége. Igényli a gép HW-ben megvalósított betöltési lehetőségét /papírszalagról/, majd a második fázisban lemezzől veszi további részeit. Saját lemezolvasói /file-kezelői részeket tartalmaz.

Feladatát képezi az operációs rendszer indítása a gép bekapcsolása után. Eredményeként az operációs

rendszer rezidens része, a rezidens eszközzelők programok bekerülnek a tárba, s a vezérlést a monitor /a KBM/ kapja.

6.4.2. A BOOT program részletes leírása

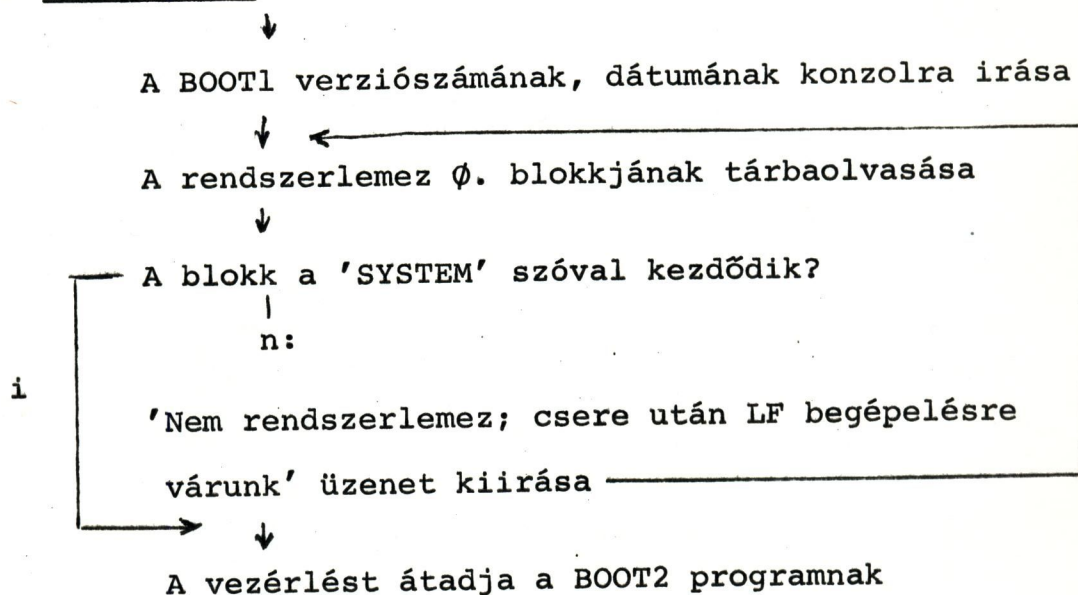
A./ BOOTØ,

HW lehetőség, amely a gép standard részét képezi. Segítségével válik lehetővé a kezdő betöltő program /BOOT1/ memóriába juttatása. Jelenleg ez az adottság csak papírszalagos perifériára létezik, tehát a BOOT1 csak egy lyukszalag lehet.

B./ BOOT1

Funkciója: A rendszerlemezen elhelyezkedő BOOT2 program tárba töltése. Ezáltal válik lehetővé, hogy a BOOT2 programot lemezen tároljuk - BOOTØ ugyanis csak papírszalagot képes kezelni. Mivel a BOOT2 csak egyetlen /?/ lemezblokkból áll, a program feladata mindössze ennek az egy blokknak /#Ø/ a tárbatöltése és a vezérlés átadása.

Algoritmus:



Speciális adatok: nincsenek, stand-alone program, más rendszer elemekkel nem kommunikál.

C./ BOOT2

Funkciója: Az OS rezidens és a szükséges periféria handlerek/driverok betöltése a tárba, s a normál, folyamatos üzemmód megindítása, amely a KBM indítását jelenti. Eredményként a rendszer üzemképes és felhasználói parancsra vár. Lehetőséget ad többféle OS rezidens közötti válogatásra, filenév kiterjesztéstől függően, ahol ilyenkor kétjegyű verziószámnak kell szerepelnie. A KBM BOOT parancsával újra hívható, ezáltal lehetővé válik az OS rezidens cseréje papírszalag felhasználása nélkül.

Algoritmus:

↓ ←
'A rezidens verziószáma?/nn/' kérdés begépelése

Keresés a file direktory-ban PCSSYS.nn-ért

FOUND?
n
i 'Nincs ilyen verziószám' üzenet gépelése

↓
PCSYS.nn tárho töltése

↓
a PCSYS-ben szereplő handlerok betöltése a
kivánt cimre

↓
A KBM betöltése

↓
Vezérlésátadás KBM-re

↓

Speciális adatai:

Speciális adata a 'nn' verziószám. Az OS rezidens neve ebből képződik, a perifériák nevei kötöttek, és egyben a PCSYS-ben tároltak is.

Lehetőség szerint a programnak 1 lemezblokkon el kell férnie, ellenkező esetben 2 részre osztandó és a második fél BOOT3 néven a file direktory-ból hivandó /overlayként/. Mivel a KBM betöltése az utolsó fázis, ennek helyén szerepelhet.

6.4.3. Kezelési rend

A BOOTØ használatát a HW kézikönyvek tartalmazzák. A BOOT1 stand-alone program papírszalagon található és befűzése/beolvasása után az Ø.lemezegységen található rendszerlemezről a BOOT2 program betöltődik.

A kezelői konzolra feliródik a betöltő program verziószáma, és bekéri a betöltendő rendszerrezidens 2 számjegű verzószámát. A további betöltés automatikus, nem igényel operátori közreműködést, s a legközelebbi üzenet a KBM-től érkezik, amely parancsra vár, a rendszer üzemképes.

- A fellépő hibák - a lyukszalagolvasó vagy Ø.lemez nincs bekapcsolva
- hibás HW indítás
 - nem a BOOT1 papírszalagot olvastuk be
 - a Ø. lemezegységen nem rendszerlemez található
 - nem létező verziószámot gépeltünk be

6.4.4. Tesztelés és implementálás

A tesztelés a szimulációs rendszer alatt történhet, bár a BOOT2 már egy, a lemezen felépített file rendszert igényel. Mivel saját filekezelést használ, így független

FCS-től, az INIT funkcióit azonban már előzőleg alkalmazni kellett. Természetesen a betöltendő handlereknek, OS rezidensnek és KBM-nek már léteznie kell a végső teszthez.

7. Tárkezelés

A ϕ -s memóriában a rendszer foglal helyet. Egy része statikus; más része dinamikusan változik.

Fixen benn van:

- System /és esetleg a konzol/ handler,
- A Keyboard Monitor,
- File kezelő user service rutin-ok.
- FCS

A tranziens része core image file-ok formájában a diszk befagyasztott területén van; onnan töltődnek be; fix címekre. Egy területet több program foglalhat; így célszerű egy memória-foglaló táblát létesíteni.

munka ter. 1 kezdőcím	Az éppen benn levő program jele
munka ter. 2 k cím ⋮	programjel

Az 1,2 memóriablokkokban felhasználói program fut. Ennek jellemzőit egy Memory Control Block-ban, a ϕ -s blokkban tároljuk, ennek címe a SYSCOM-ban található.

A Memory Control Block a ráugrási címet és a program memóriaszegmenseinek adatait tartalmazza.

Az MCB max. 10 szegmenst enged meg.

Az MCB-t a RUN és GET műveletek töltik be.

8. Implementációs megfontolások

8.1. Kidolgozási terv:

A feladat megoldása az alábbi ütemezésben képzelhető el:

okt. végéig: a vázlatos rendszerterv véglegesítése.
dec.15: programok elkészítése.
jan.15: rendszerbelövés+fejlesztői dokumentációk.
jan.31: alkalmazói dokumentációk.

A fejlesztést egy időben többen végzik. Ellenőrzési pont:

okt. vége, dec.15, jan.15., jan. 31.

A fő egységek kidolgozói:

Keyboard Monitor: Abaffy József

User Service Routines: Békési Gábor

FCS+device handler-ek: Manfréd Krzikalla és Bánkfalvi Zsol

INIT, GEN, BUILD, BOOT valamint az ellenőrzési pontok megszervezése ill. a koordinálás: Héjjas Attila és Krafft Walter.

8.2. Dokumentáció és programstílus

Az operációs rendszer dokumentációja áll:

- Vázlatos rendszerterv
- Programdokumentációk
- Felhasználói dokumentációk

A vázlatos rendszerterv koncepcionális szinten tárgyalja a rendszert.

A programdokumentációk felépítése:

1. Általános leírás

Funkció; algoritmus; jellemző adatok; modulstruktúra

2. Programmodulok leírása

/elég egy jól kommentált lista/

3. Példák ill. néhány szóban a teszt-megjegyzések.

4. Kezelési és hívási tudnivalók.

/Példával!/

Függelék: hibajelzések, összefoglaló lista stb.

Felhasználói dokumentációk:

1. Kezelői kézikönyv:

A rendszer indítása, kezelése

2. Programozói kézikönyv:

FCS, USR használata programból

Láncolás, stb. USR funkciók

Hívási módok

Tesztelési utmutató /kezelési vonzatokkal/

3. Generálási kézikönyv

Programstílus:

- A fejlesztést a kereszt rendszeren, PDP-n kezdjük el.
- Minden program első verzióját PCL-ben írjuk!
- A még nem kész részeket szimuláljuk!

Ha valakinek másik témában átnyuló eredménye vagy változtatása van, azonnal adja le ATTILÁ-nak vagy Walter-nak. A hasznosnak vélt saját rutinokat is letisztázva mindenki bocsássa a köz rendelkezésére!

Késznek egy programot csak a PC-4000 futás alapján lehet nyilvánítani.

Várhatóan sok változás lesz a kiinduló állapothoz képest; ezért igen fontos az állandó, párhuzamos dokumentálás, hogy ne vesszen el a fonál!