Ing. Tomáš Bechný, Ing. Pavel Juřina
NHKG Ostrava

RACIONÁLNÍ POUŽÍVÁNÍ PROGRAMOVACÍHO JAZYKA PL/I POD PL/I(F) KOMPILÁTOREM

A, ÚVOD

Na loňském semináři jsme v diskusi hovořili o zkušenostech s programováním v jazyce PL/I v NHKG. Tehdy bylo konstatováno, že programy, převedené z počítače LEO 360 na počítač IBM 370/145, spotřebovávají více strojového času. Rozborem převedených programů bylo zjištěno,že se v nich používají programovací techniky PL/I, které mají vysoké nároky na spotřebu paměťového prostoru a na spotřebu času základní jednotky (času CPU). Proto jsme se zhruba od dubna 1976 zabývali problematikou překladu jazyka PL/I do strojového kódu. Byla to práce dosti obtížná, neboť firma IBM neposkytuje k této problematice dostatek materiálů a PL/I(F) kompilátor bylo nutno zkoumat jako černou schránku.Přesto se nám podařilo objasnit řadu problémů překladu jazyka PL/I do strojového kódu a na základě těchto zkušeností jsme pak vypracovali programy, jejichž velikost a spotřeba času CPU je srovnatelná s programy, napsanými v jazyce ASSEMBLER.

Ve článku "Why Not PL/I?" v časopise DATAMATION z dubna 1977 Mr Lee Milligan, specialista na programovací jazyky jedné americké společnosti, při rozboru efektivity jazyka PL/I uvádí: "Kvalitní programátor v PL/I může vytvořit stejně efektivní programy jako průměrný programátor v jazyce ASSEMBLER. Přitom na jejich tvorbu spotřebuje zpravidla méně času a ještě méně času spotřebuje na jejich ladění.". Tento názor považujeme za správný. Podařilo se nám totiž napsat programy v jazyce PL/I, u nichž jeden příkaz PL/I odpovídá přibližně dvěma strojovým instrukcím,přičemž některé partie takového progra-

mu jsou přeloženy tak jednoduše, že v jazyce ASSEMBLER to jednoduseji provést nelze. Mohli jsme provést srovnání programů, které byly přepracovány podle nových poznatků. Jejich velikost (velikost přeloženého modulu) klesla na čtvrtinu, vyjímečně až na desetinu a spotřeba času CPU klesla na čtvrtinu, vyjímečně až na osminu původní hodnoty.

Tyto skutečnosti, podle našeho názoru, nesnižují kvalitu jazyka PL/I. Tento jazyk umožňuje na jedné straně, že pracovník po čtrnáctidenním školení může napsat jednorázový program, na jehož efektivitě nezáleží a na druhé straně umožňuje zkušenému programátorovi napsat efektivní program, zpracovávaný mnohokrát za den. Časové a pamětové nároky těchto dvou programů budou samozřejmě diametrálně odlišné. Z tohoto pohledu nemůžeme souhlasit s hodnocením jazyka PL/I, tak jak bylo proneseno v některých příspěvcích na minulých seminářích, kdy se konstatovalo - cituji: "...provedli jsme porovnání stejných programů, programovaných v PL/I a v COBOLu. Závěr svědčil o naprosté nevhodnosti PL/I pro agendy...". Zřejmě zde nebyla brána v úvahu úroveň znalostí programovacího jazyka, jeho možnosti, kvality programátora a pod. Jednoznačné porovnání je možné pouze na základě rozboru přeloženého programu.

V našem materiálu jsme se pokusili shrnout poznatky o programovacím jazyce PL/I do několika uceleních kapitol. Nečiníme si nárok na úplnost probírané problematiky. Protože místo pro náš příspěvek v tomto sborníku bylo omezeno, museli jsme upravit nebo vypustit tyto kapitoly:
B.5.1. Optimalizace smyček a indexování (vypuštěna)
C.2. Převod údajů ze zhuštěného do zónového tvaru (upravena)
F. Zabudované funkce a pseudoproměnné pro řetězce (vypuštěna)
J. PL/I(F) kompilátor a systém OS-VS (vypuštěna)
Obsah následujícího příspěvku spolu s výše uvedenými kapitolami vyjde asi ve 2. čtvrtletí 1978 v NHKÚ.
V textu příspěvku se pro vysvětlení způsobu překladu používají symbolické názvy strojových instrukcí (bližší vysvětlení viz [4]).

B. DEKLARACE A ATRIBUTY DAT

B.1. ATRIBUTY TŘÍDY PAMĚTI

B.1.1. STATIC

Atribut STATIC používejte:

- pro proměnné s atributem INIT
- pro jednoduché pomocné proměnné, čítače, indexy, proměnné typu POINTER a pod.
- pro proměnné, které chcete překrývat technikou DEFINED s výjimkou viz kapitola E.2.1. bod I.
- pro proměnné, které chcete překrývat technikou BASED.

B.1.2. AUTOMATIC

Atribut AUTOMATIC můžete použít:

- pro jednoduché pomocné proměnné s atributem BINARY nebo DECIMAL
- pro řetězce, výjma viz kapitola E.2.1. bod II., které chcete překrývat technikou BASED
- je-li nutno mít datové oblasti paměti, které chcete při návratu z podřízeného bloku automaticky uvolnit; (takto organizované datové oblasti však u virtuální paměti téměř ztrácejí smysl).

Atribut AUTOMATIC nepoužívejte:

- pro proměnné s atributem INIT (viz kapitola E.2.1. bod II.)
- pro proměnné, které chcete překrývat technikou DEFINED (viz kapitola E.2.1. bod III.).
- pro proměnné typu struktura v případech, kdy prvky struktury vyžadují DO nebo DO (viz kapitola E.2.1).

Poznámka 1:

Při adresování datových oblastí, vytvářených kompilátorem PL/I(F) (STATIC AREA, DYNAMIC STORAGE AREA) větších než 4k bytes dochází k problémům s adresováním proměnných a konstant, umístěných na relativních adresách větších než 4k. To představuje nejméně jednu strojovou instrukci *LOAD* navíc pro každou příslušnou instrukci PL/I, což prodlouží jak délku zkompilované procedury, tak čas jejího zpracování. Z tohoto důvodu se doporučuje rozvážnost použití atributu STATIC a AUTOMATIC a při-

hlédnutím k pravidlům uvedeným v bodě B.1.1. a B.1.2 tak, aby délky oblastí byly menší než 4k bytes.

B.1.3. BASED

Každá instrukce PL/I, v níž je použito proměnné s atributem BASED, je po překladu vždy provázena strojovou instrukcí LOAD navíc ve srovnání s třídou paměti STATIC (je-li STATIC AREA menší než 4k viz kap. B.1.2.). Má-li příslušný adresový ukazatel atribut INTERNAL, jedná se o jednu strojovou instrukci LOAD, pro atribut EXTERNAL jsou to 2 instrukce. U modelu 370/145 trvá instrukce LOAD jen 1,688 mikrosec. a je dlouhá 4 bytes. V prologu bloku se však nevyskytují žádné přípravné instrukce pro třídu paměti BASED.
Vzhledem k tomu, že třída paměti BASED dává programátorovi volné pole při adresování jakýchkoliv datových oblastí, lze jejím vhodným použitím dospět k velmi efektivním programům.

B.1.4. CONTROLLED

Z hlediska času zpracování tuto třídu paměti nedoporučujeme. Lze ji úspěšně nahradit třídou BASED.

B.2. ATRIBUTY PRO OBLAST LOKALIZACE DAT

B.2.1. EXTERNAL

Nedoporučujeme používat pro předávání dat mezi jednotlivými externími procedurami z následujících důvodů :

- v prologu procedury (bloku), ve které je daná proměnná deklarována, jsou generovány 2 až 3 strojové instrukce pro každou proměnnou;
- každá instrukce PL/I, v níž se vyskytuje proměnná s atributem EXTERNAL, je provázena vždy 2 instrukcemi LOAD;
- použití těchto proměnných zvyšuje vazebnost mezi externími procedurami.

Při frekventovaném používání externích procedur, pracujících s proměnnými s atributem EXTERNAL, může docházet k enormní spotřebě času CPU.

B.2.2. INTERNAL

Doporučujeme používat.

B.3.ATRIBUT DEF A PSEUDOPROMĚNNÁ(ZABUDOVANÁ FUNKCE) SUBSTR

Při vhodném použití atributu DEFINED tak, aby nedocházelo k operacím v prologu bloku (dle pravidel,uvedených v kap.B.2), se proměnná s tímto atributem chová jako proměnná s atributem STATIC (je-li STATIC AREA menší než 4k viz kap. B.1.2).

Použití SUBSTR je o něco náročnější na velikost paměti a čas CPU. Nejsou však žádné přípravné operace v prologu bloku.

Příklad :

```
DCL  (A,B)  CHAR(5) STATIC,
      A1    CHAR(1) DEF A,
      B1    CHAR(1) DEF B;

/* A */
A1=B1;     /* MVC, 6 BYTES,  5.7 MIKROSEC   */
/* B */
SUBSTR(A,1,1)=SUBSTR(B,1,1);  /* LA,LA,MVC */
                    /* 14 BYTES,  8.6 MIKROSEC  */
/* C */
DCL   CBYTE BIN(15,0) STATIC INIT(3);
SUBSTR(A,CBYTE,1)=B; /* LH,STH,LA,A,BCTR,MVC */
                    /* 24 BYTES,  14.3 MIKROSEC */
```

Upozornění: Je-li 3.argument funkce SUBSTR proměnná,překládá se pomocí modulu PL/I a čas CPU se značně zvyšuje.

B.4. ATRIBUTY DAT (CHARACTER, DECIMAL, BINARY)

Doporučujeme, aby při návrhu popisů souboru se použily výhradně atributy CHARACTER nebo FIXED DECIMAL bez měřítka
(př. FIXED(p) a ne FIXED(p,q)),
pokud to při složitých výpočtech není nutné.

Přitom popis typu CHARACTER by měl být výhradně použit pouze pro ty údaje, kde se písmena a ostatní znaky skutečně vyskytují, nikdy by neměl být použit pro číselné údaje.

Důvody, které nás vedou k tomu, abychom nedoporučili používat pro číselné údaje popisu CHAR, jsou tyto :

a) Při jakémkoliv počítání s takto deklarovanou proměnnou dochází ke konverzi, což je cca 30x pomalejší než u typu DEC FIXED.

b) Položky typu CHAR zabírají cca o 40% více místa na paměťových mediích.

c) Přesun dat typu CHAR se ve většině případů provádí přes pracovní oblast (WSl...) a tedy pro jeden přesun je třeba 2.instr. MVC. Provedeme-li tentýž přesun s daty typu FIXED DECIMAL, výsledný efekt je téměř vždy jen 1 instrukce MVC nebo ZAP, nehledě k tomu, že přesouváme menší počet bytes.

d) Omezíme práce programu s DOPE VECTORY (viz [3]).

e) Vhodným přiřazením lze převést data typu FIXED DECIMAL do znakového (zónového) tvaru přes popis PICTURE přímo (IN-LINE viz kapitola C.2.).

f) Ve většině případů jsou číselné údaje formálně kontrolovány, což při použití atributu CHARACTER předetavuje mnohdy zbytečné konverze.

- Nepoužívejte proměnné s atributem BIT(n) v případech,kdy n není násobkem 8. Nepoužívejte proměnných s atributem BIT(n) ve strukturách.
- Nepoužívejte bitové pole (nastávají značné komplikace při adresování, které je prováděno pomocí modulu PL/I).

8.5. ATRIBUTY DIMENZE DAT

Používáme-li v programu indexované proměnné, je nutno mít na paměti, že při každém jejím výskytu se vypočítává adresa pomocí indexu. Blíže o výpočtu adres informuje [2] na str.209-217. Údaje v následující tabulce se týkají instrukce

A = A + 1; resp. A(I) = A(I) + 1;

z uvedeného vyplývá, že se budeme používání indexovaných proměnných vyhýbat.

DIMENZE	CPU (μsec)	BYTES
bez dimenze	8.75	6
1	26.43	30
2	40.61	38
3	51.20	48
4	61.19	56

V každém případě, kdy používáme polí je nutno program kompilovat s volbou OPT=2 (případně při deklaraci bloku uvedeme volbu REORDER), protože jinak dochází k výpočtu adres pro levý i pravý operand i když adresy jsou stejné.
U polí používejte tam,kde je to možné,jako indexy konstanty místo proměnných.
Doplňující optimalizační fáze kompilátoru (t.j. ty, které se vztahují na optimalizaci smyček a indexování) jsou vyvolávány pouze při specifikování volby OPT=2.

8.5.2. MOŽNOSTI NÁHRADY INDEXOVANÝCH STRUKTUR

Vedeni snahou o odstranění indexování jsme použili ve velmi frekventovaných částech některých programů následující metodu:

```
/* START PROGRAM */
  DCL 1 SOUB STATIC,
        2 KLIC CHAR(3),
        2 SKUP(10),
          3 S1 FIXED(9,3),
          3 S2 FIXED(7,2),
            :                (délka = 120 bytes)
          3 S21 FIXED(3);
```

```
/* ZPRACOVANI */
DO I=1 TO POCET;
  R1=R1+S1(I);
  IF S2(I)=S3(I)
    THEN R2=R2 + S2(I);
  R3=R3+S1(I)*S3(I);
     :
END;

/* NOVY PROGRAM */
DCL 1 SOUB STATIC,
      2 KLIC     CHAR(3),
      2 SKUP(10) CHAR(120);
DCL STR1 CHAR(120) STATIC;
DCL 1 NS DEF STR1,
      2 S1   FIXED(9,3),
      2 S2   FIXED(7,2),
        :
      2 S21  FIXED(3);
/* ZPRACOVANI */
DO I=1 TO POCET;
  STR1=SKUP(I);      /* PRENOS I-TE CETNOSTI */
  R1=R1+S1;
  IF S2=S3
    THEN R2=R2+S2;
  R3=R3+S1*S3;
     :
END;
```

Použitím této metody, jejímž důsledkem bylo odstranění určování adresy prvku pole pomocí VO(VIRTUAL ORIGIN viz [2] str. 214) a násobitelů, bylo dosaženo několikanásobného snížení spotřeby času CPU na výše uvedené instrukce PL/I.

Poznámka 2:

Místo přesunu řetězců můžeme použít techniku BASED.Strukturu

NS můžeme deklarovat takto:

```
DCL  1  NS  BASED(P),
        2  S1   FIXED(9,3),
        2  S2   FIXED(7,2),
           :
           :
        2  S21  FIXED(3);
```

a příkaz

```
STR1 = SKUP(I);
```

nahradíme příkazem

```
P = ADDR(SKUP(I));
```

Tímto způsobem přiložíme masku přímo na I-tou četnost.

B.6. POUŽÍVÁNÍ POMOCNÝCH PROMĚNNÝCH

- Pomocné proměnné deklarujte na úrovni 1.
- Pomocné proměnné DECIMAL deklarujte vždy, kdy je to možné s měřítkem, které je shodné s měřítkem jako má proměnná, se kterou chcete manipulovat. Uvědomte si, že při použití již deklarované proměnné F9

  ```
  DCL  F9 FIXED(9,2) STATIC;
  ```

  pro uchování obsahu proměnné Z7

  ```
  DCL  Z7 FIXED(7) STATIC;
  ```

  sice ušetříte 4 byte na deklaraci pomocné proměnné F7

  ```
  DCL  F7 FIXED (7) STATIC;
  ```

  ale při přiřazení

  ```
  F9 = Z7;
  ```

  ztrácíte 8 byte na instrukce MVO a XC, které kompilátor vygeneruje navíc pro převod údajů s různým měřítkem.
- Pomocné proměnné určené k výpočtům deklarujte s atributem BINARY, pokud je to účelné (pokud tím nezapříčiníte zbytečné konverze z binárního tvaru do decimálního a naopak).
- Indexy deklarujte zásadně s atributem BINARY FIXED.

C. PŘEVODY DAT

C.1. PŘEVOD ÚDAJŮ ZE ZÓNOVÉHO DO ZHUŠTĚNÉHO (PACKED) TVARU A JEJICH KONTROLA

Údaje ze vstupních medií(ať ze štítků nebo z terminálů) přicházejí v zónovém tvaru.Pro kontrolu a převod číselných údajů do tvaru PACKED doporučujeme následující postup,který je zhruba 6x rychlejší než dosud používané kontrolní techniky,používající přímého přiřazení CHAR→FIXED,při kterém dochází ke konverzi dat vyvolánim modulu PL/I.

Příklad:

```
        DCL  A   CHAR(80)   STATIC;
        DCL  A1  CHAR(3)    DEF A,
             A2  CHAR(5)    DEF A POS(4);
        DCL  (F1 FIXED(3), F2 FIXED(5))  STATIC;
        DCL  P1  PIC'999'   DEF A,
             P2  PIC'99999' DEF A POS(4);

    CTI: READ FILE(VSTUP) INTO(A);
         IF VERIFY(A1,'0123456789 ')>0
         THEN DO;      /* IDENTIFIKACE A OZNACENI CHYB */
           .
           .
         END;
         ELSE DO;      /* PREVOD A DALSI KONTROLA */
           F1=P1;
           IF F1<10 THEN ...
           IF F1>55 THEN ...
             .
             .
         END;
         .
         .
```

Vysvětlení k deklaracím:

na stejná místa vstupního media jsou definovány 2 proměnné: proměnná typu CHAR a proměnná typu PIC stejné délky.

Mezera (BLANK) se automaticky převádí na nulu,ať je umístěna kdekoliv.Z toho vyplývá nutnost zajistit,aby číselné(nebo všechny) údaje byly ve vstupním médiu přiřazeny vpravo.

U vstupu z děrných štítků to lze zcela dobře zajistit. U vstupu z obrazovky využijte možností MESSAGE FORMAT SERVICE v IMS,který provede přiřazení číselných údajů vpravo(viz IMS/360 UTILITIES REFERENCE MANUAL).

Při vstupu dat z dálnopisu,nepodporovaného IMS,vypracujte univerzální modul,který provede přiřazení dat vpravo a bude použitelný pro více programů.Pro systém ARSKELET v MHKG byl vypracován modul HTEST.

Upozornění: při použití uvedeného způsobu převodu údajů se neaktivuje podmínka ON CONV a proto je nutno pomocí instrukce VERIFY nebo jinak zajistit číselnost údajů,protože jinak může nastat při použití proměnné typu FIXED DECIMAL DATA INTERRUPT nebo může dojít ke zkomolení údajů.

Příklad:

```
      DCL  (A CHAR(5),  F FIXED(5))  STATIC,
           P  PIC'99999'  DEF A;

           F=P;        /* STROJOVY KOD: PACK,MVN
                          BYTES: 12, MIKROSEKUND:  9.6 */
```

	1	2	3	4	5	6
A	b123b	b1b23	A1Bb2	CDEFb	1:<3A	bbbbb
F	01230	01023	11202	34560	1AC31	00000

V případě 5 při prvním testu proměnné F nastane chyba typu DATA INTERRUPT.

Vysvětlení funkce převodu(případ 3):

A	C1F1C240F2 (hexadecimálně)
po PACK	1 1 2 02F
po MVN	1 1 2 02C

C.2. PŘEVOD ÚDAJŮ ZE ZHUŠTĚNÉHO DO ZÓNOVÉHO TVARU

Pokud se použije pro popis cílových proměnných(např. pro tisk) PICTURE typu 1,2 a 3 (viz dále),podmínka SIZE není aktivována a hodnota parametru kompilátoru OPT je minim. 1, konverze se zhuštěného do zónového tvaru se provádí IN LINE. Úspora výpočetního času operace IN LINE může být řádově 10 ku jedné ve vztahu ke stejné operaci,prováděné modulem PL/I (OFF LINE).

Ne všechny znaky popisu PICTURE mohou být použity ve specifikaci konverze,prováděné IN LINE.

Povolené jsou:

1. V a 9
2. Pohyblivé(DRIFTING) a nepohyblivé(NONDRIFTING) znaky
   $ S + -
3. Znaky pro potlačení nevýznamných nul Z z
4. Vložené znaky , . / B

Pro konverzi IN LINE je specifikace popisu PICTURE a této podseskupením znaků zařazena do třech typů:

Typ 1: specifikace PICTURE obsahující výlučně devítky s volitelným doplňkem V a předsunutým či připojeným znaménkem nebo symbolem pro označení měny a až se čtyřmi vloženými znaky.

Příklady: '99V999', '9$', '$99V99', '99V+', '$99.9'

Typ 2: specifikace PICTURE se znaky pro potlačení nevýznamných nul, vloženými znaky, znaménkem nebo označením symbolu měny.Kromě toho oproti typu 1 se připouští více než čtyři vložené znaky.

Příklady: 'ZZ$', '$$,$$9', 'ZZ9V.99', '+ZZ.ZB', '$////99'

Typ 3: specifikace PICTURE s prvky pohyblivých znaků, vloženými znaky a se symbolem měny.

Příklady: '$$$$', '-,---9', '$/$$/99', '++++9.99', '$$$9-', 'ZZZ9-', '---', '-.---'

Někdy není konverze,obsahující položku s popisem PICTURE, zpracována IN LINE, i když se popis shoduje s některým z uvedených typů.Toto může nastat z následujících důvodů:

1. Hodnota optimalizačního kódu kompilátoru (OPT=n) je příliš nízká.
2. Podmínka SIZE je aktivní.
3. Není překrytí mezi pozicemi desetinné tečky ve zdroji a v cíli. (Např. konverze mezi FIXED DECIMAL(6,2) nebo FIXED DECIMAL(5,-3) na PICTURE '999V99' bude požadovat vyvolání modulu PL/I.
4. Specifikace údajů,definovaných popisem PICTURE může mít určité charakteristiky,které zabraňují konverzi IN LINE.
   a) Vložený znak mezi Z (nebo z) a prvou 9 není předchózen znakem V (např. 'ZZ.99').
   b) Jsou přítomny pohyblivé znaky nebo znaky pro potlačení nul vpravo od znaku V (např. 'ZZVZZ', '++V++').
   c) Chceme-li potlačit tisk nulových hodnot a znaménko chceme tisknout vpravo od čísla (např. 'ZZZZ-').

Poznámka: vyvolání modulu PL/I (modulu pro konverzi OFF LINE) je v kompilačním výpisu označeno zprávou typu WARNING - DATA CONVERSION WILL BE DONE BY SUBROUTINE CALL IN THE FOLLOWING STATEMENTS ...

Pro převod dat typu FIXED DECIMAL do znakového tvaru a jejich případné zřetězení můžeme použít velmi efektivní způsob,který je uveden v následujícím příkladě.Je zapotřebí převést položky den,měsíc a rok do znakového tvaru a zřetězit je(např. pro vytvoření klíče záznamu).

Provedeme to takto:

```
      DCL  ((DEN,MES,ROK) FIXED(3),
           DATUM PIC '999999') STATIC,
           DR    PIC '99' DEF DATUM,
           MR    PIC '99' DEF DATUM POS(3),
           RD    PIC '99' DEF DATUM POS(5);

           RD=DEN;     MR=MES;     DR=ROK;
      /* STROJOVY KOD KAZDEHO PRIKAZU : OBPK,OI  */
```

D. MANIPULACE S DATY.

D.1. NULOVÁNÍ POLÍ A STRUKTUR.

Pro nulování polí a struktur se dosud používala instrukce typu STR='';, označovaná jako NULL STRING ASSIGNMENT. Tato instrukce se však překládá způsobem,který není pro rychlost zpracování programu vhodný.Dříve než objasníme způsob překladu této instrukce spolu s možnostmi jiného typu nulování,je nutno zcela obecně konstatovat,že nulování by mělo být použito pouze tam,kde je to bezpodmínečně nutné,to je např. u polí(struktur) pro kumulaci hodnot,pro vytváření nových záznamů do kmenových souborů a podobně.

D.1.1. ZPŮSOB PŘEKLADU NULOVÁNÍ

Nulování je přeloženo do strojového kódu tak,že se provádí pro každou položku zvlášť.Pokud je položkou pole,pak kompilátor vygeneruje pro toto pole smyčku,ve které se vynulují všechny prvky tohoto pole.Lze říci,že čím více položek (polí) struktura obsahuje,tím je vygenerovaný strojový kód komplikovanější a provádění pomalejší.

Jako příklad uvádíme orientační překlad instrukce STR='';, byla-li STR deklarována následovně:

```
DCL 1 STR STATIC,          /* STROJOVY KOD */
      2 S1    FIXED(3),    /*    MVC       */
      2 S2    CHAR(5),     /*    MVC       */
      2 S3(3),             /*  SM 1 - 3    */
        3  S4 FIXED(5),    /*    MVC       */
        3  S5 CHAR(4),     /*    MVC       */
      2 S6    CHAR(8),     /*    MVC       */
      2 S7(5) CHAR(2),     /* SM 1 - 5 MVC */
      2 S8(7) FIXED(5),    /* SM 1 - 7 MVC */
      2 S9    FIXED(5);    /*    MVC       */
```

Vysvětlení: MVC - strojová instrukce MOVE CHARACTER
SM 1 - 3 - zjednodušené označení smyčky.

Skutečný překlad této instrukce je mnohem složitější a zahajovací a ukončovací operace smyček a o výpočet adres

prvků pole pomocí virtuálního začátku(VIRTUAL ORIGIN) - viz [2] .Z uvedeného vyplývá,že nemá žádný význam rozepisovat nulování po položkách.

D.1.2. ŘEŠENÍ PROBLÉMU NULOVÁNÍ.

D.1.2.1. DÁVKOVÉ PROGRAMY

Nulování pomocných polí(struktur),které slouží pro kumulaci hodnot a nulují se pouze na začátku programu,je možno nadále nulovat instrukcí typu STR='';

Pro nulování uvnitř hlavních smyček programů používání této instrukce nedoporučujeme.Tam,kde je nulování nutné,použijeme následující techniky:dříve deklarované struktuře STR přiřadíme atribut DEF a dále si deklarujeme pomocný řetězec stejné délky.

```
DCL MSTR CHAR(70) STATIC,
    NULSTR CHAR(70) STATIC;
DCL 1 STR DEF MSTR,
      2 S1 FIXED(3),
        :
```

Na začátku programu(před hlavní smyčkou) provedeme instrukce:

```
STR='';
NULSTR=MSTR;
```

Tím je ve znakovém řetězci NULSTR připraven řetězec hexadecimálních hodnot 40,00,0C pro nulování struktury STR. Uvnitř hlavní smyčky pak pro nulování struktury STR použijeme instrukce MSTR=NULSTR;

Strojový překlad této instrukce obsahuje za předpokladu, že NULSTR < 256 bytes,2 strojové instrukce,když NULSTR ≥ 256 bytes,5 strojových instrukcí + vyvolání modulu PL/I.Použití modulu PL/I lze obejít způsobem,uvedeným v kapitole D.2.

D.1.2.2. PROGRAMY ON-LINE A EXTERNÍ PROCEDURY

V těchto případech je nutno přihlížet k tomu,že program (procedura) se po vyvolání provádí vždy celý i s úvodními operacemi.

Lze použít následujícího způsobu řešení:

1. Nulovací řetězec vyděrujeme buď na děrovači nebo u složitých struktur dávkovým programem.
2. Vyděrované štítky vložíme do programu za štítek MSTR='
(' je vyděrován ve sloupci 72).Za tyto štítky vložíme štítek ';

Tabulka pro děrování:

HEXADECIMÁLNĚ	DĚROVÁNO	ZNAKY
40		mezera
00	12-0-1-8-9	HI/
0C	12-4-8-9	HID nebo ∂I

Příklad:

Obsah nulovacího řetězce pro struktura,deklarovanou v D.1.1.

S1	S2	S4(1)	S5(1)	S4(2)	S5(2)	S4(3)
000C	40404040	00000C	40404040	00000C	40404040	00000C

S5(3)	S6	S7(1)	S7(2)	S7(3)	S7(4)	S7(5)	S8(1)
40404040	4040404040404040	4040	4040	4040	4040	4040	00000C

S8(2)	S8(3)	S8(4)	S8(5)	S8(6)	S8(7)	S9
00000C	00000C	00000C	00000C	00000C	00000C	00000C

| odděluje symbolicky jednotlivé položky.

Složitější řešení jsou nepochybně pracné,je zde nutno zvážit hospodárnost provádění jednotlivých programů.

D.2. PŘESUNY DAT

D.2.1. PŘESUN DAT TYPU STRUKTURA→STRUKTURA

Při přesunech dat tohoto typu je nutno pamatovat na to, že přesun se ve většině případů provádí po položkách(je-li prvkem struktury pole,provádí se přesun pomocí smyčky pro každý prvek pole zvlášť),což klade značné nároky jak na velikost programu,tak na čas provádění.Tento fakt lze obejít přesunem znakových řetězců.

Příklad:

```
DCL (A,B) CHAR(50) STATIC;
DCL 1 STRA DEF A,
      2 A1 FIXED(11),
      2 A2 CHAR(5),
        :
DCL 1 STRB DEF B,
      2 B1 FIXED(11),
      2 B2 CHAR(5),
        :
```

Nemísto: STRA=STRB; výhodnější: A=B;

Je-li délka řetězců <256 bytes, je strojový překlad instrukce A=B;2 instrukce MVC, jinak se přesun provádí pomocí modulu PL/I, což je nevýhodné. Toto lze obejít následujícím způsobem:

```
SUBSTR(C,1,250)=SUBSTR(D,1,250);
SUBSTR(C,251,250)=SUBSTR(D,251,250);  /* ATD */
```

D.2.2. ŘÁDOVÉ POSUNY A MANIPULACE S POLOŽKAMI FIXED DECIMAL

Pro řádové posuny položek typu FIXED DECIMAL lze výhodně použít techniku DEFINED namísto provádění dělení a násobení (jedny z nejpomalejších strojových instrukcí).

Příklad:

```
DCL (F7     FIXED(7),
     F72    FIXED(7,2),
     F5     FIXED(5),
     NULA   FIXED(11)) STATIC;
DCL  DESET  FIXED(11,1) DEF NULA,
     STO    FIXED(11,2) DEF NULA,
     TISIC  FIXED(11,3) DEF NULA;  /* ATD */
```

Nemísto: F7=F72 * 1000; výhodnější: TISIC=F72;
F7=NULA;

Nemísto: F5=F7 / 10; výhodnější: NULA=F7;
F5=DESET;

Naznačený způsob je asi 10 x rychlejší.

Poznámka: ve vhodných případech lze použít přímého překrytí položky FIXED DECIMAL..

```
      F7=F72 * 100;          /* 24 BYTES,195 MIKROSEKUND */
Lze řešit:  STO=F72;         /* MVC */
            F7=NULA;         /* 2 x MVC */
Výhodnější: DCL MF72 FIXED(7) DEF F72;
            F7=MF72;         /* MVC */
```

V případech,kdy je nutno z položky FIXED DECIMAL použít pouze některé řády,je výhodnější namísto operací dělení a používání zabudované funkce MOD použít přesun proměnné FIXED DECIMAL do proměnné s atributem PICTURE,na kterou je přiložena odpovídající maska.Pro další operace se přiřadí tato maska do odpovídající proměnné typu FIXED DECIMAL.

Příklad:
```
DCL (F5 FIXED(5),
     F1 FIXED(1),
     P5 PIC'99999') STATIC,
     P1 PIC'9' DEF P5 POS(5);
Namísto:    F1=MOD(F5,10);   /* 56 BYTES;230 MIKROS. */
Výhodnější: P5=P5;           /* UNPK,OI */
            F1=F1;           /* PACK,MVN */
            /* CELÉ INSTRUKCE DOHROMADY : 22 BYTES
               18 MIKROSEKUND */
```

D.2.3. PŘIŘAZOVÁNÍ ZNAKOVÝCH PROMĚNNÝCH A PROMĚNNÝCH FIXED DECIMAL

Při přiřazování znakových proměnných je výhodné používat proměnných o stejné délce.Při přiřazování konstant je výhodné doplnit tuto konstantu na stejnou délku,jakou má cílová proměnná.

Příklad:
```
DCL A CHAR(3) STATIC;
Namísto:    A='XY';      výhodnější: A='XY ';
```

Při přiřazování proměnných typu FIXED DECIMAL je výhodné používat proměnné se stejným počtem desetinných míst.

Příklad:

```
DCL ((F7,C7) FIXED(7),
     F92     FIXED(9,2)) STATIC;

F7=C7;   /* 6 BYTES  -  MVC          */
F92=C7;  /* 18 BYTES -  MVC,XC,MVZ   */
```

Poznámka:netýká se přiřazování konstant(na rozdíl od znakových řetězců).

D.3. ARITMETICKÉ OPERACE

Při provádění aritmetických operací je vhodné dodržovat následující zásady:

a) aritmetické operace neprovádíme s proměnnými s atributem CHAR nebo BIT.

b) aritmetické operace provádíme pokud možno s proměnnými se stejným měřítkem(se stejným počtem desetinných míst), jinak se překlad komplikuje.

c) pro uložení výsledku používáme proměnné se stejným měřítkem,jaké určí kompilátor(viz[1] strana 47).

Příklad:

```
DCL ((F1,F2,F3,F4) FIXED(7),
     S1 FIXED(7,2))STATIC;


F1=F1+F2;          /* 6 BYTES,9 MIKROS.  */
F1=F1+S1;          /* 42 BYTES,68 MIKROS.*/
F1=F1+F2+F3+F4;/* 42 BYTES,68 MIKROS.*/
F1=F1+F2;
F1=F1+F3;      }   /* 18 BYTES,27 MIKROS.*/
F1=F1+F4;
F1=F2+F3;          /* 18 BYTES,25 MIKROS.*/
```

Poznámka:pro odečítání je překlad prováděn stejně.

Z příkladu vyplývá,že při sečítání(odečítání) více jak dvou proměnných je výhodné přičítat(odečítat) postupně jednu proměnnou pd druhé.

Pro násobení a dělení toto neplatí,protože výpočet se vždy provádí přes pracovní oblast paměti.Při dodržování pravidel a) a b) se násobení provádí na 3 strojové instrukce (18 bytes).Pro dělení však musí být splněna ještě jedna podmínka,a to,aby délka dělence byla 8 bytes.Pak se i dělení provádí na 3 strojové instrukce.

Příklad: DCL ((F1,F2,F3) FIXED(5),
F72 FIXED(7,2),
F93 FIXED(9,3),
F15 FIXED(15),
F156 FIXED(15,6),
F153 FIXED(15,3)) STATIC;

```
F1=F2*F3;       /* ZAP,MP,MVC - 18 BYTES,190 μSEC */
F93=F1*F72;     /* ZAP,XC,MVZ,MVN,MP,MVO,MVN
                   42 BYTES, 220 MIKROSEC.        */

F153=F156/F93;  /* ZAP,DP,MVC - 18 BYTES,230 μSEC */
F1=F15/F2;

F93=F1/F2;      /* ZAP,XC,MVZ,MVN,DP,MVO,MVN
                   42 BYTES,295 MIKROS.           */
F153=F15/F72;   /* ZAP,DP,MVC,XC,MVZ,MVN
                   36 BYTES,244 MIKROS.           */
```

E. PROCEDURY A BLOKY

E.1. FUNKCE PROLOGU A EPILOGU

Každá procedura (interní,externí,hlavní) obsahuje stejně tak jako každý blok typu BEGIN jisté operace,které jsou nutné pro zahájení práce bloku nebo jeho ukončení. V terminologii PL/I se tyto operace (tyto části programu) nazývají prolog a epilog.

V prologu a epilogu se provádějí jednak nutné standardní operace, které nemůže programátor ovlivnit (přidělování paměti,zaměňování registrů a pod.),a jednak se zde provádějí operace,které jsou ovlivněny deklaracemi v daném bloku,pro-

žitými příkazy v bloku a pod.

Obsah prologu, to jsou vlastně nevýkonné (obslužné) strojové instrukce. Při nevhodném použití různých typů dat se může u PL/I(F) kompilátoru stát, že obsah prologu je nesnesitelně velký a při mnohonásobném vstupu do takového bloku dochází k enormní spotřebě času CPU. Uveďme příklad: procedura obsahující 60 příkazů PL/I měla po překladu v prologu asi 1.100 nevýkonných strojových instrukcí, přičemž výkonná část bloku obsahovala cca 100 strojových instrukcí. Po úpravě deklarací v bloku (technika DEFINED byla nahražena technikou BASED) se obsah prologu snížil na 10 strojových instrukcí.

E.2. PROLOG

E.2.1. OBSAH PROLOGU

V prologu jsou kromě standardních instrukcí generovány pomocné instrukce pro následující případy (výčet nemusí být úplný) :

I) Proměnné s atributem DEFINED, jejichž základna má třídu paměti STATIC v případech :

a) mají-li proměnné (nebo struktury) atribut DEFINED a základnou je struktura (Základnou rozumíme proměnnou nebo strukturu, na kterou provádíme překrývání),

b) mají-li proměnné nebo struktury atribut DEFINED a základnou je prvek struktury (identifikátor s úrovní vyšší než 1),

c) jsou-li pro proměnné použity řídící bloky PL/I (DOPE VECTORS (DV), VIRTUAL ORIGINS (VO) viz [3]).
Tyto DV nebo VO používá PL/I v případech :

- vyvolání procedur PL/I (konverze dat, manipulace s bitovými řetězci, STREAM I/O operace, a jiné);
- předání proměnných jako argumenty jiným procedurám.

Poznámka :
Používáme-li v proceduře prvek struktury, přičemž se vyvolává např. konverzní modul (je nutný DV) nebo je-li prvkem struktury pole (při jeho použití je nutný VO), pak jsou v prologu procedury vytvořeny DV i pro zbývající prvky struktury, i když se v programu tyto prvky i DV nikdy nepotřebují.

Doporučení:

Nemůžeme-li se vyhnout použití DV nebo VO, deklarujeme prvek struktury, který vyžaduje DV nebo VO, samostatně na úrovni 1 s atributem DEF POS, čímž zamezíme vytváření DV,VO v prologu pro ostatní prvky struktury.

II) Proměnné s třídou paměti AUTOMATIC pouze tehdy, vyžadují-li DV, nebo mají-li atribut INIT.

III) Proměnné s atributem DEFINED, jejichž základna má třídu paměti AUTOMATIC.

IV) Proměnné s atributem LABEL AUTOMATIC.

V) Proměnné typu LABEL ARRAY, používané jako přepínač (př. DCL L(52) LABEL;).

VI) Proměnné s atributem EXTERNAL.

VII) Proměnné s atributem DEFINED, jejichž základna má atribut EXTERNAL.

VIII) Operace spojené s upřesněním oboru platnosti podmínek typu ON.

IX) Případné operace pro DUMMY argumenty a dočasné proměnné generované kompilátorem.

X) Adresace parametrů.

XI) Určení dimenzí pole s dynamickými mezemi.

XII) Operace související se sekundárními vstupními body.

XIII) Přípravné operace pro proměnné s atributem VARYING.

Poznámka k bodu V_8:

V mnoha programech, zejména tiskových, se používá pole návěští. Možno ho použít dvěma způsoby :

Příklad 1:
```
DCL L(3) LABEL INIT (L1, L2, L3);
GOTO L(I);
L1:
L2:
L3:
```

Příklad 2 : DCL L(3) LABEL ;
GOTO L(I);

L(1):
L(2):
L(3):

Oba způsoby jsou co do rychlosti zpracování zhruba stejné. Každá deklarace pole návěští znamená však prolog, který narůstá úměrně velikosti dimense pole návěští.

Z těchto důvodů nedoporučujeme deklarovat pole návěští jinde než v proceduře, jejíž prolog se provádí jen jednou.

Poznámka k bodu III, VI, VII :
je-li prováděno překrývání pomocí struktur, v prologu jsou prováděny přípravné operace (vytváření DV, VO) pro každý prvek struktury.

E.2.2. VELIKOST PROLOGU

Pro každou proměnnou (prvek struktury) vygenerují se v prologu bloku strojové instrukce zpravidla takto :

pro body I,II,III,IV,VI,IX :
2 - 3 strojové instrukce (8 - 12 bytes)

pro bod V: 3 strojové instrukce (14 bytes) pro každý prvek pole

pro bod VII: 4 strojové instrukce (16 bytes)

- Pro každou ON jednotku (bod VIII):
2 - 3 strojové instrukce (8 - 12 bytes)
- Pro každý parametr (bod X):
2 strojové instrukce (10 bytes).

Je-li seznam parametrů tvořen výhradně POINTERy, jsou to 2 instrukce pro celý seznam parametrů.

- Pro každý sekundární vstupní bod (bod XII):
asi 8 strojových instrukcí (asi 32 bytes)
- Pro každou VARYING proměnnou (bod XIII):
asi 5 strojových instrukcí (asi 24 bytes)

Poznámka :

Údaje o počtu instrukcí a velikosti paměti je nutno brát jako informativní. Přesné informace je možno získat z výpisu OBJECT modulu, pořízeného volbou kompilátoru LIST.

Doporučujeme pro každou frekventovanou externí proceduru provést výše uvedený výpis za účelem minimalizace prologu dle pravidel uvedených v oddílu E.2.

E.3. EPILOG

Epilog se většinou provádí standartním způsobem přes modul PL/I a programátor jeho obsah nemůže ovlivnit.

E.4. PŘEDÁVÁNÍ ÚDAJŮ MEZI PROCEDURAMI

PL/1 (F) kompilátor umožňuje předávat údaje mezi procedurami dvěma způsoby :

a) použitím dat s atributem EXTERNAL

b) parametry

ad a) Nevýhody :

- zvyšuje se vazebnost procedur
- zmenšuje se přehlednost
- generují se přípravné instrukce v prologu procedury, v níž jsou externí data deklarována (viz.kap.B.2.1)
- jsou problémy s adresováním externích dat (viz.kap.B.2.1)

ad b) Při vyvolávání procedur s parametry generuje kompilátor instrukce pro tvorbu seznamu argumentů při každém výskytu instrukce CALL. Nejvíce instrukcí generuje pro argumenty s atributem BASED.

Je-li však použito pro argumenty atributu POINTER STATIC, kompilátor připraví seznam argumentů již v době kompilace (ARGUMENT LIST ve STATIC AREA) a překlad instrukce CALL se zkrátí. Rovněž v prologu volané procedury se překlad zjednoduší.

G. VĚTVENÍ PROGRAMU

G.1. PŘEPÍNAČ

Pro uchovávání stavu programu můžeme použít proměnné různých typů. Vhodnost jejich použití ukazuje následující tabulka.

TYP TESTU	ATRIBUT	BYTES	CPU
K=1	BIN FIXED(15)	12	4.73
K=1	DEC FIXED(1)	8	9.03
K='1'	CHAR(1)	10	3.89
K	BIT(1)	8	2.91
K='00100100'	BIT(8)	10	3.89
K='001'	BIT(8)	22	8.40

Poznámka: ve sloupci CPU je uveden potřebný čas v mikrosekundách, třída paměti proměnných je STATIC.

Z této tabulky je zřejmé, že pro přepínač, pro který je potřeba hodnota 0 nebo 1, je nejvhodnější proměnná s atributem BIT(1), ale testovaná takto:

```
IF K THEN ....
ELSE .....
```

Vysvětlení:

```
IF K='1'B THEN;    /* 24 BYTES,10.5 MIKROS. */
IF¬K THEN;         /* 16 BYTES,10.8 MIKROS. */
IF K THEN;         /*  8 BYTES, 2.9 MIKROS. */
```

V případech, kdy je nutno uchovávat jiné hodnoty než 0 a 1, použijeme proměnné s atributem CHAR nebo BIN FIXED. Velmi nevýhodné je používání bitových polí, kdy při jejich používání dochází k vyvolávání modulu PL/I, což má za následek značnou spotřebu času základní jednotky.

6.2. ROZHODOVACÍ PŘÍKAZY

Tato kapitola navazuje na kapitolu 6.1., kde byly uvedeny testy přepínačů. Dále zde uvádíme několik pravidel, které by se měly dodržovat při psaní rozhodovacích příkazů:

a) Při testování znakových proměnných je nutno psát konstanty v apostrofech (ve formě znakových řetězců), protože jinak dochází k vyvolávání modulu PL/I pro konverzi, což má za následek značnou spotřebu času CPU.

b) Při testování znakových řetězců je vhodné používat pro obě operandy stejných délek - týká se i konstant. V opačném případě jsou nároky na paměť a čas téměř dvojnásobné.

Příklad: DCL A CHAR(4) STATIC;

Nevhodně: IF A='X' THEN ...

Vhodněji: IF A='X ' THEN ...

c) Při testování proměnných FIXED DECIMAL je vhodné používat obě operandy se stejným měřítkem - týká se i konstant.

Příklad: DCL F FIXED(5,2) STATIC;

Nevhodně: IF F>15 THEN ...

Vhodněji: IF F>15.00 THEN ...

d) Není vhodné testování proměnných s různými atributy, protože dochází ke konverzi dat (například při porovnávání hodnoty proměnné s atributem CHAR s hodnotou proměnné s atributem FIXED).

e) Ve frekventovaných částech programu není vhodné zejména u příkazu IF používat složených logických výrazů. Obsahuje-li složený logický výraz proměnnou s atributem BIT, dochází k vyvolávání modulu PL/I.

Příklad: IF A>10 & B < 20 & C='X' & D='YA' THEN ...

Vhodněji:
```
IF A > 10 THEN
   IF B < 20 THEN
      IF C='X' THEN
         IF D='YA' THEN ...
```

f) Při používání zabudované funkce VERIFY je výhodné používat testu typu IF VERIFY(A,'0123')> 0 THEN ...
g) Při testování obsahu paměti v zónovém tvaru je výhodné tam ,kde je to možné,používat testy znakových řetězců.

Příklad:

```
DCL  P  PIC'999' STATIC,
     AP CHAR(3)  DEF P;
     IF P='126' THEN;     /* PACK,MVN,CP
                             18 BYTES,18 MIKROS.*/
     IF AP='126' THEN;    /* CLC
                             6 BYTES,5 MIKROS. */
```

H. VYUŽITÍ VOLEB KOMPILÁTORU

a) Volba STMT je vhodná pouze pro období ladění programu. Z rutinně zpracovávaných programů by měla být vypuštěna (přeložený program je menší,stejně tak i spotřeba času CPU).
b) Rutinně zpracovávané programy je třeba kompilovat s volbou OPT=2.
c) Pro pořízení výpisu strojového kódu přeloženého programu za účelem jeho rozboru je možno použít volbu LIST.

I. ON - PODMÍNKY A PŘEDPONY

Není vhodné řešit situace,vzniklé v programu,pomocí ON-jednotek (ON SIZE,ON CONV,ON FIXEDOVERFLOW atd.).ON-jednotky je vhodné používat pouze pro havarijní situace programu.Výjimku tvoří ON ENDFILE pro zjištění konce souboru. U externích procedur je vhodné nepoužívat ON-jednotky.Havarijní situace lze řešit v ON ERROR v hlavní proceduře.

Není vhodné používat předpony NO pro podmínky,které jsou zavedeny a lze je vypustit(NOCONV,NOZDIV a pod.).

U rutinně zpracovávaných programů není vhodné používat ON-podmínek trvale nezavedených(SIZE,CHECK,SUBSCRIPTRANGE, STRINGRANGE).Jejich použití zvětšuje přeložený program a spotřebu času CPU.

K. OSTATNÍ DOPORUČENÍ.

- U frekventovaných programů není vhodné používat STREAM INPUT/OUTPUT operací, vůbec nepoužívat u programů, pracujících v režimu ON LINE, a u externích procedur.
- Je výhodné provádět otevírání a zavírání souborů (tam, kde je to možné), jedinou instrukcí OPEN nebo CLOSE.
- Zabudovaná funkce DATE by se měla používat v programu jen jednou mimo hlavní smyčku programu (stejně i TIME).
- Není vhodné používání BEGIN bloků.

L. ZÁVĚR.

Ve 4. příspěvku jsme se pokusili shrnout naše poznatky o používání programovacího jazyka PL/I s podporou PL/I(F) kompilátoru. Tento kompilátor se používá ve většině našich výpočetních středisek, všude tam, kde se používá programovací jazyk PL/I. Firma IBM však vyvinula nový, výkonnější kompilátor (OPTIMIZING COMPILER), který odstraňuje některé nedostatky dnes již zastaralého PL/I(F) kompilátoru. Protože však OPTIMIZING COMPILER není běžně k dispozici, považujeme za vhodné seznámit všechny uživatele programovacího jazyka PL/I s našimi poznatky. Nečiníme si však nárok na komplexní hodnocení jazyka, ani na úplnost probírané problematiky. Při složitosti systému kompilátoru nebylo v našich možnostech prozkoumat všechny varianty, které jazyk PL/I umožňuje. Budeme proto vděčni všem, kteří budou ochotni rozšířit, popřípadě korigovat, naše poznatky a tak přispět k racionálnímu využívání výpočetní techniky všude tam, kde se používá programovací jazyk PL/I.

M. SEZNAM LITERATURY.

[1] IBM SYSTEM/360 OPERATING SYSTEM PL/I(F) LANGUAGE REFERENCE MANUAL

[2] IBM SYSTEM/360 OPERATING SYSTEM PL/I(F) PROGRAMMER'S GUIDE

[3] IBM SYSTEM/360 OPERATING SYSTEM PL/I SUBROUTINE LIBRARY PROGRAM LOGIC MANUAL

[4] IBM SYSTEM/370 PRINCIPLES OF OPERATIONS

[5] IBM SYSTEM 370/145 FUNCTIONAL CHARACTERISTICS