TPV telefonközpontok hívásfeldolgozó feladatainak programozása

MAKAY ATTILA—
HASENAUER MIKLÓS—
DR. REZNÁK ROXÁN
BHG

1. BEVEZETÉS

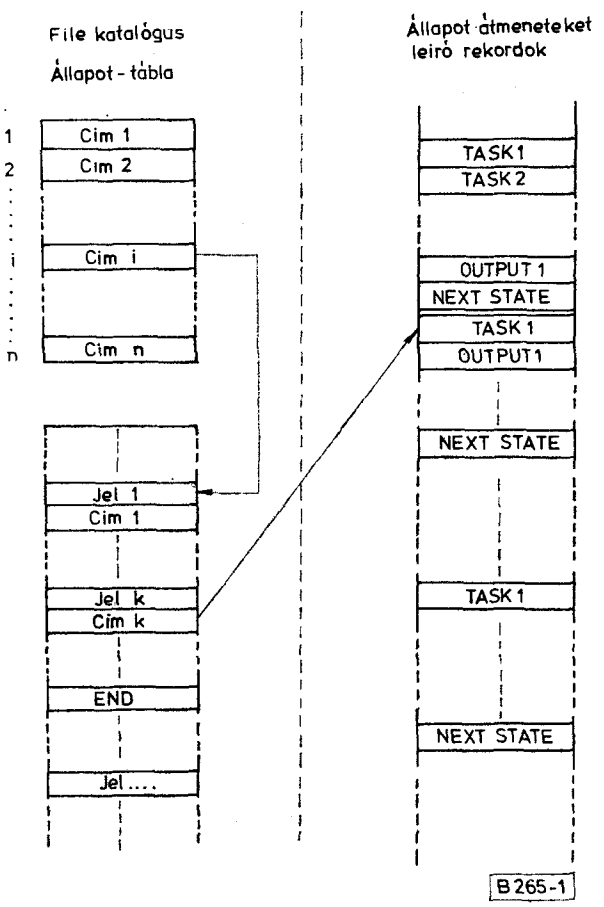
Valamely algoritmizálható feladat számítógépes megoldása során a tényleges programozói munkát általában több tervezői lépcső előzi meg. A feladat specifikálása, algoritmusterv készítése, az adatbázis megtervezése stb. mind jól körülhatárolt, konkrét tervezési munkát jelent, melyek outputja képezi a programozói munka kiindulópontját. Tárolt Program Vezérlésű (TPV) telefonközpontok esetén ezen kiinduló tervezési fázisokra az (1) alatt találhatunk konkrét példát. Ott a tervezési munka „végtermékei" a flowgramok és a Specification and Description Language (SDL) diagramok voltak. Az automaták programjait leíró SDL ábrák már igen közel állnak a gépi megvalósításhoz, hiszen minden állapotban, minden egyes jelre megadják a választ, vagyis a következő állapotot, az állapotátmenet során elvégzendő taskokat és az outputokat. Így ha találunk egy olyan módszert, melynek segítségével az SDL diagrammokat gépi úton olvashatóvá kódoljuk, akkor tulajdonképpen megoldottuk az automaták programozását. Egy lehetséges módszer a következő: Állítsunk össze egy olyan file-t, melynek annyi rekordja van, ahány állapota van a választott automatának. Az egyes rekordok annyi tételt tartalmaznak, ahány jelet az automatának az illető állapotban meg kell különböztetnie. Az egyes jelekhez tartozó tételek felsorolják az illető jel vételekor elvégzendő taskokat és a megvalósítandó outputokat, végül a felveendő új állapotot. Az ilyen módon megszerkesztett file olvasására kiképzett „processzor"-nak csak meg kell találni a pillanatnyi állapot és az érkezett jel alapján az aktuális tételt, és végrehajtani az ott felsorolt rutinokat *(1. ábra)*.

A valóságban a helyzet azonban nem ilyen egyszerű. Nem vettük figyelembe ui. az SDL diagramokban előforduló belső döntéseket, elágazásokat, amelyek hatására a file szekvenciális olvasása helyett „ugrásokra" kell kényszeríteni processzorunkat. Ez azt jelenti, hogy a feldolgozás sorrendje függhet pl. a végrehajtott rutinok által végzett számítások eredményétől. Gondoskodni kell tehát a processzor számára többféle „utasításról", a file-ban levő adatok olvasásának értelmezésére vonatkozóan. Végeredményben egy speciális utasításkészlet, más néven „nyelv" összeállítására jutunk. A szóbanforgó file így ezen a nyelven írt programmá minősül.

A fentiek jegyében született meg a BHG-ban az EPEX központcsalád hívásfeldolgozó folyamatainak programozására szolgáló magasszintű programnyelv, a CPL (Call Processing Language). A nyelv definiálásakor nem az általánosságra törekedtünk, hanem arra, hogy az adott speciális feladat követelményeinek maximálisan megfeleljen. Ennek ellenére úgy véljük, hogy a CPL nyelv ismertetése tanulságos lehet egyéb real-time feladat megoldásánál is.

2. A CPL NYELV ALAPUTASÍTÁSAI

Tekintsük át röviden a nyelv definiálásának, utasításkészlete megválasztásának szempontjait.



1. ábra. SDL nyelvű program „leképzése" gépi formátumra

Egy állapotátmenet alatt a következő típusú feladatok végrehajtására kerülhet sor:

— az automata adatain vagy a közös adatbázison végzett műveletek,
— periféria működtetések,
— üzenet küldése másik automatának (outputok),
— elágazás előkészítése,
— elágazás a beállított feltételek alapján,
— állapotváltoztatás.

Mint real-time alkalmazásoknál általában, a telefonközpontok esetén is vannak olyan feladatok, melyek végrehajtási idejét lehetőség szerint alacsonyan kell tartani. Ezért ezeket az algoritmikusan egyszerű, de sokszor és gyorsan végrehajtandó feladatokat célszerű assembly nyelven megírni. Így az első négy feladattípus elvégzésére előre megírt assembly szubrutinok állnak a CPL nyelv rendelkezésére, a nyelv ezek behívására biztosít formai eszközöket. Ezek az assembly rutinok a nyelv ún. blokkjai, és az ilyenekre épülő nyelvet az irodalomban blokkorientált programnyelvnek is szokták nevezni (2).

A CPL nyelvben a szubrutinok behívására az ‚EXEC' utasítás szolgál. Operandusként adjuk meg a behívandó szubrutin nevét és a végrehajtáshoz esetleg szükséges paramétereket. A szubrutinok feladatuk elvégzése után annak eredményeiről információt tudnak közölni, melynek alapján a CPL programban elágazások hajthatók végre. A rutinok a paraméterátadást a célgép hardware regiszterein keresztül végzik. A CPL nyelv három ilyen paraméter fogadására van felkészítve: két regiszter (X, I) és egy flag (T) típusú változó átvételét terveztük, akár egyidejűleg is. Ezek tartalma a következő ‚EXEC' utasításig hozzáférhető marad az elágazást végrehajtó CPL utasítások számára. Ezen utóbbiak a következők lehetnek: ‚GOTO', ‚BRANCH', és ‚CASE'.

Az elágazások megvalósításához az szükséges, hogy a vezérlésátadás helyét meg lehessen adni. Ezért a CPL nyelvben bármely végrehajtható utasítás címkézését megengedtük.

A feltétel nélküli ‚GOTO' utasítás hatására a program végrehajtása a címkével megadott helyen folytatódik. A feltételes „GOTO' utasításoknál a feltétel teljesülése esetén ugyanez történik, ellenkező esetben a következő CPL utasítás hajtódik végre. A feltételek a következők lehetnek:

$$T = 1$$
$$T = \varnothing$$
$$X = \text{paraméter}$$
$$1 = \text{paraméter}$$
$$X \neq \text{paraméter}$$
$$I \neq \text{paraméter}$$

ahol a paraméter szám vagy szimbolikus név lehet.

Több irányú vezérlésátadást valósít meg a ‚BRANCH' és a ‚CASE' utasítás. A ‚BRANCH' utasításban az X vagy I átadott paraméter szerinti elágazást kérhetjük, oly módon, hogy az utasítás további paramétereiben címkéket sorolunk fel. A vezérlés az n. címkére kerül, ha a megadott paraméter értéke n.

A ‚CASE' utasításban az X vagy I megadása után operandusként paraméter=címke alakú kifejezéseket sorolunk fel. A vezérlés arra a címkére adódik, amelyhez tartozó „paraméter" érték megegyezik a hivatkozott átadott paraméter (X ill. I) aktuális értékével.

A ‚BRANCH' utasítás használata akkor célszerű, ha X vagy I értékkészlete kis egész számokra korlátozódik, ezzel szemben a ‚CASE' használata szükséges akkor, ha abban tetszőleges értékek is előfordulhatnak. Így például az 1. ábrán az „állapottábla" által előírt elágazást ‚BRANCH', az állapoton belül az üzenetkódok szerinti elágazást ‚CASE' típusú utasítással célszerű megvalósítani.

Az állapotátmenet befejezését az ‚END' vagy a ‚NEXT' utasítással írhatjuk elő. Az ‚END' utasítás hatására a feldolgozás az automata állapotának megváltoztatása nélkül fejeződik be. A ‚NEXT' utasítás ezzel szemben az automata új állapotának az utasítás paraméterében megadott értéket rendeli, és így fejezi be a feldolgozást. Ha ez a paraméter ‚IDLE', akkor az automata megszűnik létezni, az automatát reprezentáló erőforrás felszabadul.

Két további végrehajtható utasítást is definiáltunk, melyeket azonban nem a programozó írja le, hanem a fordítóprogram generálja automatikusan. Ezekre a későbbiekben visszatérünk.

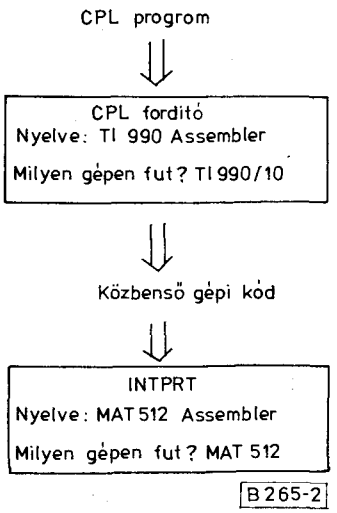
Az utasításokat a függelékben foglaltuk össze.

3. FORDÍTÁS, FUTTATÁS

A nyelv implementációja során a várható alkalmazások által támasztott feltételekből indultunk ki. Az EPEX központokban jelenleg egy speciális, BHG által kifejlesztett CPU-ra épített vezérlő berendezést használunk (MAT 512), de a közeljövőben várhatóan kommersz mikroprocesszor alapú vezérlőberendezésre fogunk áttérni. A CPL nyelv természetesen gép-független, és ezt a függetlenséget célszerű a fordítóprogram számára is lehetőség szerint megőrizni. Ezért a fordítást cross-compilerre bíztuk, mely a BHG fejlesztőrendszerén futtatható. (Ennek magja egy TI 990/10 kisszámítógép.) A fordító emellett nem a vezérlő számítógép gépi kódjára, hanem egy közbenső kódra fordít, melyet a vezérlő számítógép betöltés után utasításról utasításra értelmez, interpretál. Az ilyen ún. interpreter típusú fordítók használatának másik ismert (3) előnye, hogy a tárgykódot tekintve jelentős memóriamegtakarítást eredményez. A fő előnye azonban az, hogy a vezérlő berendezés cseréjekor csupán az új vezérlőn futó, a közbenső kódot értelmező interpreter programot kell újraírni, melynek mérete a compiler-hez képest elhanyagolható. MAT 512-re az INTPRT program (*2. ábra*) mérete 0,5 K byte.

Az INTPRT program üzenet érkezésekor aktivizálódik, és paraméterként az automata azonosítóját (STATE) és az üzenet kódját kapja meg. Ezeket az értékeket kapják az INTPRT X, I változói, mielőtt a CPL utasítások értelmezése elkezdődik.

Teljes mértékben azonban természetesen nem lehet figyelmen kívül hagyni az alkalmazott vezérlő berendezés bizonyos tulajdonságait. Egyrészt a 8 bites processzor korlátokat szab a CPL programban megengedett konstansok és változók méretére, másrészt figyelembe kell venni a MAT 512 lapszervezésű me-



2. ábra. CPL nyelvű program fordítása és futtatása

móriáját (256 Byte-os lapok, ún. szegmensek önállóan címezhetők).

Az INTPRT program rendkívüli mértékben egyszerűsödik, ha a CPL utasítások a közbenső kódban nem lépik át soha a szegmensek határát. Ezt a fordító segítségével viszonylag egyszerűen biztosítani lehet. Ha egy adott CPL utasítás már nem fér ki a lap aljára, akkor azt a következő lap elejére kell fordítani, és a fennmaradó helyeket üres utasításokkal kell kitölteni. Erre szolgál az előző pontban említett két végrehajtható utasítás közül az egyik, a ‚NOP'. A másik, a ‚BASE', melynek szerepe az alábbi:

Mint említettük, az ‚EXEC' utasítás szolgál a hívásfeldolgozás assembly rutinjainak behívására. A szubrutinok címeit a fordító egy IRT nevű táblázatba helyezi el (a tárgykódban konkrét értéket természetesen a szerkesztés során kap).

Ezt a táblát az INTPRT bázisregiszteres relatív címzéssel címzi. A bázisregiszter állítása azonban nem a programozó feladata, erről is a fordítóprogram gondoskodik. Ha a bázisregiszter értékének módosítása szükséges, akkor az ‚EXEC' utasítás elé a fordító egy ‚BASE' utasítást generál.

A lapszervezésű memória további következménye, hogy kétféle címzési módot vezettünk be további memória megtakarítás elérése végett. Ez annyit jelent, hogy kétféle ‚BRANCH' és ‚CASE' típusú utasításunk van jelenleg. A kettő között csak annyi a különbség, hogy az utasítások által hivatkozott címkéket tartalmazó táblázatok mérete dupla vagy szimpla, aszerint, hogy a cimkék teljes (szegmens + sorcím) vagy csak sorcímét adják-e meg.

A ‚BRANCH' és ‚CASE' utasításokhoz hasonlóan a ‚GOTO' utasítás is tartalmaz egy ugrási címet, mely szintén lehet teljes vagy rövid.

4. A FORDÍTÓPROGRAM

A CPL fordító a felhasználó által írt CPL nyelvű forrásprogramból a függelékben ismertetett gépi formátumú object kódot generál. Ezt kívánságra hajlékony mágneslemezre viszi (floppy disc) és/vagy egy olyan listát készít, mely a forrásnyelvi sorok mellett tartalmazza a vezérlő számítógép memóriájába betöltendő tényleges tartalmat is (memóriacím + object kód; lásd 3. ábra).

A fordító a forrásprogramban a már említett hatfajta utasításon kívül az ‚EQU', ‚PAGE' és ‚STOP' direktívákat fogadja el. Az ‚EQU' direktívával kell megadni a programban előforduló nem konstans paramétereket (pl. állapotkódokat). A ‚PAGE' lapváltásra szolgál az output listán, míg a ‚STOP' jelzi a program végét.

A forrásprogram formátuma tulajdonképpen kötetlen. Ez azt jelenti, hogy az utasítás és az operandusmezők bármilyen pozícióban kezdődhetnek. Két megkötés van mindössze:

— Ha van címke, akkor annak az első pozícióban kell kezdődnie és fordítva — a megjegyzés sor kivételével — az első pozícióban csak címke kezdődhet.
— A címke-, utasítás- és operandusmezőket legalább egy szóköznek kell elválasztania.

A fordítóprogram megengedi folytató és megjegyzés sorok alkalmazását is. A szimbólumnevek nem lehetnek 8 karakternél hosszabbak és csak betűvel kezdődhetnek.

Az object program előállításához a fordító program háromszor olvassa végig a forrásprogramot. Az első menet végzi a szintaktikai ellenőrzést. Megkeresi a definiálatlan vagy többször definiált címkéket és paramétereket, túl nagy értékű konstansokat, hibás formátumú utasításokat stb. Az első menet felveszi a további menethez szükséges címke, paraméter és szubrutin táblázatokat is, kitölteni azonban csak a paraméter táblázatot tudja. Ennek az az oka, hogy a felhasználónak nem kell definiálni a programjában előforduló szubrutinokat. Az első menet a hivatkozott szubrutinokat felveszi egy táblázatba, ahová a hivatkozások számát is feljegyzi. Ha az első menet hibátlan volt, akkor a menet végén a fordító gyakoriság szerint csökkenő sorrendbe rendezi a rutintábla elemeit.

Ez a sorrend már megegyezik azzal a sorrenddel, amilyen sorrendben a szubrutinok tényleges címei fognak szerepelni az object kód IRT táblájában. A leggyakrabban előforduló rutinok így a tábla elejére kerülnek, és így minimalizálni lehetett a programba beiktatandó ‚BASE' utasítások számát.

A második menetre marad tehát a ‚BASE' utasítások generálása és így a tényleges utasításszám ismeretében a címketáblázat kitöltése.

A harmadik menet feladata, hogy az előző menetek által felépített táblázatok és a program „végcímének" ismeretében generálja az egyes forrásnyelvi programsoroknak megfelelő object kódokat. A végcímre azért van szükség, mert a fordító az ugrástáblázatokat és az IRT táblát a ‚STOP' utasítás után generálja.

A fordító TI 990/10-es kisszámítógépre készült a gép assembly nyelvén. A program mérete 3200 sor, ami kb. 10 kbyte programot jelent, a táblázatok nélkül. A táblázatok terjedelme 14 kbyte.

A *3. ábrán* az EP 512 hívásfeldolgozó programjának egy kis részletét láthatjuk. A választott automata állapotátmeneteit az (1) alatt hivatkozott példa ismerteti (lásd ott a 3., 4., és 11. ábrákat).

```
1949                        ********************************
1950                        LISTEN   CASE   X/TIMEOUT=CL1
1951                                          OK=CL2
1952                                          CLEAR=CL3
1953                                          CONTINUE=CL2
1954                                          HANSUP=CL5
1955                                          LEAVE=CL6
1956 187/140   23 196 117                     CONGEST=CL4
1957 187/143   41           CL1       EXEC   ALARM
1958 187/144   53                     EXEC   CSPEAS
1959 137/145    4                     NEXT   IDLE
1960 137/146    1           CL2       EXEC   SPA
1961 137/147    8                     END
1962 137/148   25   27      CL3       EXEC   INTMS1/SPEAK
1963 187/150   29   42                EXEC   INTMS2/RELEASE
1964 137/152 135                     EXEC   REAS2
1965 137/153    4                     NEXT   IDLE
1966 187/154   25    8      CL5       EXEC   INTMS1/HANGUP
1967 187/156   29   42                EXEC   INTMS2/RELEASE
1968 137/158 185                     EXEC   REAS2
1969 137/159    4                     NEXT   IDLE
1970 187/160    1           CL4       EXEC   SPA
1971 187/161 253   54      CL6       EXEC   MSG10/UNLOCK
1972 137/163   53                    EXEC   CSPEAS
1973 137/164    4                    NEXT   IDLE
1974                        ********************************
```

3. ábra. CPL nyelvű program fordítási listája

5. EREDMÉNYEK

Az EP 512 fejlesztése során a hívásfeldolgozó programok már CPL nyelven készültek. Az első központok már üzemben vannak és a tapasztalatok igen kedvezőek. Amellett, hogy lerövidült a programíráshoz szükséges fejlesztési idő, a világos, jól olvasható és követhető dokumentáció nagyban hozzájárult a belövés és próbaüzem ideje alatt fellépő hibák gyors behatárolásához és javításához.

Köszönetnyilvánítás

Köszönet illeti elsősorban Schultz Krisztina és Kádár Irén munkatársainkat, akikkel a programozási munkát együtt végeztük. Ezenkívül köszönetünket fejezzük ki a BHG Fejlesztési Intézet vezetőinek, akik lehetővé tették a cikk publikálását.

FÜGGELÉK

A CPL utasítások leírása

1. BRANCH

CPL formátum:

[címke] BRANCH {Regiszter}, címke 1 [, címke 2, ...]

Gépi formátum:

utasításkód
ugrástábla
kezdő címe

Az utasítás hossza: 3 byte
Az utasításkód függ: a megadott regisztertől és attól, hogy rövid vagy teljes címes-e az ugrástábla.

2. CASE

CPL formátum:

[címke] CASE {Regiszter}, param 1=címke 1[, ...]

Gépi formátum:

utasításkód
ugrástábla
kezdő címe

Az utasítás hossza: 3 byte
Az utasításkód függ: a megadott regisztertől és attól, hogy rövid vagy teljes címes-e az ugrástábla

3. END

CPL formátum:

[címke] END

Gépi formátum:

ut. kód

Az utasítás hossza: 1 byte

4. EXEC

CPL formátum
[címke] EXEC szubrutinnév [, par 1[, par 2[, ...]]]

Gépi formátum

ut. kód
param 1
⋮
param k

Az utasítás hossza: 1—31 byte
Az utasításkód függ attól, hogy a szubrutin címe az IRT táblázat adott — a bázisregiszter által kijelölt — félszegmensén belül hol helyezkedik el.

5. NEXT

CPL formátum
[címke] NEXT paraméter

Gépi formátum:

ut. kód
param.

Az utasítás hossza: 1—2 byte
Az utasításkód függ attól, hogy van-e paraméter is megadva.

6. GOTO

CPL formátum
[címke] GOTO [feltétel] címke

feltétel: $\left\{\begin{matrix}0\\1\end{matrix}\right\}\left\{\text{Regiszter}\right\}\left\{\begin{matrix}\text{EQ}\\\text{NE}\end{matrix}\right\}\left\{\begin{matrix}\varnothing\\\text{paraméter}\end{matrix}\right\}$

Az utasítás hossza:

feltétlen ugrás, rövid cím esetén : 2 byte
feltétlen ugrás, teljes cím esetén : 3 byte
feltételes ugrás, rövid cím esetén: 2—3 byte
feltételes ugrás, teljes cím esetén: 3—4 byte

Az utasításkód függ az ugrás típusától és attól, hogy a címkére teljes vagy rövid címmel kell-e hivatkozni.

Gépi formátum:

ut. kód
param.
címke
értéke

7. BASE

CPL formátum:

[cimke] BASE

Gépi formátum

ut. kód

Az utasítás hossza: 1 byte
Az utasításkód függ a bázisregiszter aktuális értékétől

8. NOP

CPL formátuma nincs

Gépi formátum:

0 0 0 0 0 0 0 0

Az utasítás hossza: 1 byte

IRODALOM

[1] *Makay Attila:* TPV-telefonközpontok hívásfeldolgozó rendszerének funkcionális specifikációja. BHG Műszaki Közlemények XXVIII. évf. 1982. 5. 4.
[2] *Takács Gábor:* BCL—80 blokkorientált real-time programnyelv. Automatizálás 79/11 p. 46—57.
[3] *M. T. Hills, S. Kano:* Programming electronic switching systems 1976. Published by Peter Peregrinus Ltd.
[4] *I. Blackhurst; J. S. Gandee:* Programmiersprache zur Verbindungssteuerung in Vermittlungssystemen. Elektrisches Nachrichtenwesen, Band 53, 1978/3 p. 236—240.