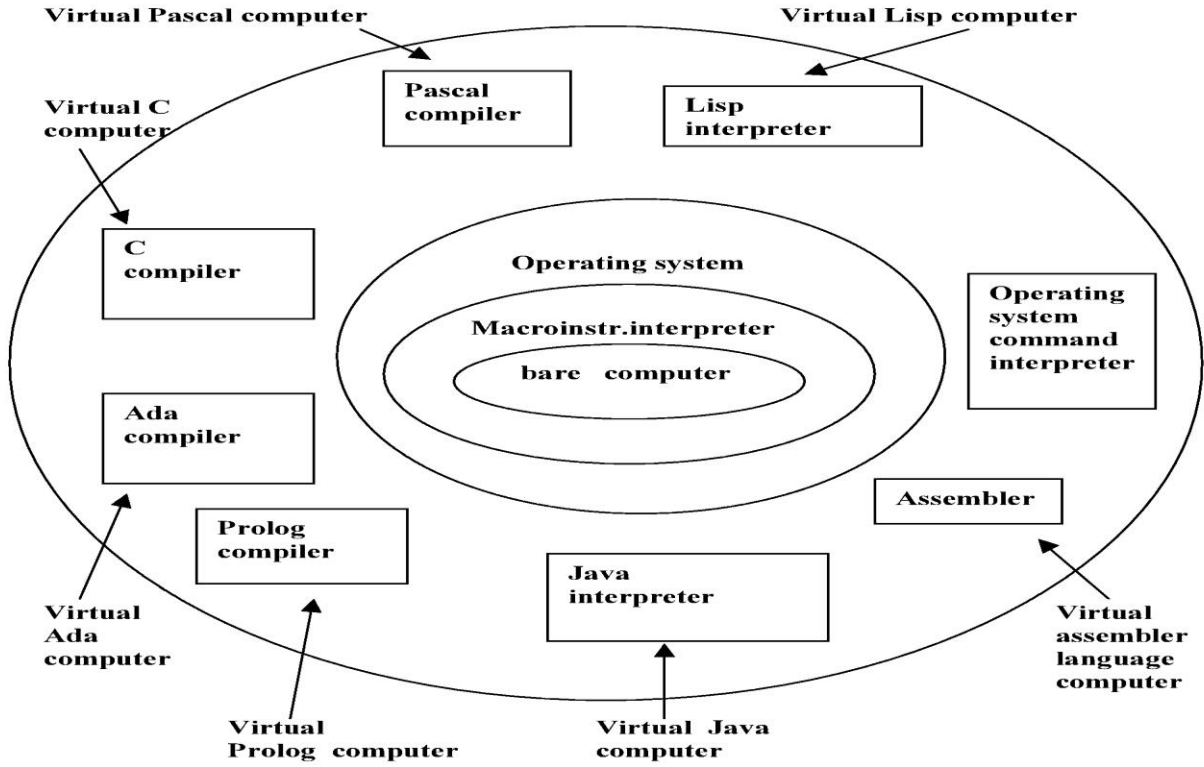


Virtuální počítač

Hladiny

- Uživatelský program
- Překladač programovacího jazyka
- Operační systém
- Interpret makroinstrukcí
- Procesor

Virtual computer



Překladač

Z formálního hlediska je překladač zobrazení

Kompilátor : Zdrojový jazyk → Cílový jazyk

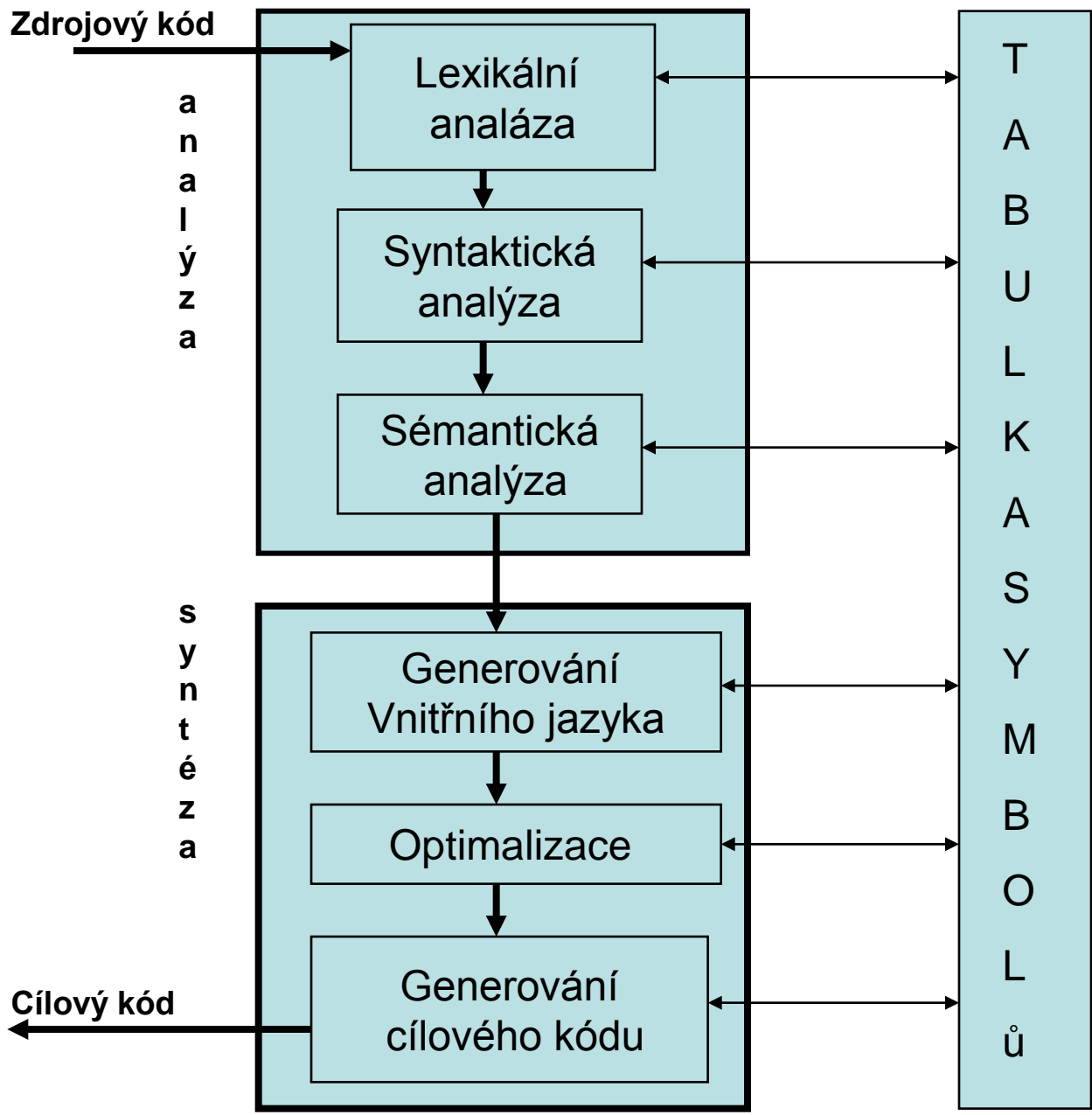
Jeho hlavní části tvoří

Analytická část:

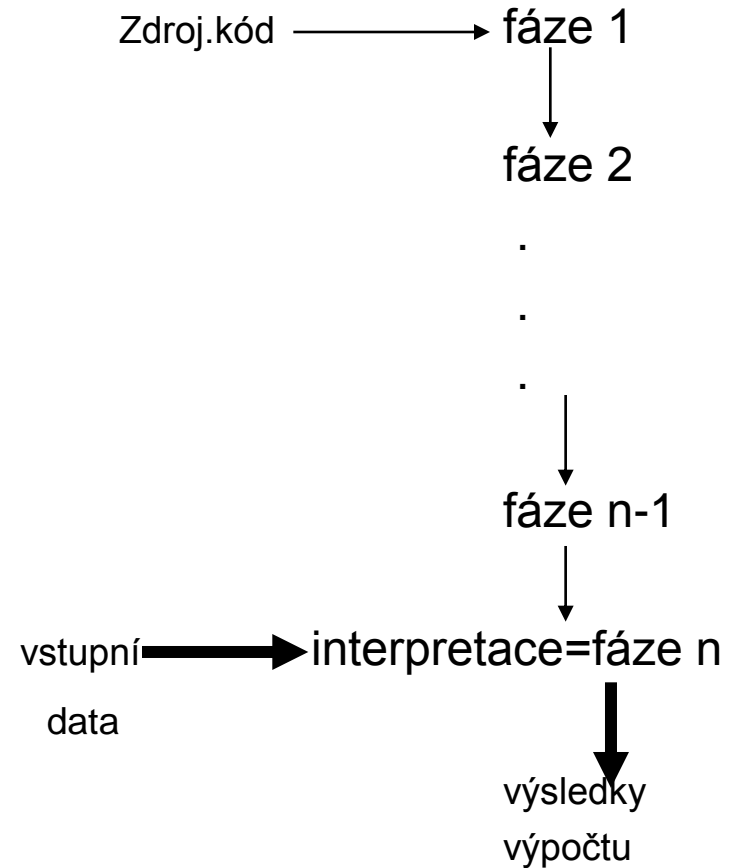
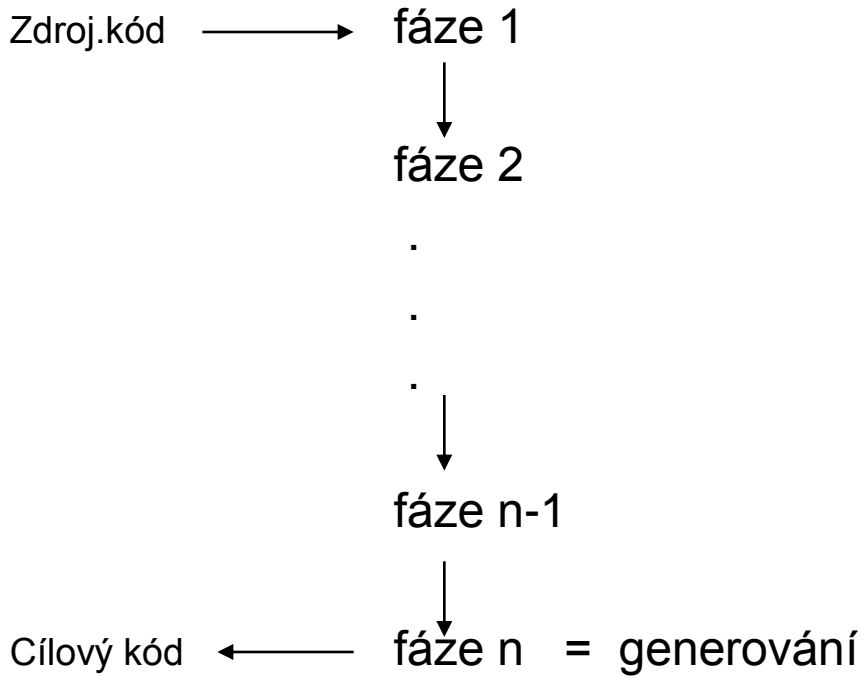
- Lexikální analýza –rozpoznává symboly jazyka
- Syntaktická a sémantická analýza –rozpoznává a kontroluje strukturu a kontextové závislosti programu

Syntetická část:

- Sémantické zpracování –převádí program do vnitřního jazyka překladače
- Generování cílového kódu -generuje cílový kod z vnitřního jazyka



Kompilátor | Interpret



Interpret

V čistě jen interpretační podobě se nepoužívá (neefektivní). Od kompilátoru se liší poslední částí

Analytická část:

- Lexikální analýza
- Syntaktická a sémantická analýza

Syntetická část:

- Sémantické zpracování
- Interpretace

Lexikální analýza

1. Zakódování lexikálních elementů (do číselné podoby)
2. Vynechávání komentářů

Výhody: pevná délka symbolů pro další fáze zpracování

```
Např. { a = b * 3 + 5 ; /* poznámka */  
      c = a + 1 ;  
      }
```

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

převéde na: 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

Tvary lexikálních symbolů jsou popsatelné regulárními gramatikami:

$\langle \text{symbol} \rangle \rightarrow \langle \text{identifikátor} \rangle \mid$
 $\langle \text{číslo} \rangle \mid$
 $\text{class} \mid \text{if} \mid \text{while} \mid \dots \mid$
 $+ \mid - \mid * \mid / \mid \dots \mid$
 $(\mid) \mid \{ \mid \} \mid \dots \mid$
 $=+ \mid ++ \mid == \mid \dots$

$\langle \text{identifikátor} \rangle \rightarrow$ $\langle \text{identifikátor} \rangle a \mid \langle \text{identifikátor} \rangle b \mid \dots$
 $\langle \text{identifikátor} \rangle 0 \mid \dots \langle \text{identifikátor} \rangle 9 \mid$
 $a \mid b \mid \dots$

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

$\langle \text{složený příkaz} \rangle \rightarrow \{ \langle \text{seznam příkazů} \rangle \} \quad (1)$

$\langle \text{seznam příkazů} \rangle \rightarrow \langle \text{příkaz} \rangle ; \langle \text{seznam příkazů} \rangle \mid$ (2)

$\langle \text{příkaz} \rangle ; \quad (3)$

$\langle \text{příkaz} \rangle \rightarrow \langle \text{proměnná} \rangle = \langle \text{výraz} \rangle \quad (4)$

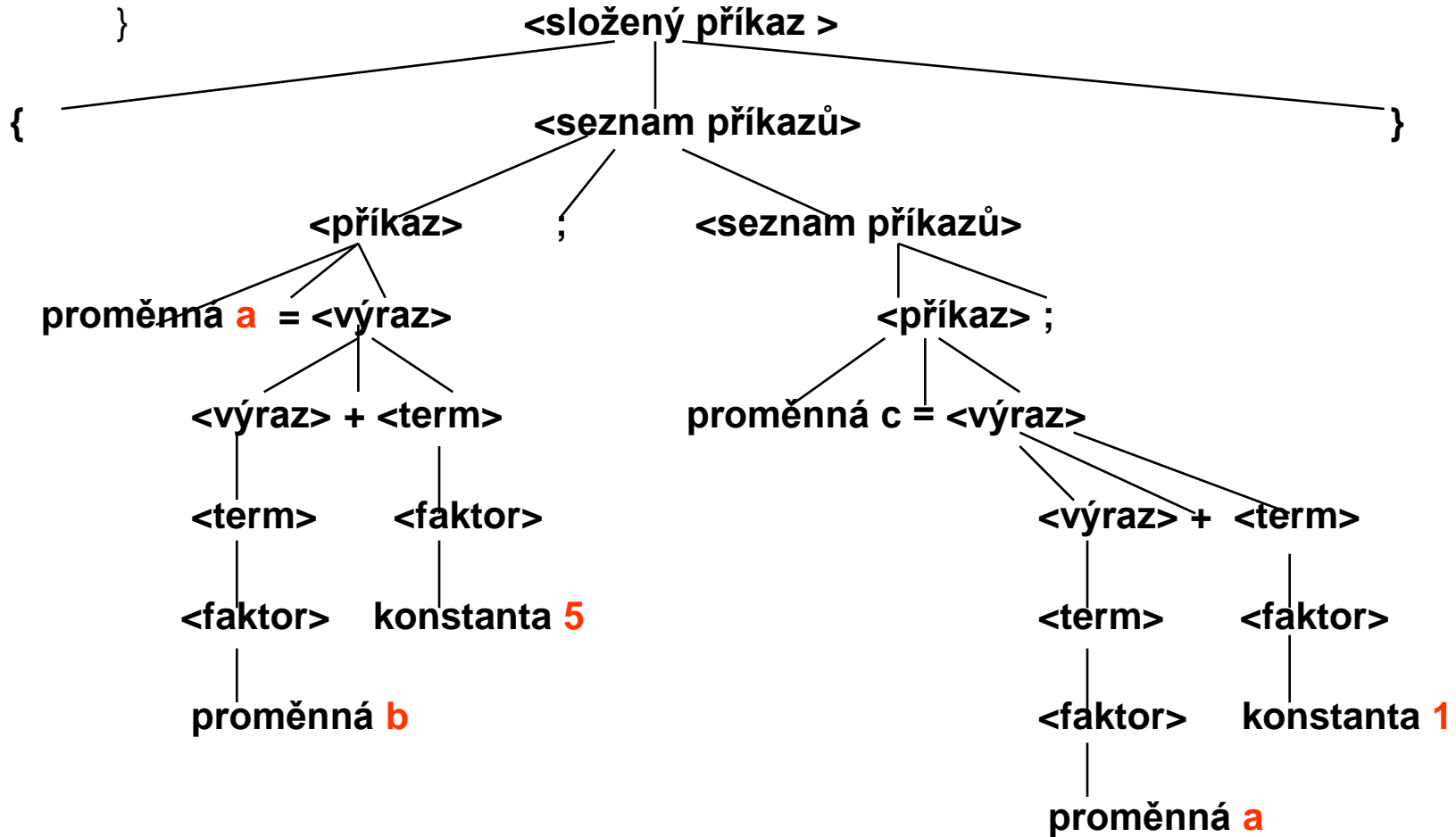
$\langle \text{výraz} \rangle \rightarrow \langle \text{výraz} \rangle + \langle \text{term} \rangle \mid \langle \text{výraz} \rangle - \langle \text{term} \rangle \mid \langle \text{term} \rangle \quad (5)(6)(7)$

$\langle \text{term} \rangle \rightarrow \langle \text{term} \rangle * \langle \text{faktor} \rangle \mid \langle \text{term} \rangle / \langle \text{faktor} \rangle \mid \langle \text{faktor} \rangle \quad (8)(9)(10)$

$\langle \text{faktor} \rangle \rightarrow (\langle \text{výraz} \rangle) \mid \text{proměnná} \mid \text{konstanta} \quad (11)(12)(13)$

Syntaktický analyzátor

Např. { a = b + 5 ; /* poznámka */
c = a + 1 ;
}



Syntaktický analyzátor

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

Struktura je zachycena SA jako posloupnost použití gramatických pravidel při odvození tvaru programu z počátečního symbolu gramatiky

Možnosti:

- Levá derivace 1,2,4,5,7,10,12,10,13, . . .
 (rozepisuje vždy nejlevější neterminální symbol)
- Pravá derivace 1,2,3,4,5,10, . . .
 (rozepisuje vždy nejpravější neterminální symbol)

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:

- a. **Postfixová notace (operátory následují za operandy)**
- b. **Prefixová notace**
- c. **Víceadresové instrukce**

Např. $a = (b + c) * (a + c) ;$

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

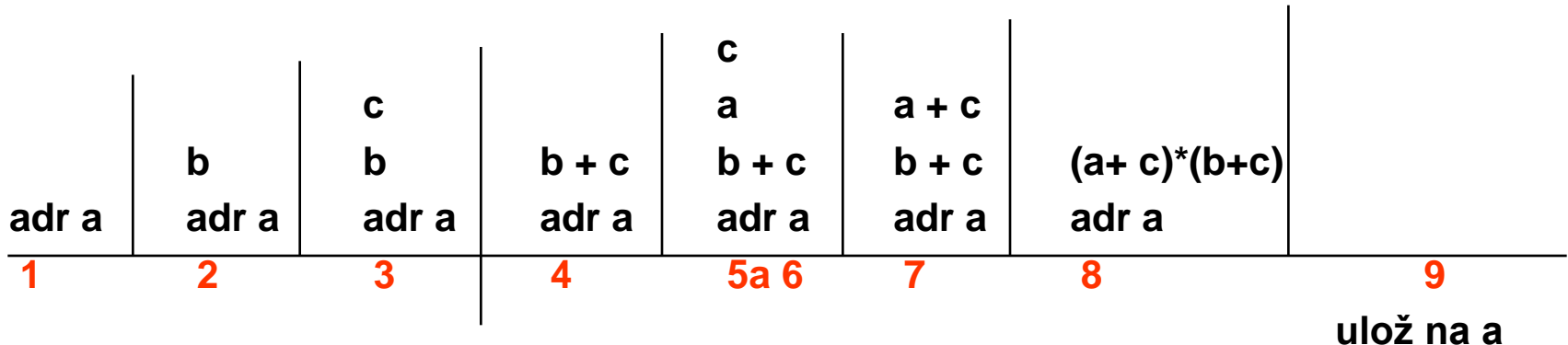
```
PLUS b    c    pom1
PLUS a    c    pom2
MUL pom1 pom2 pom3
ST  a    pom3
```

Redukce počtu pomocných proměnných, eliminace nadbytečných přesunů mezi registry, optimalizace cyklů

```
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)

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

vnitřní jazyk víceadresových instrukcí

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

přeložený strojový kód