# Instrukční sada

Bajtkódové instrukce Tinstruction pro náš interpret jsou 2operandové, podobně jako v instrukční sadě x86. První adresa značí obvykle první operand a určuje také cíl výsledku a druhá adresa druhý operand. Tinstruction má následující strukturu:

**typedef** union **{** size\_t index\_size**; // index proměnné**  
 void **\*** ptr**;**  
 int integer**;**   
 double real**;**   
**}** Toperand**;**

**typedef** struct **{**

char code**;**

Toperand1 op1**;**

Toperand op2**;**

**}** Tinstruction**;**

První operand na rozdíl od druhého nemůže obsahovat přímou hodnotu typu double. V reálné instrukci stačí, když operace pracuje pouze s jedním přímým operandem, jelikož operace mezi dvěma konstantami je též konstanta. Je to tedy operace, která lze provést během elaborace výrazu a je zbytečné ji provádět až při interpretaci. Na 32bitových systémech nám to umožňuje mít menší[[1]](#footnote-1) velikost instrukce a tím i menší velikost instrukční pásky. Naprostá většina instrukcí pak využívá oba operandy, zbývající alespoň jeden[[2]](#footnote-2) – můžeme tedy hovořit, že bajtkód je efektivně uložen v paměti*[[3]](#footnote-3)*.

Jelikož může být přímý operand typu double pouze jako druhý operand instrukce, pro jednotnost napříč instrukcemi platí, že jakákoliv přímá hodnota je vždy v druhém operandu bez ohledu na datový typ. U nekomutativních aritmetických operací (odčítání a dělení) proto existuje ještě reverzní varianta instrukce, kde se operandy interpretují prohozeně. Zatímco výraz a/1.5 by byl převeden na normální instrukci, výraz 1.5/a na její reverzní variantu. U nekomutativního relačního operátoru je jeho reverzní varianta jiný relační operátor (b > a je totéž jako a < b ), není tedy potřeba dalších reverzních instrukcí pro tyto operace.

Všechny instrukce s sebou v operačním znaku (kódu) nesou informace o datových typech operandů, čímž prakticky interpret odstiňují od rozhodování, nad jakými datovými typy pracuje. Také nese informaci o tom, zda je daný operand přímým operandem, nebo proměnnou. Důsledkem tohoto řešení je množství variant pro každou typovou instrukci. Zatímco instrukce BC\_CJMP pro podmíněný skok má 2 varianty (int a double; je tedy zakódována na 2 bitech); instrukce BC\_WRITE pro výpis na stdout má 8 variant (3 datové typy + varianty s konstantou) má 6 variant a je zakódována na 3 bitech; tak instrukce BC\_LOAD pro přesuny hodnot mezi proměnnými má 14 variant a je zakódována na 4 bitech (využívá tedy 14 z 16 možností) a je naší instrukcí s největším počtem variant.

Počet variant jsme nepatrně snížili tím, že instrukce s přímým operandem nesou informaci pouze o jednom datovém typu (výsledném) a konstanta je do správného typu případně zkonvertována během elaborace výrazu.[[4]](#footnote-4) Celkově naše instrukční sada obsahuje 24 typů instrukcí a celkově 136 instrukcí se započtením všech variant, které jsou zakódovány na 1 B (char).

Seznam všech typů instrukcí a jejich zakódování jsou uvedeny v příloze instrukční sada.

1. Velikost instrukce je 16 B na Windows x86 vs. 24 B na Linux x64. [↑](#footnote-ref-1)
2. Kromě instrukce EOP, ta je však v každém programu jenom jedna. [↑](#footnote-ref-2)
3. Délka instrukční pásky je pro referenční programy ze zadání: 10.1 faktoriál – 27, 10.2 rekurzivní faktoriál – 36, 10.3 vestavěné funkce – 39. [↑](#footnote-ref-3)
4. Například konstanta 1 ve výrazu double a = 1; je zkonvertována na double již během elaborace. Z tohoto důvodu má např. instrukce LOAD jenom 14 variant, nikoliv 16. [↑](#footnote-ref-4)