PROGRAMOVÉ STRUKTURY: PRINCIPY PŘEKLADU

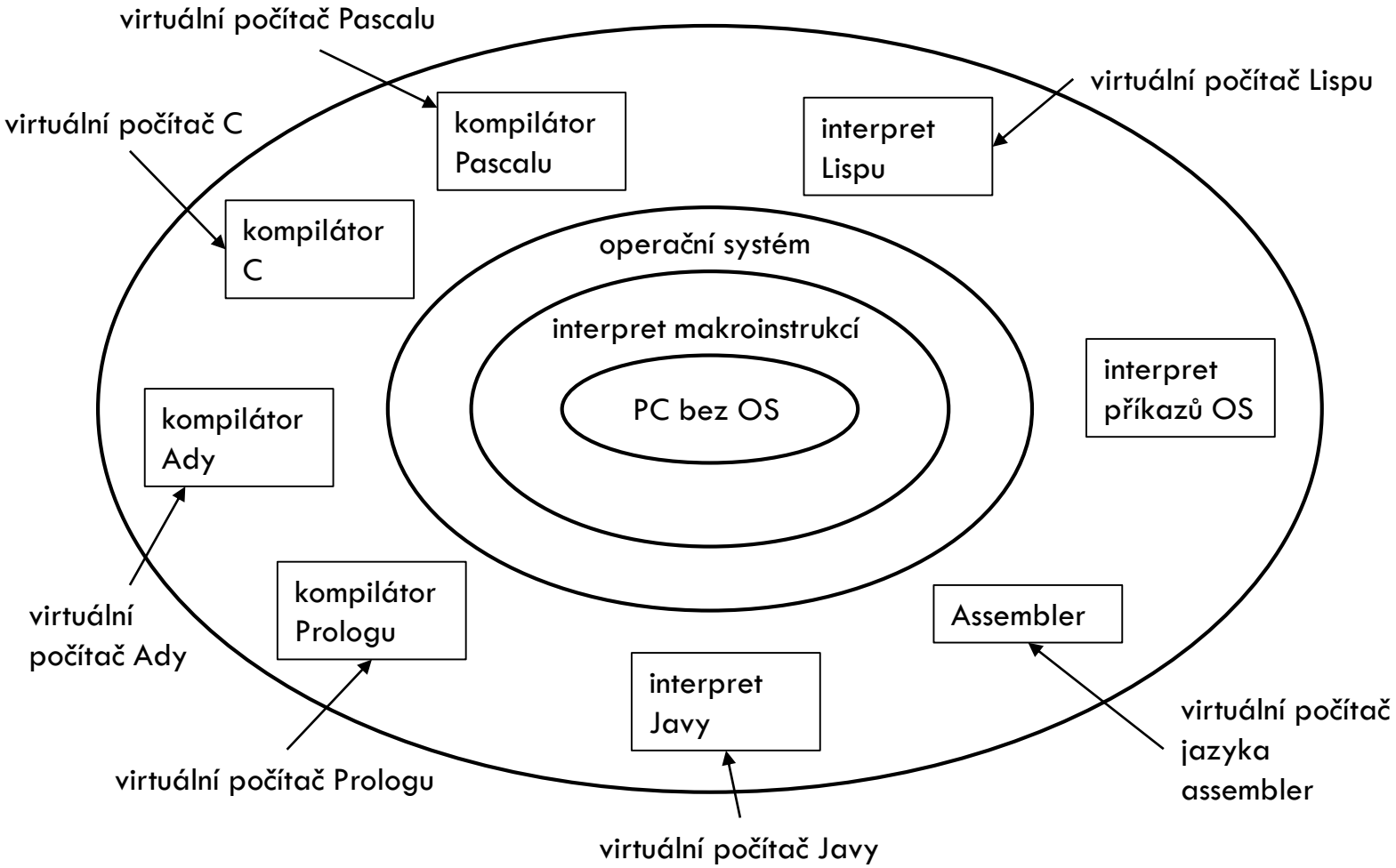
Překladač, kompiler, interpret, lexikální analýza, syntaktický analyzátor, sémantické zpracování, generování

© Karel Ježek & Martin Zíma

Virtuální počítač

- Hladiny
  - uživatelský program
  - překladač programovacího jazyka
  - operační systém
  - interpret makroinstrukcí
  - procesor

Virtuální počítač (2)

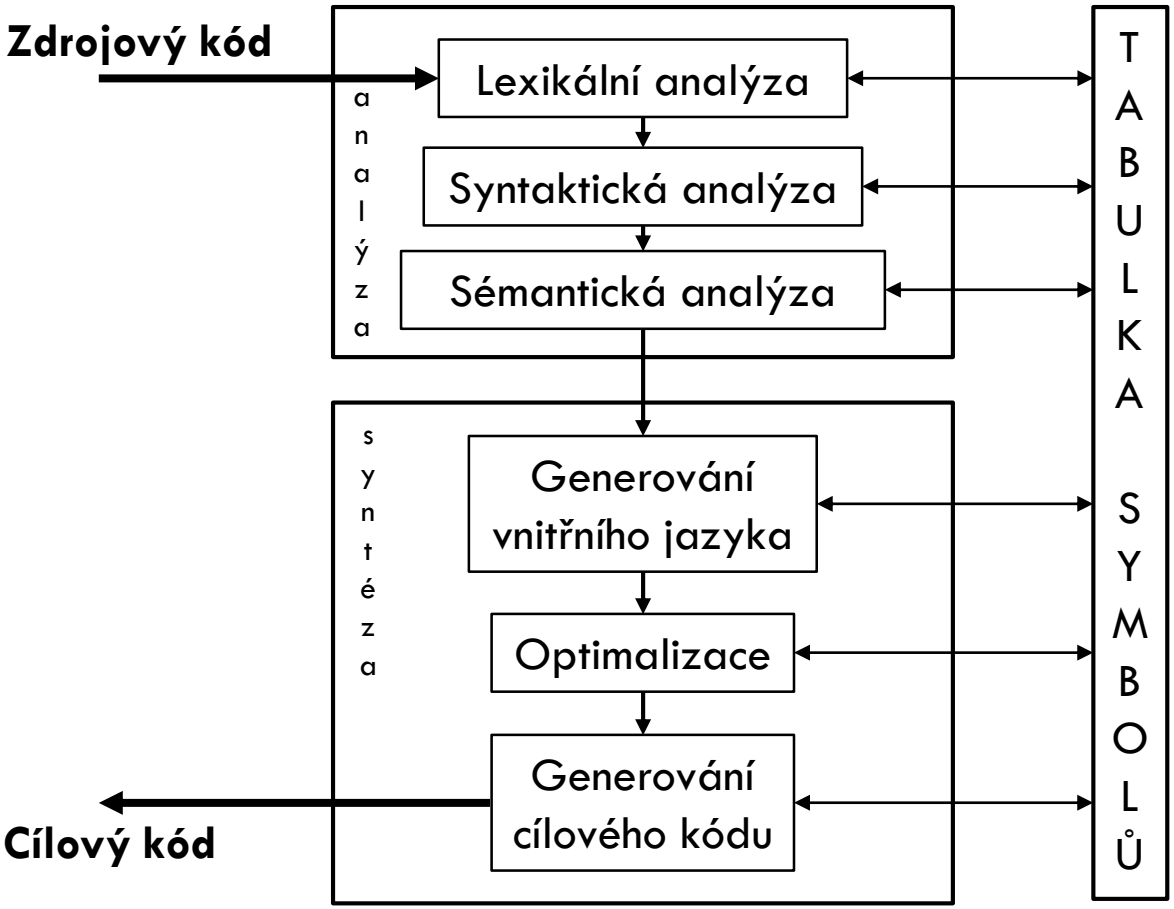


Překladač

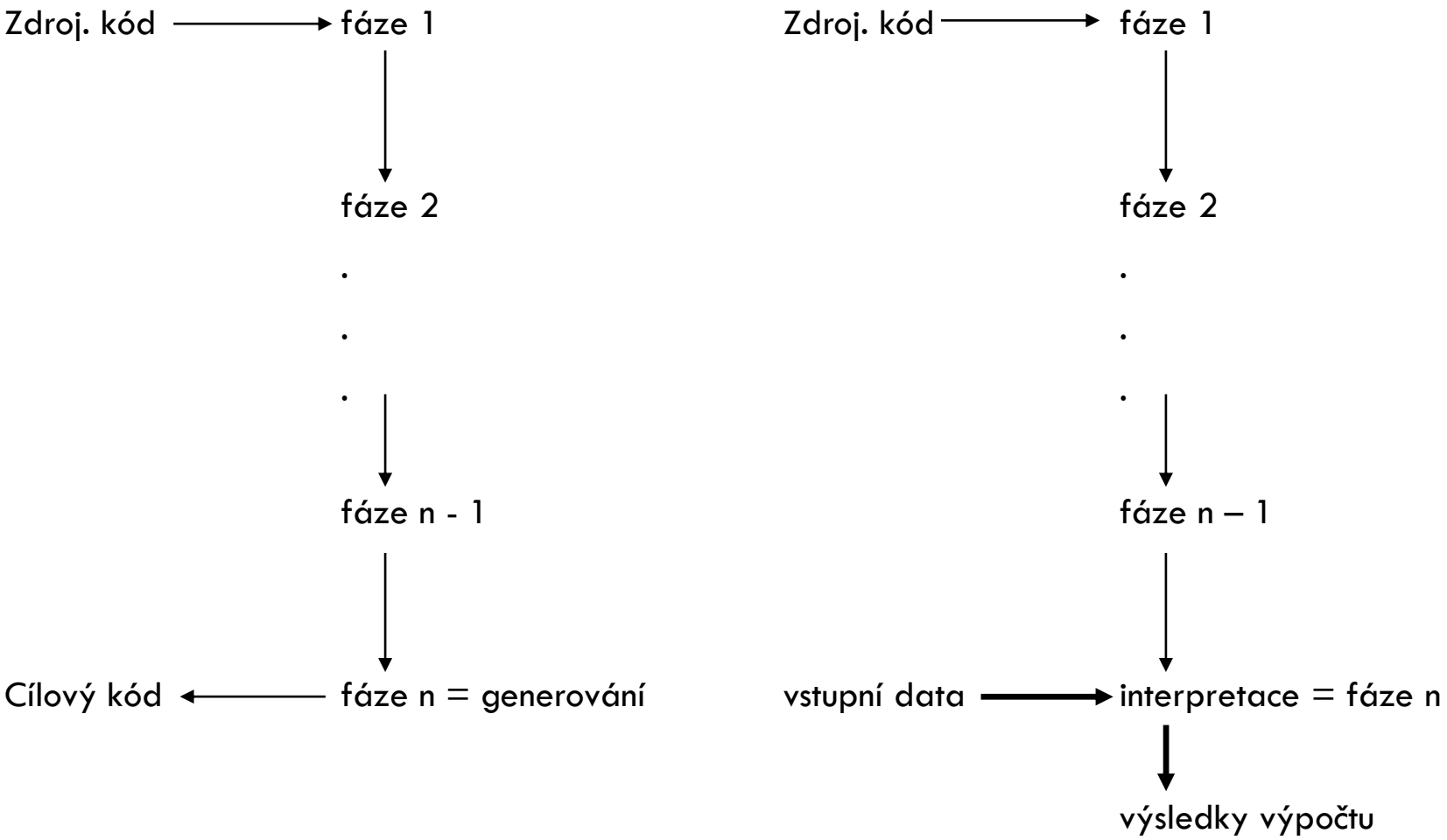
- Z formálního hlediska je překladač zobrazení:

  kompilátor: zdrojový jazyk cílový → jazyk
- Hlavní části kompilátoru
  - analytická část
    - lexikální analýza
      - rozpoznává symboly jazyka
    - syntetická a sémantická analýza
      - rozpoznává a kontroluje strukturu a kontextové závislosti programu
  - syntetická část
    - sémantické zpracování
      - převádí program do vnitřního jazyka překladače
    - generování cílového kódu
      - generuje cílový kód z vnitřního jazyka

Schéma překladače



Kompilátor vs. interpret



Interpret

- V čistě jen interpretační podobě se nepoužívá
  - neefektivní
  - od kompilátoru se liší poslední částí
- Analytická část
  - lexikální analýza
  - syntaktická a sémantická analýza
- Syntaktická část
  - sémantické zpracování
  - interpretace

Lexikální analýza

- Zakódování lexikálních elementů (do číselné podoby)
- Vynechávání komentářů
- Výhody
  - pevná délka symbolů pro další fáze zpracování
- Příklad

```
{ a = b * 3 + 5 ; /* poznamka */
  c = a + 1 ;
}
```

při kódování	ident	konst	+	*	=	{	}	;
na čísla	1	2	3	4	5	6	7	8

  - převede na: `6, 1 adr a, 5, 1 adr b, 4, 2 3, 3, 2 5, 8, 1 adr c, 5, 1 adr a, 8, 7`
    - tj. vnitřní jazyk lexikálního analyzátoru (mezijazyk)

Lexikální analýza (2)

- Tvary lexikálních symbolů jsou popsatelné regulárními gramatikami
- Příklady

```
<symbol>      ⟶ <identifikátor> |
                <číslo> |
                class | if | while | … |
                + | - | * | / | … |
                ( | ) | { | } | … |
                =+ | ++ | == | …
<identifikátor> ⟶ <identifikátor>a |
                  <identifikátor>b | … |
                  <identifikátor>0 | … |
                  <identifikátor>9 |
                  a | b | …
```

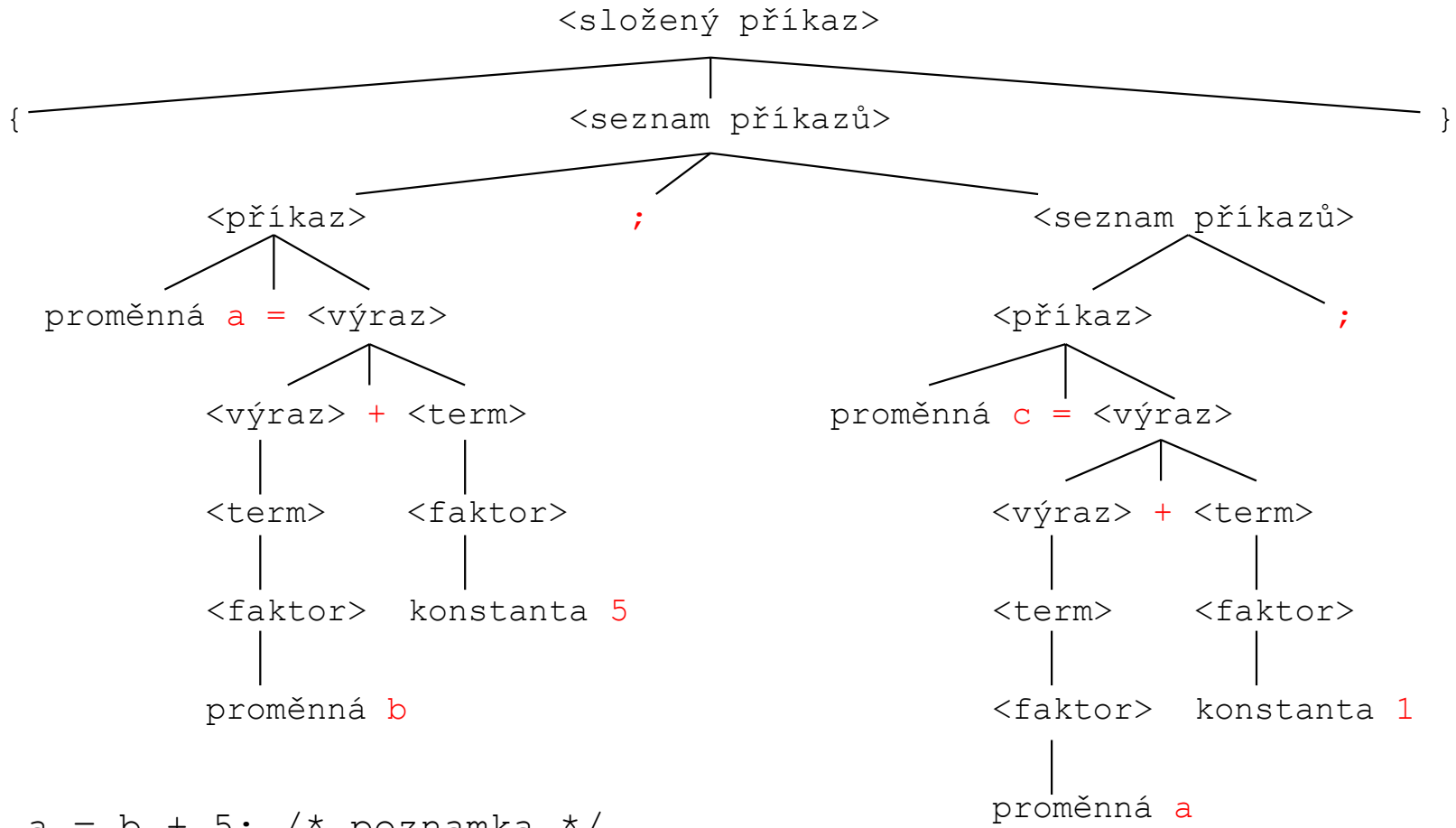
- Lexikální analyzátor je konečným automatem

Syntaktický analyzátor

- Zjišťuje strukturu překládaného textu (syntaktický strom)
- Je založen na bezkontextových gramatikách
  - dovolují popisovat vnořované závorkové struktury
- Vytváří derivační strom

```
<složený příkaz> → {<seznam příkazů>}                          (1)
<seznam příkazů> → <příkaz>;<seznam příkazů> |                 (2)
                   <příkaz>;                                   (3)
<příkaz> → <proměnná> = <výraz>                                (4)
<výraz> → <výraz> + <term> |                                   (5)
          <výraz> - <term> |                                   (6)
          <term>                                               (7)
<term> → <term> * <faktor> |                                   (8)
         <term> / <faktor> |                                   (9)
         <faktor>                                              (10)
<faktor> → (<výraz>) | proměnná | konstanta          (11) (12) (13)
```

Syntaktický analyzátor (2)




```
{ a = b + 5; /* poznamka */
  c = a + 1;
}
```

Syntaktický analyzátor (3)

- Struktura je zachycena syntaktickým analyzátorem jako posloupnost pravidel při odvození tvaru programu z počátečního symbolu gramatiky
- Možnosti
  - levá derivace
    - rozepisuje vždy nejlevější neterminální symbol
    - 1, 2, 4, 5, 10, 12, 13, …
  - pravá derivace
    - rozepisuje vždy nejpravější neterminální symbol
    - 1, 2, 3, 4, 5, 10, …

Sémantické zpracování

- Souběžně či následně s rozpoznáváním syntaktické struktury jsou vyvolávány sémantické akce (sdružené s každým z gramatických pravidel), které převádí program do vnitřního jazyka překladače
- Formy vnitřních jazyků
  - postfixová notace
    - operátory následují za operandy
  - prefixová notace
  - víceadresové instrukce

Sémantické zpracování (2)

- **Příklad:** `a = (b + c) * (a + c);`

```
1     LOA a     ST        PLUS b    c    pom1
2     LOV b     LOA a     PLUS a    c    pom2
3     LOV c     MUL       MUL  pom1 pom2 pom3
4     PLUS      PLUS      ST   a    pom3
5     LOV a     LOV b
6     LOV c     LOV c
7     PLUS      PLUS
8     MUL       LOV a
9     ST        LOV c
```

Optimalizace

- Redukce počtu pomocných proměnných
- Eliminace nadbytečných přesunů mezi registry
- Optimalizace cyklů

- Před optimalizací

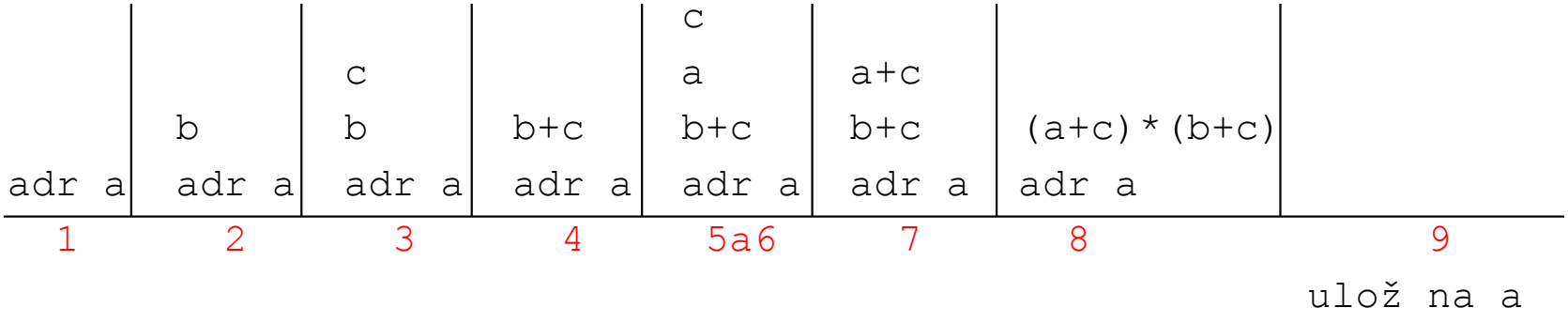
```
PLUS b    c    pom1
PLUS a    c    pom2
MUL  pom1 pom2 pom3
ST   a    pom3
```

Po optimalizaci

```
PLUS b    c    pom1
PLUS a    c    pom2
MUL  pom1 pom2 pom1
ST   a    pom1
```

Interpretace

`LOA a`	dej adresu operandu `a` na vrchol zásobníku
`LOV b`	dej obsah operandu `b` na vrchol zásobníku
`LOV c`	dej obsah operandu `c` na vrchol zásobníku
`PLUS`	sečti vrchol a podvrchol, výsledek vlož do zásobníku
`LOV a`	dej obsah operandu `a` na vrchol zásobníku
`LOV c`	dej obsah operandu `c` na vrchol zásobníku
`PLUS`	sečti vrchol a podvrchol, výsledek vlož do zásobníku
`MUL`	vynásob vrchol a podvrchol, výsledek vlož do zásobníku
`ST`	ulož hodnotu z vrcholu zásobníku na adresu uloženou pod vrcholem



Generování

- Logicky jednoduché
  - expanze makroinstrukcí vnitřního jazyka
- Prakticky komplikované
  - respektování instrukčních možností procesoru

- Víceadresové instrukce:

```
PLUS b    c    pom1
PLUS a    c    pom2
MUL  pom1 pom2 pom1
ST   a    pom1
```

Přeložený strojový kód:

```
MOV b,  R0
ADD c,  R0
STO R0, p1
MOV a,  R0
ADD c,  R0
MUL p1, R0
STO R0, a
```