Interpretace

Práce interpretu je obdobou práce základního cyklu procesoru

Registr instrukcí RI
Čítač instrukcí PC

Základní cyklus:

```
do {
   RI ← PROGRAM [ PC ];
   PC ← PC + 1;
   switch (RI.INSTRUCTION_CODE) {
      case INSTRUCTION_CODE_1: STATEMENTS_OF_INSTRUCTION_CODE_1;
      case INSTRUCTION_CODE_2: STATEMENTS_OF_INSTRUCTION_CODE_2;
                    :
      case INSTRUCTION_CODE_n: STATEMENTS_OF_INSTRUCTION_CODE_n;
   }
} while (RI.INSTRUCTION_CODE ≠ STOP);
```

Zavedme postfixové instrukce pro výpočty s integer

TA take address do zásob., parametrem je adresa (hladina, posun)
TC take constant do zásob., parametrem je hodnota
DR dereference vrcholu zásobníku,
ST store, obsah vrcholu ulož na adresu pod vrcholem,
JU jump, parametrem je adresa
IFJ if false jump, parametrem je adresa, vrchol je F=0 nebo T≠0
PLUS,
MINUS,
TIME,
DIV,
NEG změna znaménka,
AND,
OR,
NOT,
REL typ je dán parametrem (LT, LE, EQ, GE, GT, NE,
OD test lichosti,
READ čte do adresy na vrcholu zásobníku,
WRITE tiskne obsah vrcholu zásobníku ,
CSUB skok do podprogramu,
PAR parametrem je adresa skutečného parametru,
BBEG vstup do pp, vytvoří AZ, parametry hladina a velikost,
FPAR formální parametr, parametrem je VAR nebo CONST
RET, návrat z pp, likvidace jeho AZ
STOP konec výpočtu

Výpočtový zásobník je integer pole Z[1 .. MAXZ];

Aktivační záznam bude mít tvar:

```
     T   ---->    pomocne promenne
                  Lokalni promenne
                  parametry                                  4
                  hladina                                    3
                  stara hodnota  DISPLAY [ level ]           2
                  dynamicky ukazatel                         1
     B   ---->    navratova adresa                           0
```

```
program INTERPRET;
 konstanty     MAXP = ...; /*max. délka programu*/
               MAXZ = ...; /*max. hloubka zásobníku*/
               MAXD = ...; /*max. velikost displeje*/
 typy  DPI = (TA, TC, DR, ST, JU IFJ, PLUS, MINUS,
              TIME, DIV, NEG, AND, OR, NOT, REL, OD, READ, WRITE,
              CSUB, PAR, BBEG, FPAR, RET, STOP);
       TPI = struktura (diskriminant IC typu DPI) {
         když IC je (DR, ST, PLUS, MINUS, TIME, DIV, NEG,
                AND, OR, NOT, OD, READ, WRITE, RET, STOP)pak  ();
         když IC je (TA, PAR) pak (N typu 1..MAXD;P typu 0..MAXZ);
         když IC je (TC)      pak (K typu integer);
         když IC je (JU, IFJ, CSUB) pak (I typu 0..MAXP);
         když IC je (REL)  pak (RO typu (LT, LE, EQ, GE, GT, NE));
         když IC je (BBEG) pak (H typu 1..MAXD; L typu integer);
         když IC je (FPAR) pak (V typu (CONST, VAR))
            };
 proměnné PROGRAMS typu array[0..MAXP] prvky typu TPI;
          Z        typu array[0..MAXZ] prvky typu integer;
          DISPLAY  typu array[1..MAXD] prvky typu 0..MAXZ;
          PC      typu 0..MAXP;
          B,T,TP  typu 0..MAXZ;
          RI      typu TPI;

 procedure READPROGRAM;
           /*načte postfixové instrukce do pole PROGRAMS*/

/*hlavni program*/
{  READROGRAM; T ← 0; PC ← 0; /* v PROGRAMS[0] je skok na první
                              vykonávanou instrukci    */
```

Programový text základního cyklu interpretu

```
}
```

```
do
 RI ← PROGRAMS[ PC ]; PC ← PC+1;
 switch RI.IC {
   case TA    :{ T←T+1; Z[T]←DISPLAY[RI.N] + RI.P };
   case TC    :{ T←T+1; Z[T]←RI.K };
   case DR    :Z[T] ← Z [ Z [ T ] ] ;
   case ST    :{ Z[Z[T-1]] ← Z[T]; T←T-2 };
   case JU    :PC ← RI.I;
   case IFJ   :{ if Z[T]=0 then PC←RI.I; T←T-1 }:
   case PLUS  :{ Z[T-1]←Z[T-1] + Z[T]; T←T-1; };
   case MINUS :{ Z[T-1]←Z[T-1] - Z[T]; T←T-1; };
   case TIME  :{ Z[T-1]←Z[T-1] * Z[T]; T←T-1; };
   case DIV   :{ Z[T-1]←Z[T-1] div Z[T]; T←T-1; };
   case NEG   :Z[T] ← -Z[T];
   case AND   :{ Z[T-1]←Z[T-1]*Z[T]; T←T-1 };
   case OR    :{ Z[T-1]←konv_int((Z[T-1]=1)or(Z[T]=1));T←T-1;};
   case NOT   :if Z[T]=0 then Z[T]←1 else Z[T]←0;
   case REL   :{ switch RI.RO {
                    case LT: Z[T-1] ← konv_int(Z[T-1]< Z[T]);
                    case LE: Z[T-1] ← konv_int(Z[T-1]<=Z[T]);
                    case EQ: Z[T-1] ← konv_int(Z[T-1]= Z[T]);
                    case GE: Z[T-1] ← konv_int(Z[T-1]>=Z[T]);
                    case GT: Z[T-1] ← konv_int(Z[T-1]> Z[T]);
                    case NE: Z[T-1] ← konv_int(Z[T-1]<>Z[T]);
                                 };
                 T ← T-1 ;
         };
   case OD    :Z[T] ← konv_int(odd(Z[T]));
   case READ  :{ read(Z[Z[T]]); T←T-1 };
   case WRITE :{ write(Z[T]); T←T-1 };
   case CSUB  :{ T←T+1; Z[T]←PC; PC←RI.I; TP←T+4 };
   case PAR   : ;
   case BBEG  :{ Z[T+1]←B; Z[T+2]←DISPLAY[RI.H]; B←T;
                 DISPLAY[RI.H]←B; Z[T+3]←RI.H; T←T+RI.L
         };
   case FPAR  :{ case RI.V of
                   VAR  :Z[TP]←DISPLAY[ PROGRAMS[ Z[ B ]].N]
                                + PROGRAMS[ Z[ B ] ].P;
                   CONST:Z[TP]←Z[ DISPLAY[ PROGRAMS[ Z[ B ]].N]
                                  +PROGRAMS[ Z[ B ]].P];
                 };
                 Z[ B ] ← Z[ B ] + 1; TP ← TP + 1;
         };
   case RET   :{ DISPLAY[Z[B+3]] ← Z[B+2]; T←B-1;
                 PC ← Z[ B ]; B ← Z[ B+1];
         };
   case STOP  : ;
 }
while RI.IC ≠ STOP;
```

Př.

```
{ /*program s rekurzivním vnořeným podprogramem                    } Hladina 1
     integer m, n, k;                                               }

     podprogram NSD(integer i, j);

     {                                                              }
          while i <> j do                                           }
                                                                    } hladina 2
               if i > j  then i =  i - j                            }
                         else j = j – i;                            }
          k = i;                                                    }

     };                                                             }


     read(m); read(n);                                              }
     if (m > 0) and (n > 0) then                                    } Hladina1
                                { NSD(m, n); write(k);              }
                                }                                   }
}
```

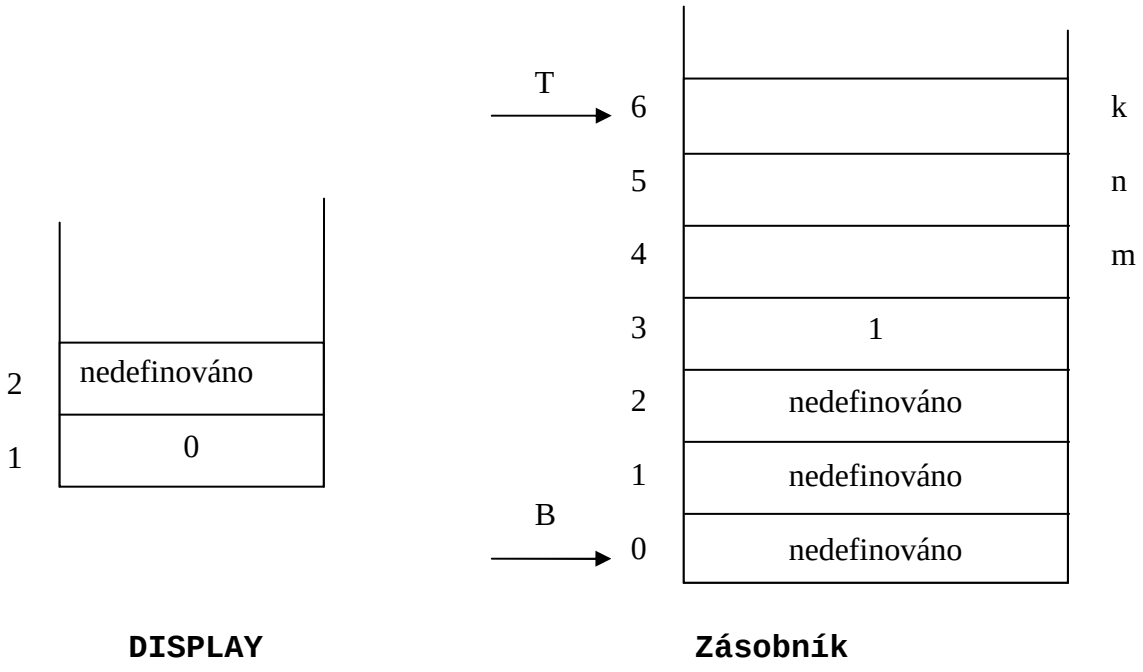
Jméno proměnné	hladina	posun
m	1	4
n	1	5
k	1	6
i	2	4
j	2	5

Program přeložený do postfixových instrukcí

```
(0)  JU   37
(1)  BBEG 2,5   začátek deklarace
(2)  FPAR CONST   procedury NSD
(3)  FPAR CONST
(4)  TA   2,4     začátek while
(5)  DR
(6)  TA   2,5     i ≠ j
(7)  DR
(8)  REL  NE
(9)  IFJ  32      výskok while
(10) TA   2,4     začátek if
(11) DR
(12) TA   2,5     i > j
(13) DR
(14) REL  GT
(15) IFJ  24
(16) TA   2,4
(17) TA   2,4
(18) DR
(19) TA   2,5     i ← i - j
(20) DR
(21) MINUS
(22) ST
(23) JU   31
(24) TA   2,5
(25) TA   2,5
(26) DR
(27) TA   2,4     j ←j-i
(28) DR
(29) MINUS
(30) ST           konec if
(31) JU   4       konec while
(32) TA   1,6
(33) TA   2,4      k ←i
(34) DR
(35) ST           konec dekl.
(36) RET          proc.NSD
(37) BBEG 1,6
(38) TA   1,4
(39) READ         read(m)
(40) TA   1,5
(41) READ         read(n)
(42) TA   1,4     začátek if
(43) DR
(44) TC   0       m > 0
(45) REL  GT
(46) TA   1,5
(47) DR
(48) TC   0       n > 0
(49) REL  GT
(50) AND
(51) IFJ  58
(52) CSUB 1
(53) PAR  1,4     NSD(m,n)
(54) PAR  1,5
(55) TA   1,6
(56) DR           write(k)
(57) WRITE        konec if
(58) STOP
```

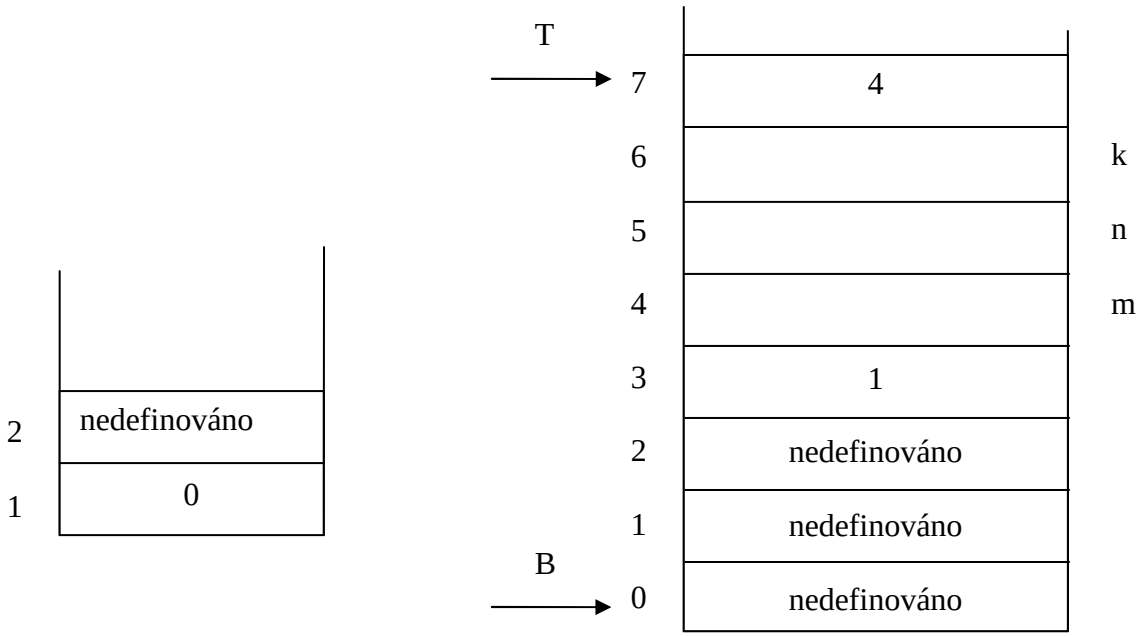
Budeme předpokládat čtení hodnot 60 a 90

Na začátku v Hlavním progr. T = 0 PC = 0
Stav po provedení instrukce (37) BBEG 1,6



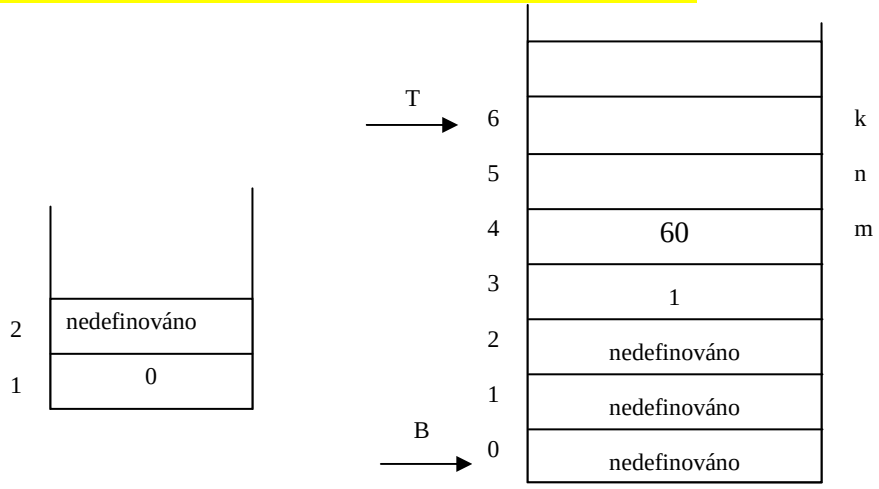
DISPLAY **Zásobník**

Stav po provedení instrukce (38) TA 1,4



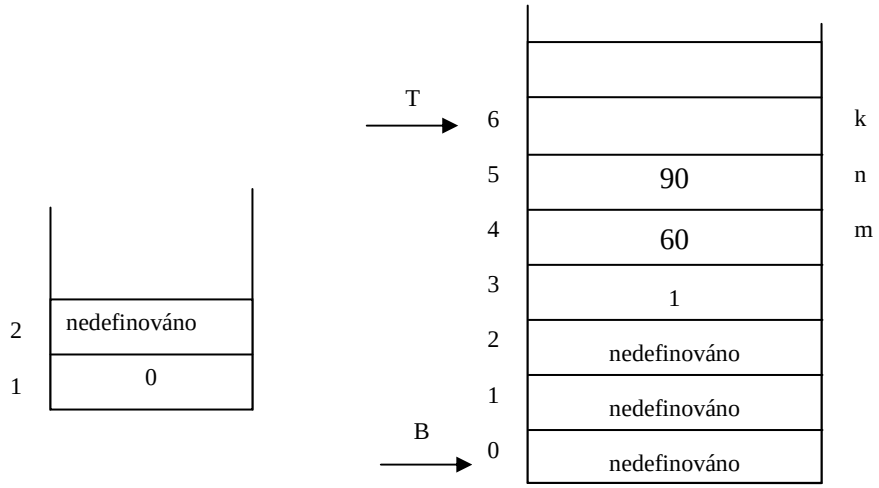
DISPLAY **Zásobník**

Stav po provedení instrukce (39) READ



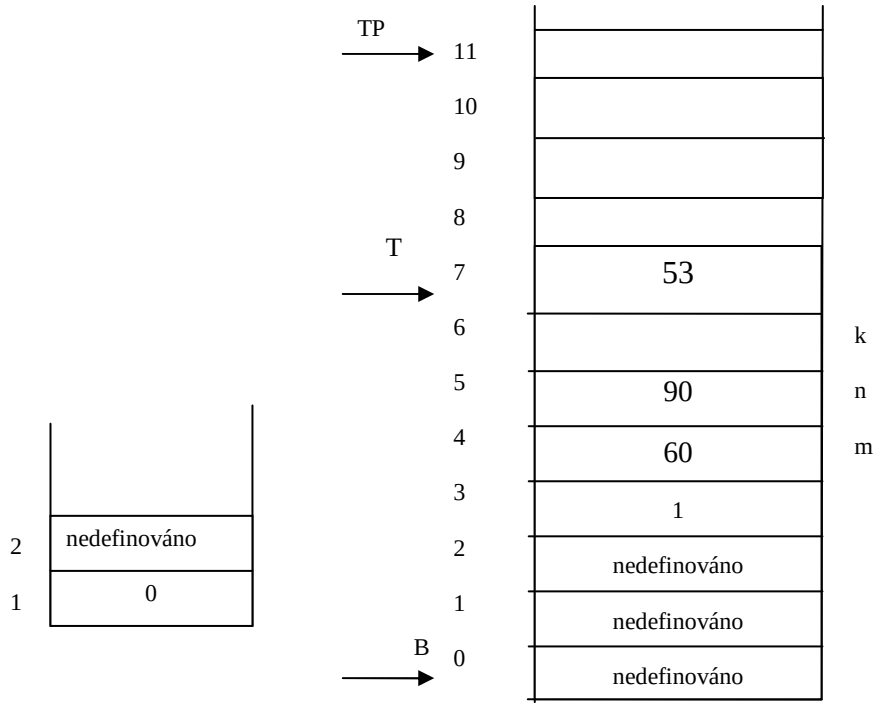
DISPLAY **Zásobník**

Stav po provedení instrukce (40) TA 1, 5
(41) READ



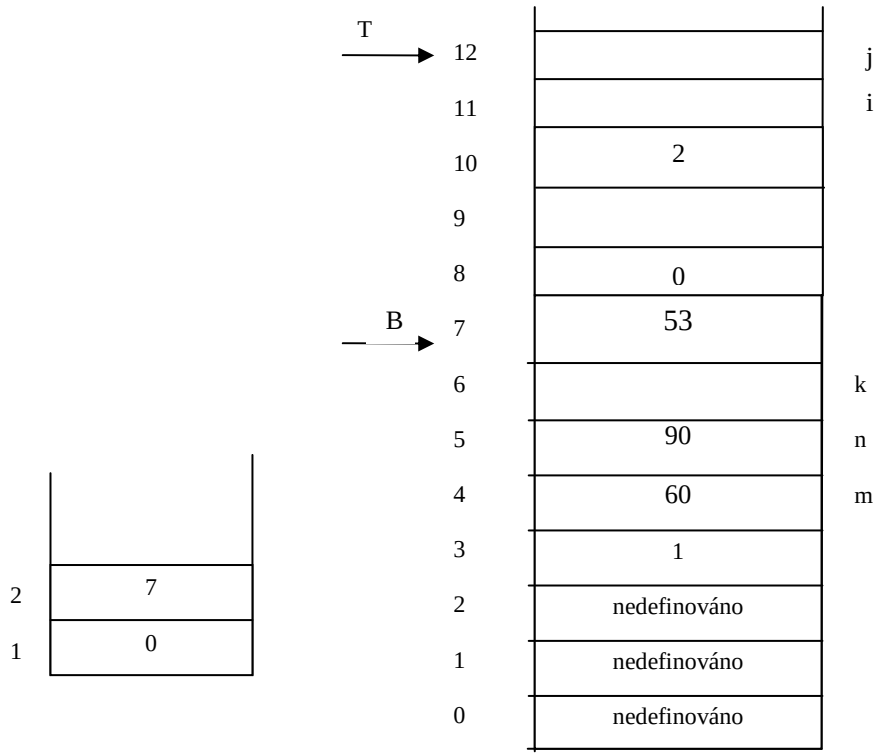
DISPLAY **Zásobník**

Stav po provedení instrukce (52) CSUB 1



PC = 1

Stav po provedení instrukce (1) BBEG 2, 5



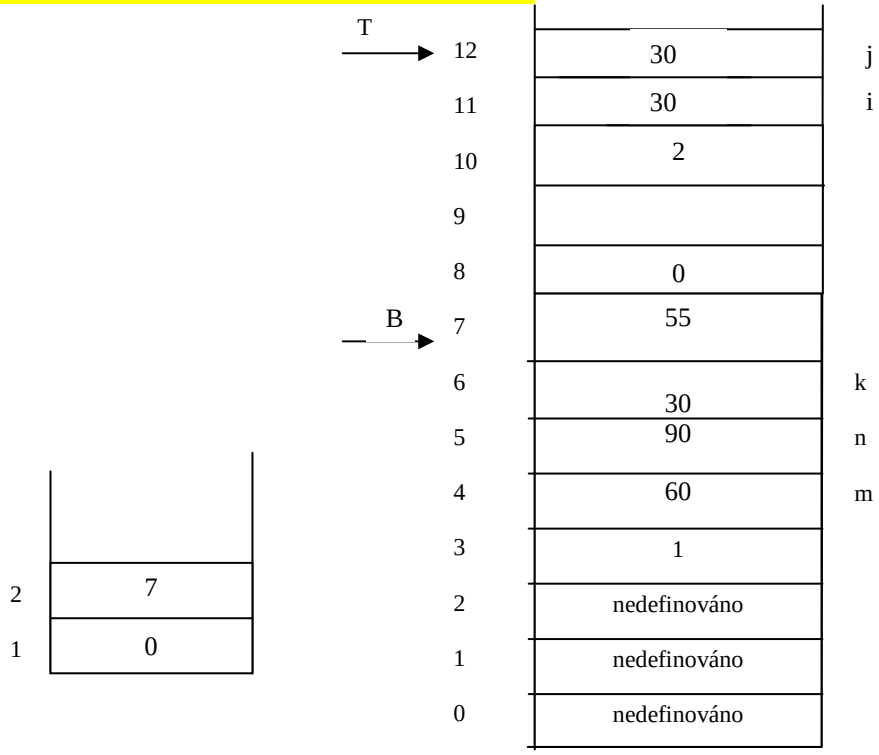
Zpracování dalších instrukcí (z podprogramu NSD)

Instrukce	Změna v Z	Další změny
(2) FPAR CONST	$Z[11] := 60$ $Z[7] := 54$	$TP := 12$
(3) FPAR CONST	$Z[12] := 90$ $Z[7] := 55$	$TP := 13$
(4) TA 2,4	$Z[13] := 11$	$T := 13$
(5) DR	$Z[13] := 60$	
(6) TA 2,5	$Z[14] := 12$	$T := 14$
(7) DR	$Z[14] := 90$	
(8) REL NE	$Z[13] := 1$	$T := 13$
(9) IFJ 32		$T := 12$
(10) TA 2,4	$Z[13] := 11$	$T := 13$
(11) DR	$Z[13] := 60$	
(12) TA 2,5	$Z[14] := 12$	$T := 14$
(13) DR	$Z[14] := 90$	
(14) REL GT	$Z[13] := 0$	$T := 13$
(15) IFJ 24		$T := 12$ $PC := 24$
(24) TA 2,5	$Z[13] := 12$	$T := 13$
(25) TA 2,5	$Z[14] := 12$	$T := 14$
(26) DR	$Z[14] := 90$	
(27) TA 2,4	$Z[15] := 11$	$T := 15$
(28) DR	$Z[15] := 60$	
(29) MINUS	$Z[14] := 30$	$T := 14$
(30) ST	$Z[12] := 30$	$T := 12$
(31) JU 4		$PC := 4$
(4) TA 2,4	$Z[13] := 11$	$T := 13$
(5) DR	$Z[13] := 60$	
(6) TA 2,5	$Z[14] := 12$	$T := 14$
(7) DR	$Z[14] := 30$	

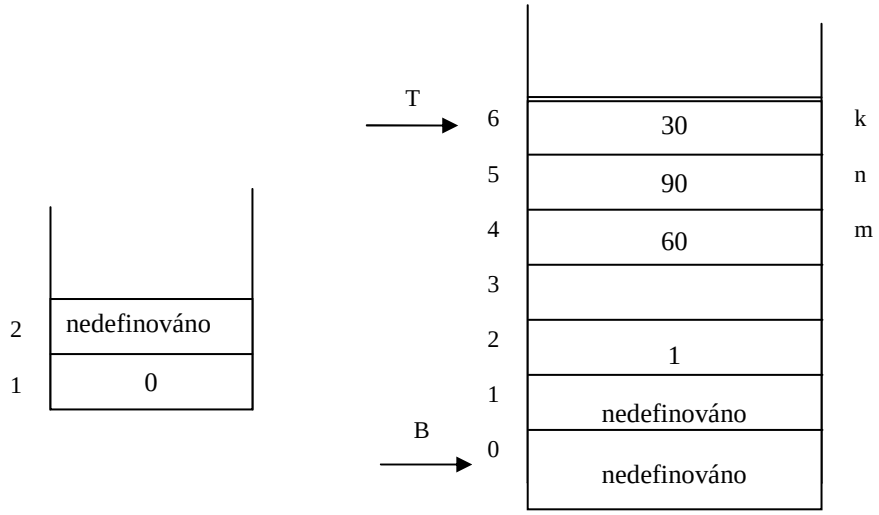
Instrukce	Změna v Z	Další změny
(8) REL NE	$Z[13] := 1$	$T := 13$
(9) IFJ 32		$T := 12$
(10) TA 2,4	$Z[13] := 11$	$T := 13$
(11) DR	$Z[13] := 60$	
(12) TA 2,5	$Z[14] := 12$	$T := 14$
(13) DR	$Z[14] := 30$	
(14) REL GT	$Z[13] := 1$	$T := 13$
(15) IFJ 24		$T := 12$
(16) TA 2,4	$Z[13] := 11$	$T := 13$
(17) TA 2,4	$Z[14] := 11$	$T := 14$
(18) DR	$Z[14] := 60$	
(19) TA 2,5	$Z[15] := 12$	$T := 15$
(20) DR	$Z[15] := 30$	
(21) MINUS	$Z[14] := 30$	$T := 14$
(22) ST	$Z[11] := 30$	$T := 12$
(23) JU 31		$PC := 31$
(31) JU 4		$PC := 4$
(4) TA 2,4	$Z[13] := 11$	$T := 13$
(5) DR	$Z[13] := 30$	
(6) TA 2,5	$Z[14] := 12$	$T := 14$
(7) DR	$Z[14] := 30$	
(8) REL NE	$Z[13] := 0$	$T := 13$
(9) IFJ 32		$T := 12$ $PC := 32$
(32) TA 1,6	$Z[13] := 6$	$T := 13$
(33) TA 2,4	$Z[14] := 11$	$T := 14$
(34) DR	$Z[14] := 30$	
(35) ST	$Z[6] := 30$	$T := 12$

Tabulka . Další provádění programu

Stav před návratem z procedury



Stav po provedení instrukce (36) RET



Instrukce	Změna v Z	Další změny
(55) `TA` 1, 6	$Z[7] := 6$	$T := 7$
(56) `DR`	$Z[7] := 30$	
(57) `WRITE`		$T := 6$; tiskne hodnotu 30
(58) `STOP`		ukončí interpretaci

Tabulka Instrukce ukončující interpretaci

Interpretace v PL0

Instrukce postfixového zápisu

lit 0,A **uloz konstantu A do zasobniku**

opr 0,A **proved instrukci A**

1	**unarni minus**
2	**+**
3	**-**
4	*****
5	**div**
6	**mod**
7	**odd**
8	**=**
9	**<>**
10	**<**
11	**>=**
12	**>**
13	**<=**

lod L,A **:ulož hodnotu proměnné z adr. L,A na vrchol zásobníku**

sto L,A **:zapiš do proměnné s adr. L,A hodnotu z vrcholu zásob.**

cal L,A **:volej proceduru s adresou kódu A z úrovně L**

ret 0,0 **:return**

ing 0,A **:zvyš obsah Top-registru zásobníku o hodnotu A**

jmp 0,A **:proveď skok na adresu kódu A**

jpc 0,A **:proveď podmíněný skok na adresu kódu A**

```
/* interpretace generovanych kodu PL0. S je vypoctovy zasobnik, program je v code[ ]*/
void interpret(void) {
int p, t;                   /* citac instrukci,  vrchol zasobniku */
INSTRUCTION i;      /*registr instrikce*/

   printf("START PL/0\n");
   t = p = s[1] = s[2] = s[3] = 0;  /*vrchol, citac, stat.retez, dynamo.retez, navrat.adresa*/
   b = 1;
   do {
      i = code[p++];

      switch (i.f) {
         case lit: s[++t] = i.a;
                  break;
         case opr:
            switch (i.a) {
               case neg : s[t] = -s[t];
                                       /*printf("INTERPRET OPR: neg\n");*/
                                       break;
               case add : t--;
                        s[t] += s[t + 1];
                                       /*printf("INTERPRET OPR: add\n");*/
                                       break;
               case sub : t--;
                                       s[t] -= s[t + 1];
                                       /*printf("INTERPRET OPR: sub\n");*/
                                       break;
               case mul : t--;
                                       s[t] *= s[t + 1];
                                       /*printf("INTERPRET OPR: mul\n");*/
                                       break;
               case di :  t--;
                                       if (s[t + 1] != 0) s[t] /= s[t + 1];
                        else {
                           // error
                           error(31);
                        }
                        /*printf("INTERPRET OPR: di\n");*/
                        break;
               case mod : t--;
                                       s[t] = s[t] % s[t + 1];
                                       /*printf("INTERPRET OPR: mod\n");*/
                                       break;
               case odd : s[t] = s[t] % 2;
                                       /*printf("INTERPRET OPR: odd\n");*/
                                       break;
               case eq : t--;
                                 s[t] = (s[t] == s[t + 1]);
                                       /*printf("INTERPRET OPR: eq\n");*/
                        break;
               case ne : t--;
                                 s[t] = (s[t] != s[t + 1]);
                                       /*printf("INTERPRET OPR: ne\n");*/
                        break;
               case lt : t--;
```

```
                        s[t] = (s[t] < s[t + 1]);
                            /*printf("INTERPRET OPR: lt\n");*/
                    break;
                case ge : t--;
                        s[t] = (s[t] >= s[t + 1]);
                            /*printf("INTERPRET OPR: ge\n");*/
                        break;
                case gt : t--;
                        s[t] = (s[t] > s[t + 1]);
                            /*printf("INTERPRET OPR: gt\n");*/
                    break;
                case le : t--;
                        s[t] = (s[t] <= s[t + 1]);
                            /*printf("INTERPRET OPR: le\n");*/
                        break;

                }
                break;

        case lod: t++;   /*natazeni adresy promenne do stacku*/
                    s[t] = s[base(i.l) + i.a];        /*fce base provede sestup o l urovni po stat.retezu*/
                    break;        /* v PL0 je dynam.adresa (hlad.pouziti minus hlad.deklarace, posuv)

        case sto: s[base(i.l) + i.a] = s[t];
                    printf("%d\n",s[t--]);
                    break;

        case cal: s[t + 1] = base(i.l);     /*staticky retezec*/
                    s[t + 2] = b;           /*dynamicky retezec*/
                    s[t + 3] = p;           /*navratova adresa*/
                    b = t + 1;              /*nova baze*/
                    p = i.a;                /*zacatek podprogramu*/
                break;

        case ret: t = b - 1;
                    p = s[t + 3];           /*do p dame navratovou adresu*/
                    b = s[t + 2];           /*nastavime starou bazi*/
                    break;

        case ing: t += i.a;         /*a je velikost AZ = 3+pocet promennych*/
                break;

        case jmp: p = i.a;
                break;

        case jpc: if (s[t] == 0) p = i.a;   /*skok při false*/
                    t--;
                    break;

    }
  } while (p);
  printf(" END PL/0\n");

}   // interpret()
```

Generátor kódu

1. Ze čtveřic

Máme k dispozici:

-Jeden obecný registr - akumulátor a jeho instrukční množinu: LOAD addr, STORE addr, ADD addr, SUB addr, MUL addr, …, CH. (CH je zezápornění).

-Glob.prom. ACCUM uchová jméno proměnné, jejíž hodnota je v akumulátoru

Ke generování použijeme podprogramy:

1)

```
podprogram Store_into_accumulator( P,Q: typu variable) {
  T: typu variable;
 if (ACCUM ≠ P)   { /*ACCUM je globální proměnná obsahující údaj co j e ve střadači*/
    if (ACCUM = undefined) { GEN('LOAD', P); ACCUM ← P;
    }
      else
      if (ACCUM = Q)  { T ← P; P ← Q; Q ← T ;
      }
   else
    { GEN('STORE', ACCUM);   GEN('LOAD', P);  ACCUM ← P;
     }
   }
}
```

čtveřice (+, OP1, OP2, Result) dtto všechny komutativní operace

2)

```
podprogram GADD(OP1, OP2, Result);  {
        Store_into_accumulator(OP1, OP2);
        Gen('ADD', OP2);
        ACCUM ← Result;
}
```

čtveřice (-, OP1, OP2, Result) dtto všechny nekomutativní operace

3)

```
podprogram GSUB(OP1, OP2, Result); {
        Store_into_accumulator(OP1, OP1);
        Gen('SUB', OP2);
        ACCUM ← Result;
}
```

(@, OP1, Result, -) unární minus

4)

```
podprogram GUN(OP1, Result); {
        Store_into_accumulator(OP1, OP1);
        Gen('CH', - ); ACCUM ← Result;
}
```

Př. generovování z posloupnosti čtveřic uděláme na tabuli (pohodáři jej najdou na konci)

2. Generování z trojic popíšeme rozhodovací tabulkou COMP

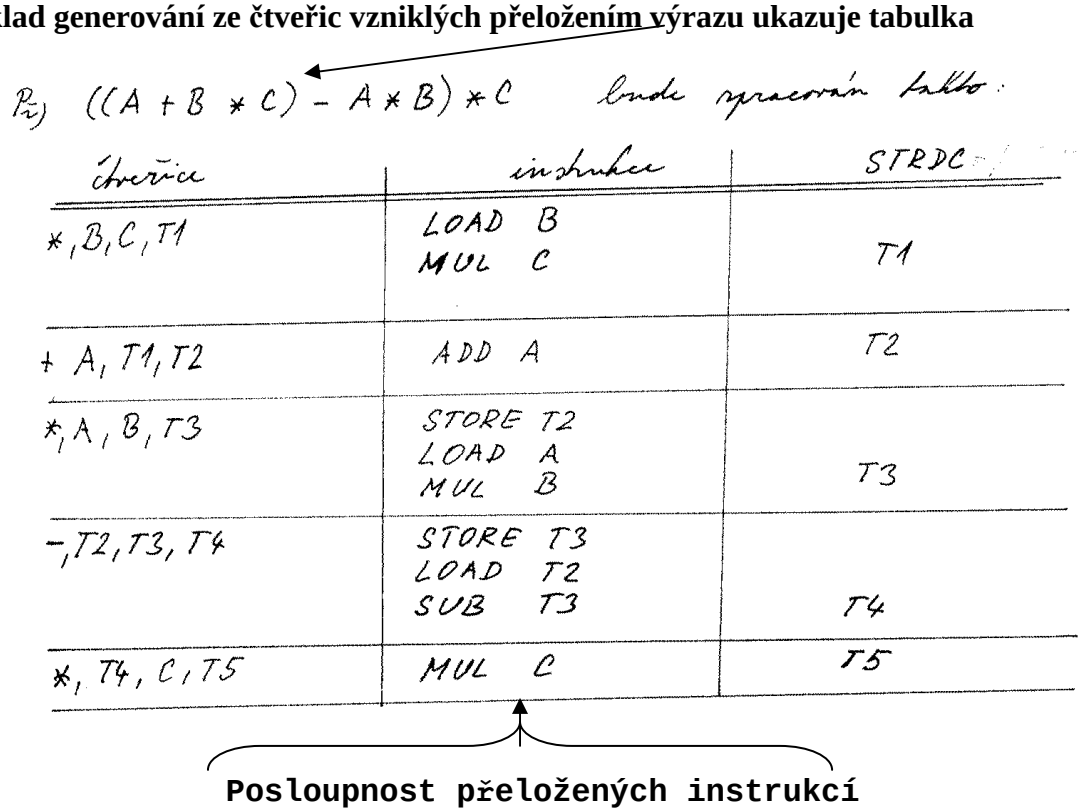
operator	op1 \ op2	accumulator	proměnná	trojice
+	accumulator		GEN('ADD',OP2)	T← NTV GEN('STORE',T) COMP(OP2) GEN('ADD',T)
	proměnná	GEN('ADD',OP1)	GEN('LOAD',OP1) GEN('ADD',OP2)	COMP(OP2) GEN('ADD',OP1)
	trojice		COMP(OP1) GEN('ADD',OP2)	COMP(OP1) OP1← accumulator COMP(Self)
-	accumulator		GEN('SUB',OP2)	
	proměnná	T← NTV GEN('STORE',T) OP2←T COMP(Self)	GEN('LOAD',OP1) GEN('SUB',OP2)	COMP(OP2) T← NTV GEN('STORE',T) OP2←T COMP(Self)
	trojice	T← NTV GEN('STORE',T) COMP(OP1) GEN('SUB',T)	COMP(OP1) GEN('SUB',OP2)	COMP(OP2) OP2← accumulator COMP(Self)
un -		GEN('CH',' ')	GEN('LOAD'OP2) GEN('CH',' ')	COMP(OP2) GEN('CH',' ')

Pozn.:

T← NTV symbolizuje generování „new temporary variable“ a vložení jejího jména(tj. adresy) do proměnné T.

Př. generovování z posloupnosti trojic uděláme na tabuli

Příklad generování ze čtveřic vzniklých přeložením výrazu ukazuje tabulka



Příklad generování z trojic přeložených z výrazu A*(B+C) – B*(A+C)

Posloupnost trojic je:

(1) +, B, C
(2) *, A, (1)
(3) +, A, C
(4) *, B, (3)
(5) -, (2), (4)

Generátor se spustí vyvoláním KOMP(číslo_poslední_trojice)

Průběh výpočtu postupným voláním KOMP a v ni specifikovaných akcí pro konkrétní trojice se snaží zachytit následující obr.

Př.
A*(B+C)-B*(A+C) přeloženo do trojic má tvar
(1): +,B,C
(2) *,A,(1)
(3): +,A,C
(4): *,B,(3)
(5): -,(2),(4)
Generování začne vždy od poslední trojice

(5): -,(2),(4)
Volá KOMP(4) (4): ***,B,(3)**
Volá KOMP(3) (3): +,A,C
GEN(„LOAD“, A)
GEN(„ADD“, C)
Návrat do (4)
GEN(„MUL“,B)
Návrat do (5)
OP2 ←**accumulator**, tzn modifikuje (5)
Volá KOMP(modif.5) **(mod 5): -,(2), accumulator**
T_1 ← NTV vytvoří novou pomocnou proměnnou
GEN(„STORE“, T_1)
Volá KOMP(2): (2) *,A,(1)
Volá KOMP(1) (1): +,B,C
GEN(„LOAD“,B)
GEN(„ADD“,C)
Návrat do (2)
GEN(„MUL“,A)
Návrat do (mod 5)
GEN(„SUB“, T_1)
Návrat z (mod5) do (5)
Konec