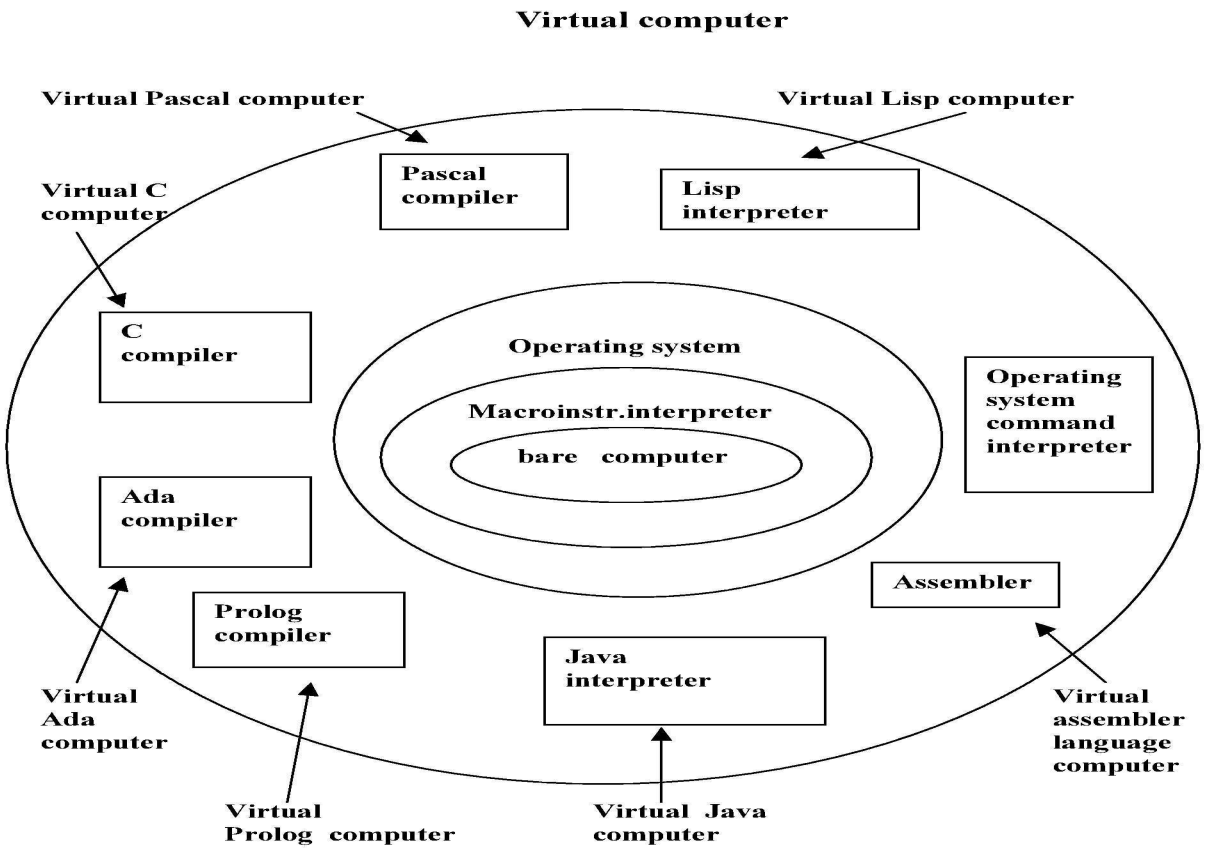
Virtuální počítač

Hladiny

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


Virtual computer
Virtual Pascal computer
Virtual Lisp computer
Virtual C computer
Pascal compiler
Lisp interpreter
C compiler
Operating system
Macroinstr.interpreter
bare computer
Operating system command interpreter
Ada compiler
Assembler
Prolog compiler
Java interpreter
Virtual Ada computer
Virtual assembler language computer
Virtual Prolog computer
Virtual Java computer

Překladač

Z formálního hlediska je překladač zobrazení

Kompilátor : Zdrojový jazyk → Cílový jazyk

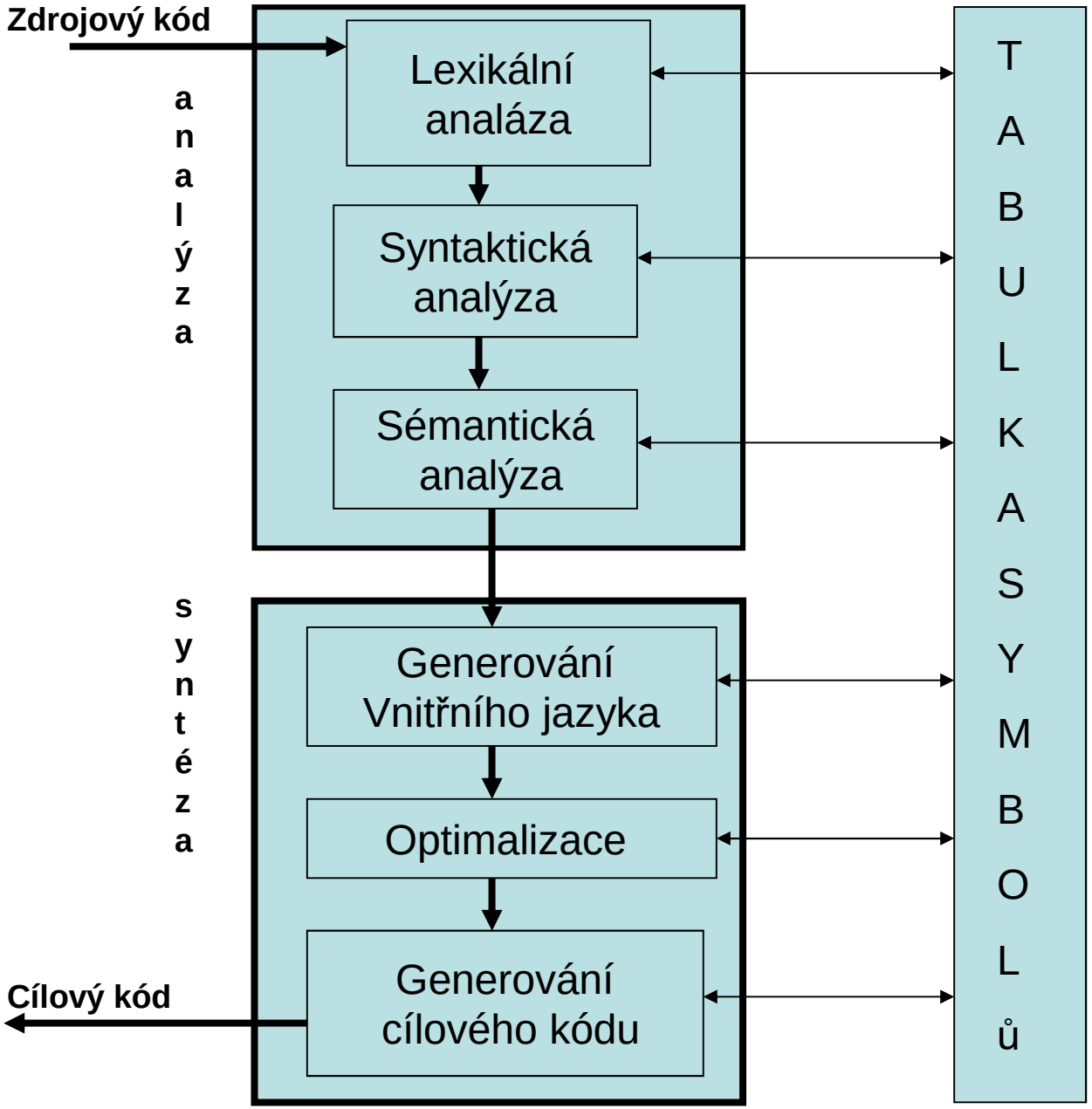
Jeho hlavní části tvoří

Analytická část:

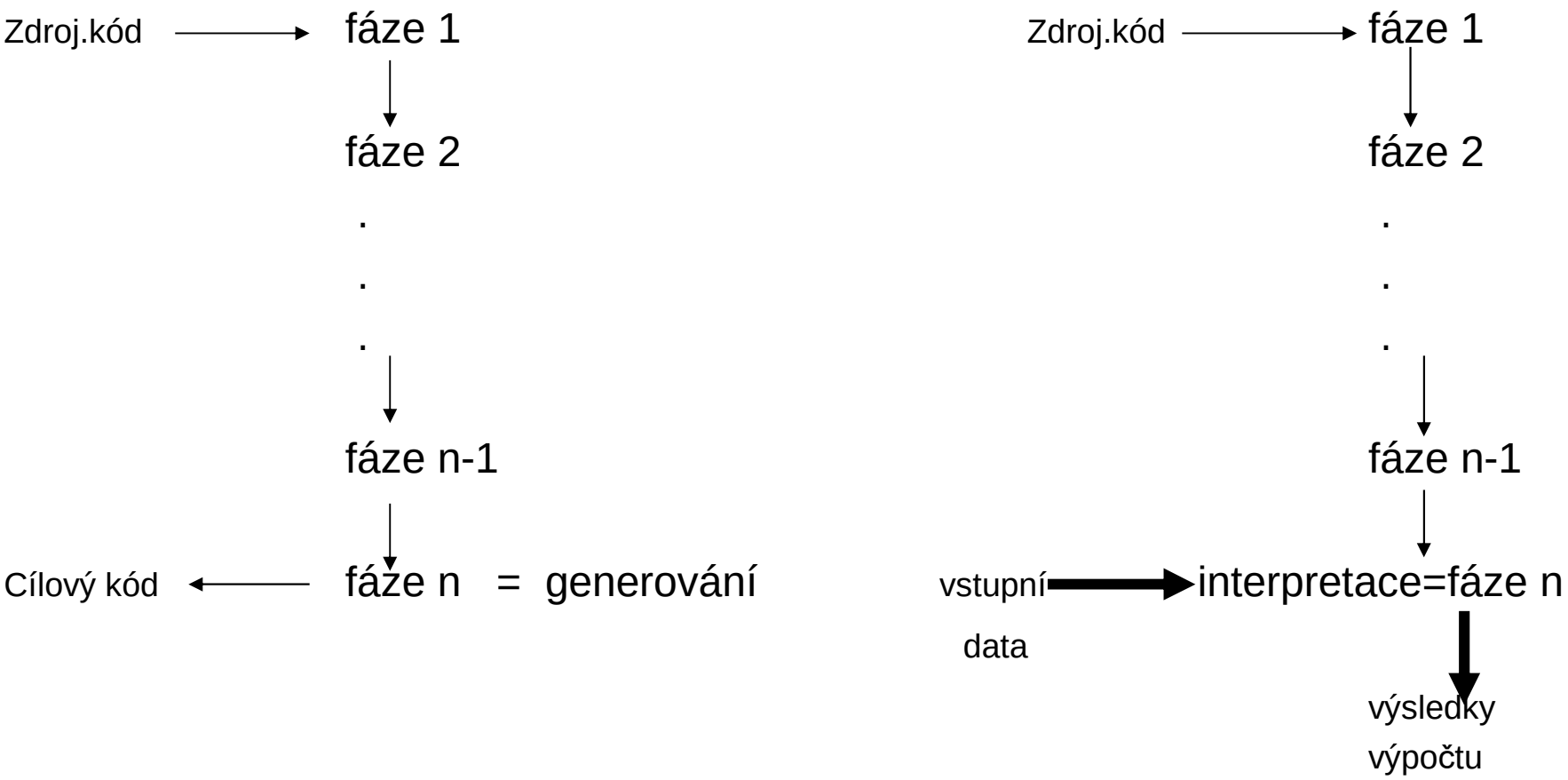
- Lexikální analýza –rozpoznává symboly jazyka
- Syntaktická a sémantická analýza –rozpoznává a kontroluje strukturu a kontextové závislosti programu

Syntetická část:

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


Zdrojový kód
analýza
Lexikální analáza
Syntaktická analýza
Sémantická analýza
syntéza
Generování Vnitřního jazyka
Optimalizace
Generování cílového kódu
Cílový kód
TABULKA SYMBOLů

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 ; /* poznamka */
c = a + 1 ;
}

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

převede 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:

<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)

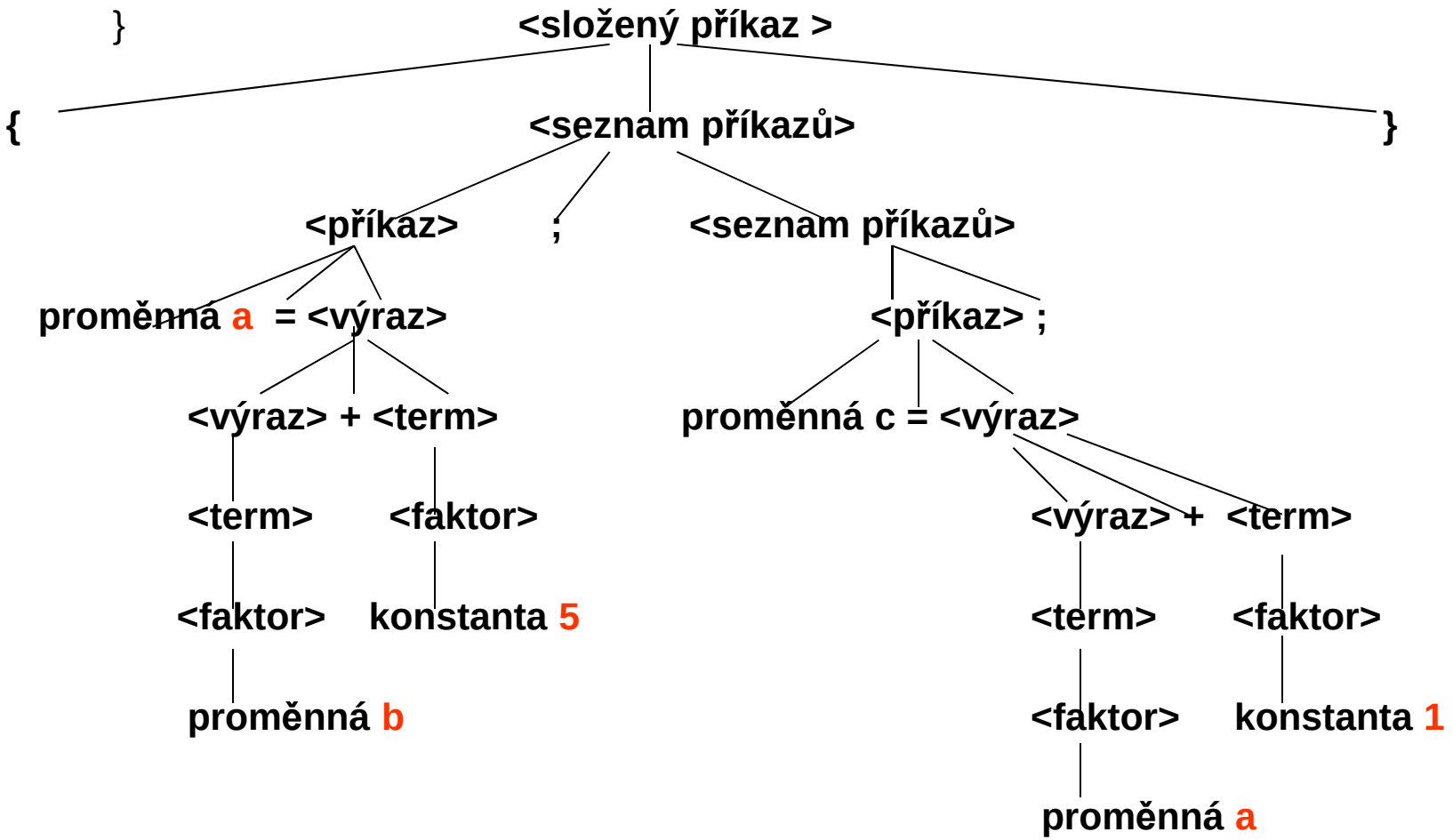
<výraz> → <výraz> + <term> | <výraz> - <term> | <term> (5)(6)(7)

<term> → <term> * <faktor> | <term> / <faktor> | <faktor> (8)(9)(10)

<faktor> → (<výraz>) | proměnná | konstanta (11)(12)(13)

Syntaktický analyzátor

Např. { a = b + 5 ; /* poznamka */
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 (operá

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

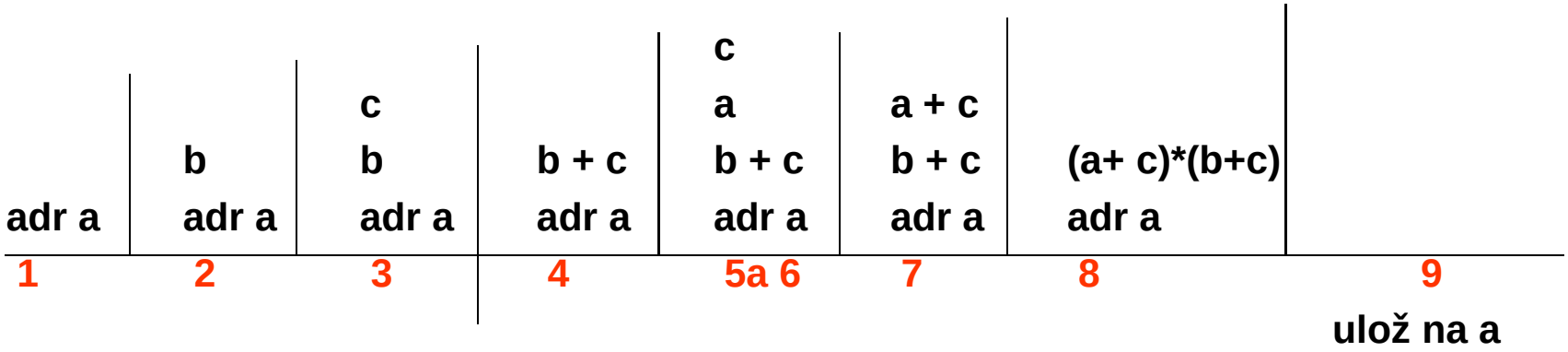
```
PLUS  b     c     pom1
PLUS  a     c     pom2
MUL   pom1  pom2  pom3
ST    a     pom3
```

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