NIEKTORÉ MOŽNOSTI NORMOVANÉHO PROGRAMOVANIA

Jan VINAŘ, prom.mat., Výpočtové stredisko VS KNV, Košice

Normované programovanie sa používa v našich výpočtových strediskách už značne dávno. Možno povedať, že jeho rozkvet u nás pripadá na začiatok 70. rokov. Neskôr ho začali vytláčať iné, progresívnejšie metodiky programovania.

Cieľom tohto príspevku je ukázať, že normované programovanie má ešte rad možností, ktoré sa oplatí využívať. V našom VS je normované programovanie (NP) stále hlavným používaným nástrojom; pokúsime sa uviesť niektoré u nás získané skúsenosti, ktoré možno rozdeliť do troch oblastí:

a) Úpravy kanonickej schémy NP
b) Niektoré otázky realizácie NP v jazyku PL/I
c) Možnosti aplikácie.

1. Úpravy kanonickej schémy

Pri projektovaní typovej štruktúry NP sme sa rozhodli zmeniť štruktúru bloku D (obr. 1). Hoci sme zachovali indikátory zmien úrovne (viď. tab.2), využívame tú zrejmú skutočnosť, že ak napr. došlo k zmene na úrovni 2 (ktoré sa spracovala procedúrou CZ1), ale nie na úrovni 3, musia sa vyvolať procedúry GH2 a GH1 v tomto poradí, takže môžeme priamo odskočiť doprostred "reťazca" procedúr GH. V bloku D sa nastavuje aj indikátor prvého priechodu QD (obr.1,2), ktorý riadi prácu bloku (pri prvom priechode) sa vykoná spracovanie záhlaví všetkých úrovní). Indikátory zmien úrovní sa zachovávajú, nie však pre riadenie práce bloku D, ale pre prípadné využitie v bloku E (odlišné spracovanie prvej vety skupiny).

2. Realizácia v jazyku PL/I

a) Vzhľadom na časovú náročnosť vyvolávania procedúr (aj vnútorných) sa všetky štandardné bloky umiestňujú na štandardných návestiach (viď.obr. 2). Návestie pre návrat z každého bloku je síce určené, ale pre uľahčenie práce programátora sa väčšina blokov končí príkazom GO TO NP_RET, kde NP_RET je programom ošetrovaná premenná typu LABEL. Toto riadené použitie príkazu GO TO prináša niektoré výhody, o ktorých budeme hovoriť neskôr.

b) Špecifika deklaračných príkazov pre štruktúry si vynútila celkom odlišný systém deklarácií kľúčových oblastí. Príklad vidíme na

obr. 4, kde sa deklarujú kľúčové oblasti pre dvojúrovňový program. Využívame nielen atribút DEFINED, ale aj BASED pre kľúče novej vety. To má dva vedľajšie účinky:

- zrýchlenie výberu ďalšej vety v bloku C (viď obr. 5),
- možnosť ukončenia bloku E príkazom SN='0', ktorý uvoľní pre spracovanie vždy správny súbor.

c) V rade prípadov je výhodné používať normovaný program ako procedúru, ktorá postupne číta vstupný súbor a vracia do hlavného programu napr. súčty jednotlivých úrovní. V danej štruktúre si to vyžaduje minimálne zmeny:
   - úpravu začiatku bloku A podľa obr. 3,
   - deklarovanie všetkých potrebných dát s atribútom STATIC

   V každom mieste programu môžeme potom príkaz GO TO NP_RET nahradiť príkazom RETURN. Pri novom vyvolaní procedúry bude indikátor prvého prechodu nastavený na '1'B a po nastavení ON-podmienok pre vstupné súbory sa okamžite pokračuje na správnej adrese (t.j. na NP_RET).

3. Možnosti aplikácie

Sústredíme sa tu na dve oblasti, kde použitie NP nie je celkom bežné: nahrávacie a kontrolné chody a aktualizácia nesekvenčných súborov.

3.1. Nahrávacie a kontrolné chody

Problémy, ktoré tu vznikajú, si ukážeme na príklade. Majme program na nahrávanie výkazov, v ktorom
- každý výkaz začína vetou záhlavia s údajmi o vykazujúcej jednotke,
- každá položková veta obsahuje o.i. číslo časti výkazu,
- dáta vstupujú z rôznych médií (DP, DŠ, KP ai)

Prvý problém - ako v rámci NP riešiť spracovanie vety záhlavia - ilustruje obr. 6. Jedným z kľúčových polí je poradové číslo výkazu. Pri načítaní vety záhlavia vetu spracujeme (t.j. vyberieme z nej údaje), IV zvýšime o 1 a prejdeme na čítanie novej vety (bez okľúky cez blok E). Ďalšia veta potom vyvolá zmenu na úrovni výkazu.

Obr. 7 ukazuje riešenie druhého problému. Pre každé vstupné médium vytvoríme konverzný modul, ktorý napĺňa údajmi pracovný disk. Z neho sa potom údaje postupne čítajú a odovzdávajú - veta po vete - vlastnému programu. Synchronizáciu oboch častí programu (korutín) obstarávajú premenné návestia L_F1 a L_F2.

Poznámka: Použitím vonkajšej procedúry, ktorá by pri každom vyvolaní začala pracovať tam, kde predtým prestala, by bolo možné sa zobrať bez pracovného disku. Napriek tomu pokusy ukázali, že toto riešenie je v daných podmienkach pomalšie.

3.2. Aktualizácia nesekvenčných súborov

Vstupným súborom je tu súbor opravných viet, kľúčom kľúč vety. Predpokladá sa, že opravné vety sú zotriedené podľa kľúča a v jeho rámci podľa poradia, v akom vstúpili do systému.

Pri otvorení nového kľúča (obr. 8a) zistíme stav kľúča (Ø-veta nie je v aktualizačnom súbore, 1-veta je v aktualizovanom súbore a načíta sa do pamäte, 2-veta bola zrušená, ale fyzicky v súbore je), ktorým obsadíme premenné ALOK a ALOKØ.

Pri spracovaní opravnej vety testujeme a meníme obsah ALOK podľa obr. 8b. Konečne pri zmene kľúča (obr. 8c) použijeme obsah ALOK a ALOKØ na rozhodnutie, či zapísať novú vetu alebo prepísať už existujúcu.

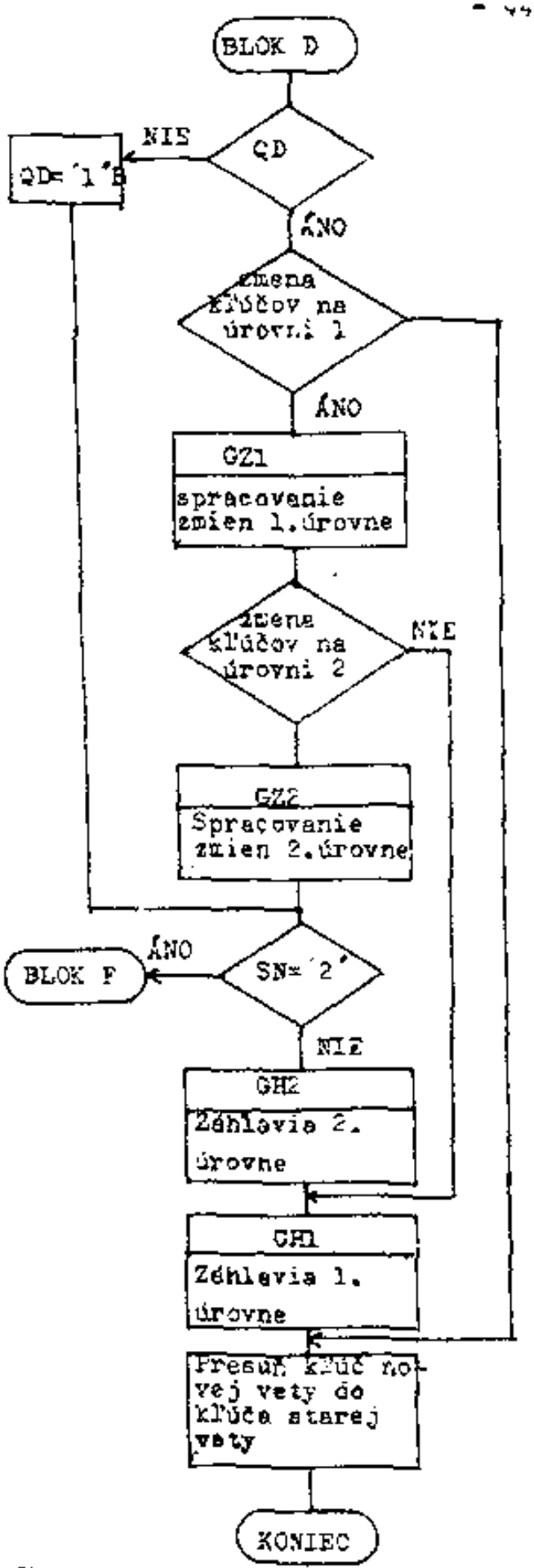
Jednou z výhod takto vybudovaného programu (okrem prehľadnosti a ľahkej údržby) je možnosť sledovania zmien na vyšších úrovniach napr. pre tlač protokolov.

4. Technológia programovania

Pre generovanie textu normovaného programu (aj pre iné rozšírenia základného jazyka) používame generátor zdrojových programov na báze jazyka Assembler. Použitím štyroch makroinštrukcií vygeneruje sa text s patričnými odskokmi na návestia, na ktoré programátor dopíše svoje bloky. Ide o otvorený systém v tom zmysle, že ktokoľvek môže doplniť svoje makrodefinície. Z toho dôvodu neobsahuje generátor NP napr. deklarácie a spracovanie súm. Kto ich potrebuje, môže si príslušné makrodefinície doplniť.

5. Záver

V príspevku tohto rozsahu nebolo možné v podstate ani jeden z uvedených problémov osvetliť do podrobností. To však ani nebolo jeho cieľom. Ak sa nám podarilo ukázať, že NP ešte nepatrí do starého softwaru, je to to, čo sme chceli.



Obr.1. Kanonická schéma bloku D pre 2 úrovne.

Typ BIT(1) - výhybky

QD - je nastavený, ak sme už prešli blokom D

QR_i - na úrovni i došlo k zmene

Typ LABEL

LA_iD - spracovanie zmien i-tej úrovne

L_iD - záhlavie i-tej úrovne

L4B_j - očakok po prečítaní vety zo súboru č. j

L_jE - spracovanie vety súboru č.j

NF_RET - adresa návratu z jednotlivých blokov

Typ BIN FIXED

NF_LEVL - číslo úrovne, ktorej zmeny sa práve spracúvajú

Typ CHAR(1)

SN, SS, S1 - stavy nového súboru, starého súboru, súboru č.1

Obr. 2. Dôležité premenné

```
BLOK_A:
     ON ENDFILE (subor č.1) GO TO LEOF_1;
                    :
     ON ENDFILE (subor č.k) GO TO LEOF_K;
     IF QD THEN GO TO NP_RET;
     pokračovanie bloku A
```

Obr. 3. Začiatok bloku A

```
DECLARE
    1 KL1 STATIC,
      2 S1 CHAR(1),
      2 A1 PIC '(2)9',
      2 B1 CHAR(3),
      2 C1 CHAR(1),
    K_Ø1 CHAR(7) DEF KL1,
    1 KL2 STATIC,
      2 S2 CHAR(1),
      2 A2 PIC '(2)9',
      2 B2 CHAR(3),
      2 C2 CHAR(1),
    K_Ø2 CHAR(7) DEF KL2,
```

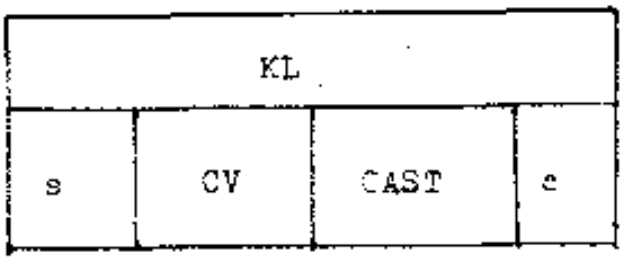
Obr. 4a) Kľúčové oblasti súborov.

```
1 KLS STATIC,
  2 SS CHAR(1),
  2 AS PIC '(2)9',
  2 BS CHAR(3),
  2 CS CHAR(1),
K_ØS CHAR(7) DEF KLS,
K_1S CHAR(6) DEF KLS,
K_2S CHAR(3) DEF KLS;
DECLARE
  UKN POINTER STATIC,
  1 KLN BASED(UKN),
    2 SN CHAR(1),
    2 AN PIC '(2)9',
    2 BN CHAR(3),
    2 CN CHAR(1),
  K_ØN CHAR(7) BASED(UKN),
  K_1N CHAR(6) BASED(UKN),
  K_2N CHAR(3) BASED(UKN);
```

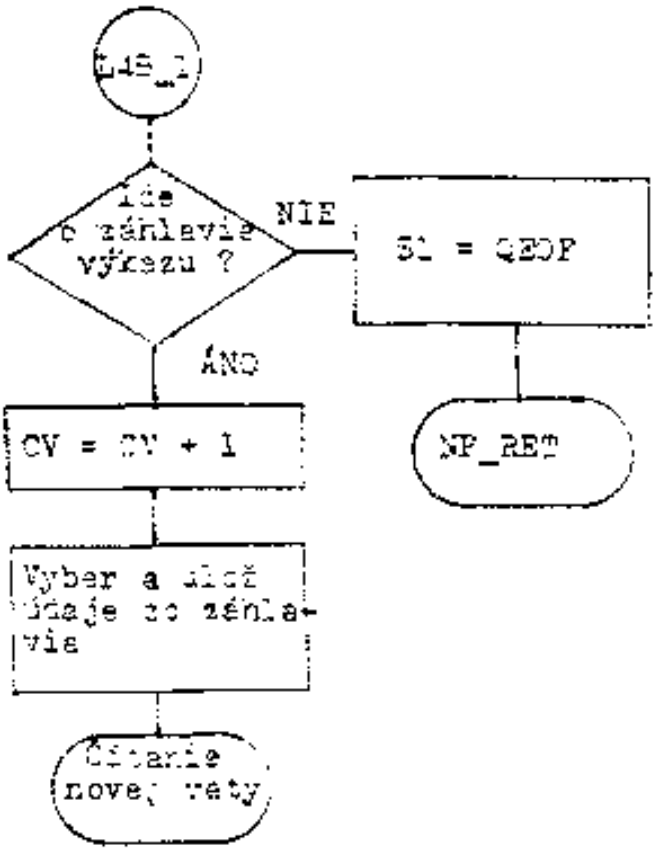
Obr. 4b) Kľúčové časti starej a novej vety.

```
BLOK_C:
   UKN = ADDR(K_Ø1);
   IF K_ØN   K_2N THEN UKN=ADDR(K_Ø2);
   GO  TO BLOK_D;
```

Obr. 5. Výber kľúča

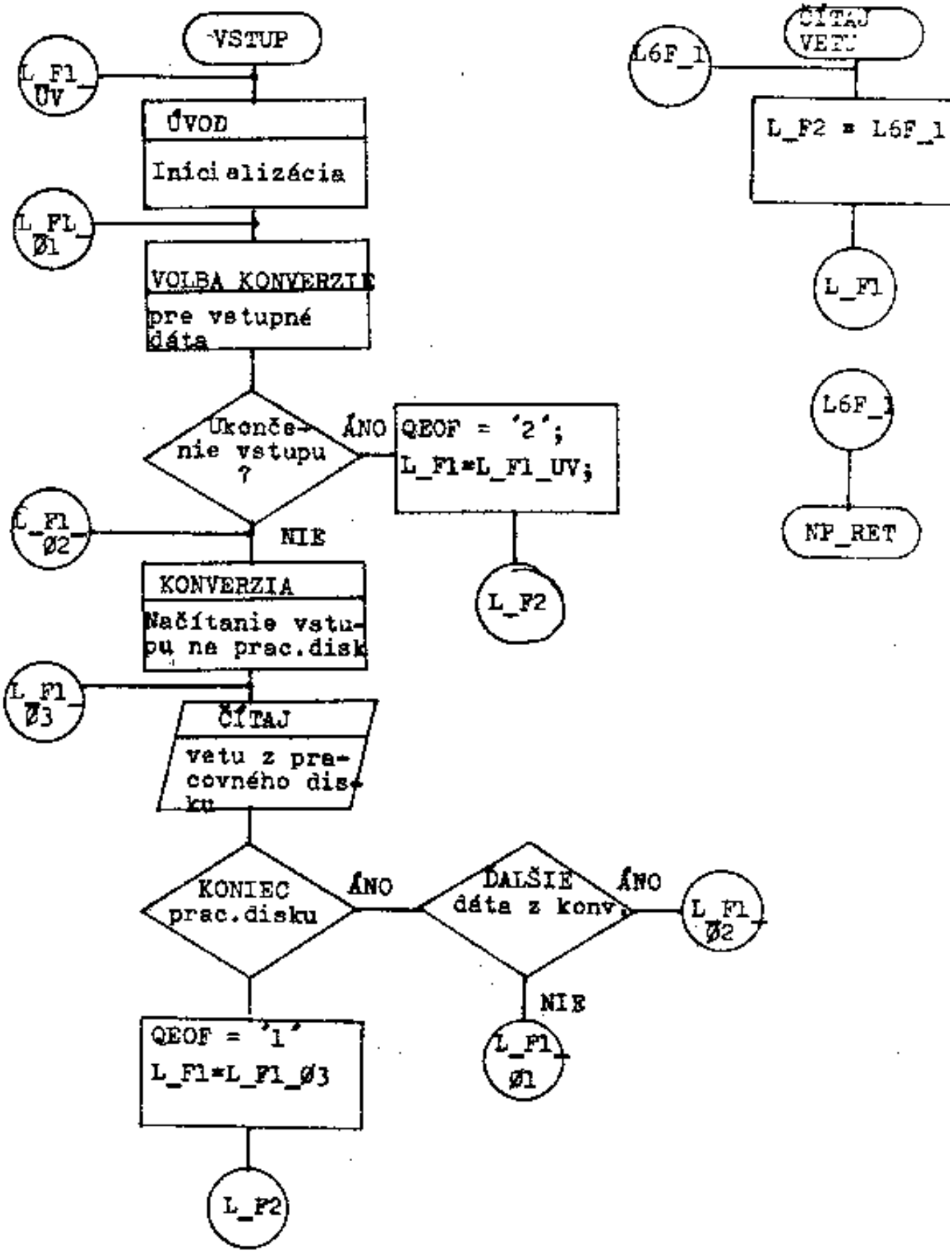


a) Štruktúra kľúčovej oblasti
   CV - poradové číslo výkazu



b) Čítanie vety

Obr. 6. Riešenie vstupu nahrávaných viet.



Obr. 7. Mechanizmus vstupu dát.

	R1	R2	R3
je veta s daným kľúčom v súbore ?	N	Y	Y
je platná ?	-	N	Y
ALOKØ ALOK	Ø	2	1

a) Zistenie počiatočného stavu (GH1)

TYP OPRAVY	Test ALOK	Zmena ALOK
Nové veta	≠ 1	1
zmena	= 1	1
zrušenie	= 1	0

b) Testovanie a zmeny ALOK v bloku B

ALOKØ / ALOK	Ø	1	2
Ø	-	prepíš	prepíš
1	zapíš	prepíš	prepíš

c) Zápis konečného stavu v bloku GZ1.

Obr. 8. Aktualizácia nesekvenčných súborov.