Vnitřní jazyky překladačů – druhy, použití v jednotlivých fázích překladu, překlad jednoduchých jazykových konstrukcí

Thursday, May 30, 2013 8:42 AM

Po ukončení syntaktické a sémantické analýzy generují některé překladače explicitní intermediální reprezentaci zdrojového programu (mezikód). Intermediální reprezentaci můžeme považovat za program pro nějaký abstraktní počítač. Tato reprezentace by měla mít dvě důležité vlastnosti: měla by být jednoduchá pro vytváření a jednoduchá pro překlad do tvaru cílového programu. Intermediální kód slouží obvykle jako podklad pro optimalizaci a generování cílového kódu. Může však být také konečným produktem překladu v interpretačním překladači, který vygenerovaný mezikód přímo provádí. Intermediální reprezentace mohou mít různé formy.

Postfixová notace

operátory následují ihned za operandy

- A B C * D + - => A – (B * C + D)
- efektivní zpracování pomocí zásobníku, musíme vědět prioritu operátorů

Instrukce postfixoveho zapisu napr. Lit 0, A = uloz konstantu A do zasobniku, opr 0, A = proved instrukci A…

Prefixová notace

operátory a pak operandy

- + A B + C D => (A + B) * (C + D)

Tříadresový kód

Abstraktní forma mezikódu sestávající ze sekvence příkazů ve tvaru **x := y op z**, kde x, y a z jsou jména, konstanty nebo dočasné proměnné, op je nějaký operátor. Na levé straně je **adresa**, na pravé **instrukce.** Adresou může být název (ze zdrojového programu, je pak nahrazen pointerem do jeho tabulky symbolů), konstanta, kompilátorem generovaná dočasná proměnná (užitečné pro optimalizaci).

Jde o linearizovanou podobu syntaktického stromu.

Překlad výrazu x+y*z na tříadresový kód:

t1 := y * z

t2 := x + t1

Další formy tříadresových instrukcí: s unárním operátorem (x = -y), *copy* instrukce (x = y), indexované copy instrukce (x = y[0]), nepodmíněný skok (goto L), podmíněný skok (if x goto L), volání procedur (call p, n; předtím uvedeno *n* parameterů).

Trojice a čtveřice

Implementací tříadresového kódu jsou záznamy se třemi nebo čtyřmi poli: trojice resp. čtveřice. Následující příklady budou ukázány na výrazu: a := b * (-c) + d [b]

Čtveřice

Záznam má čtyři položky nazývané **op, arg1, arg2 a res**. Tříadresový příkaz ve tvaru x := y op z je reprezentován umístěním op do op, y do arg1, z do arg2 a x do res. Některé tříadresové příkazy nepotřebují všechny položky (např. x := y).

Výhoda čtveřic oproti trojicím – v optimalizaci kompilátoru, kdy jsou instrukce často přemísťovány; přesun čtveřic je ok (nová pozice se dá hned určit podle dočasných proměnných), u trojic je při posunu třeba změnit reference na výsledky, protože jsou určeny svou pozicí.

Trojice

Jestliže se chceme vyhnout generování dočasných proměnných, je možné použít formu trojic. Trojice obsahuje op, arg1 a arg2. Místo dočasných proměnných jsou indexy do pole trojic (jejich pozice).

Příklady

Ukažme si tříadresový kód, čtveřice a trojice na příkladě výrazu:

a := b * (- c) + d [b]

Tříadresový kód

```
t1 := - c
t2 := b * t1
t3 := d [ b ]
t4 := t2 + t3
a := t4
```

Čtveřice

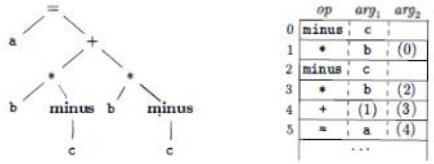
	op	arg1	arg2	res
(1)	uminus	c		t1
(2)	*	b	t1	t2
(3)	loadidx	d	b	t3
(4)	+	t2	t3	t4
(5)	:=	t4		a

Trojice

	op	arg1	arg2
(1)	uminus	c	
(2)	*	b	(1)
(3)	loadidx	d	b
(4)	+	(2)	(3)
(5)	:=	a	(4)

Překlad

u čtveřic voláme metodu pro generování instrukce a předáme jí jen parametry instrukce - např. GADD(...) = generate ADD a parametry jsou levá a pravá strana + docasna promenna;
u trojic tam musí bejt rozhodovací tabulka a kód se generuje v závislosti na tom, co je aktuálně na stacku přímo se to jmenuje Rozhodovací tabulka COMP



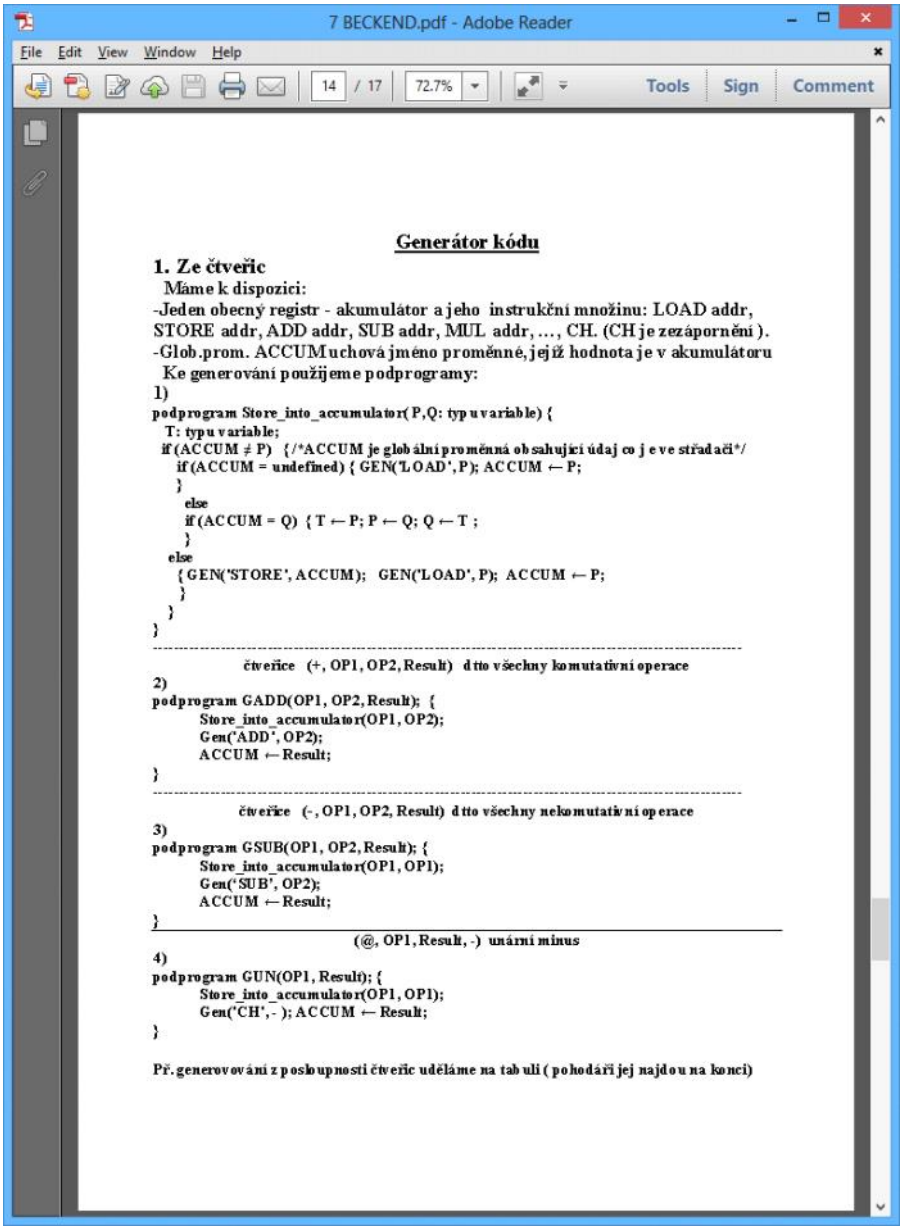
(a) Syntax tree

	op	arg_1	arg_2
0	minus	c	
1	*	b	(0)
2	minus	c	
3	*	b	(2)
4	+	(1)	(3)
5	=	a	(4)
	...		

(b) Triples

Figure 6.11: Representations of $a + a * (b - c) + (b - c) * d$

From <https://d.docs.live.net/e3534876709763a3/Dokumenty/ZCU/Statnice/Statnice.docx>



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 je 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)

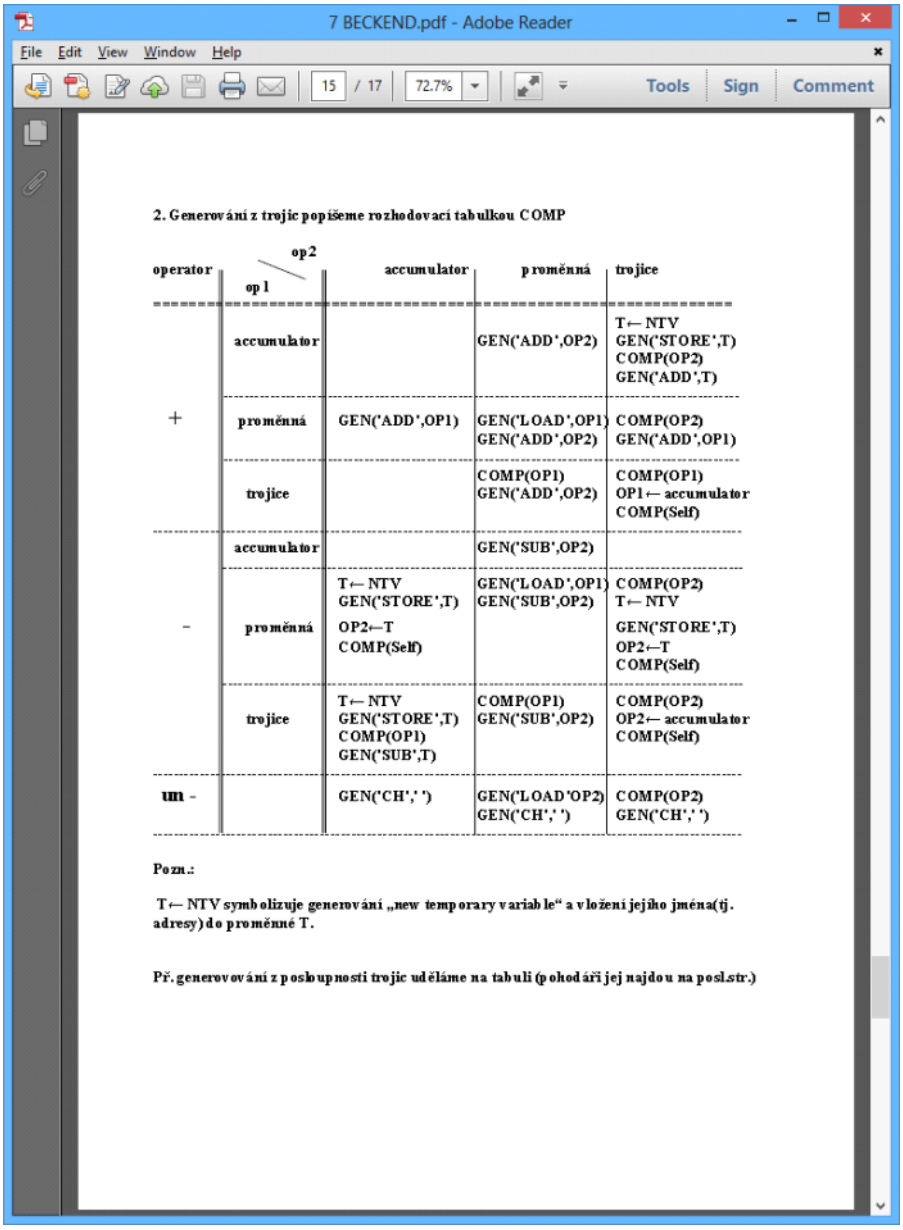
```
podprogram GSUB(OP1, OP2, Result); {
        Store_into_accumulator(OP1, OP1);
        Gen('SUB', OP2);
        ACCUM ← Result;
}
```

(@, 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 (pohodářiji najdou na konci)



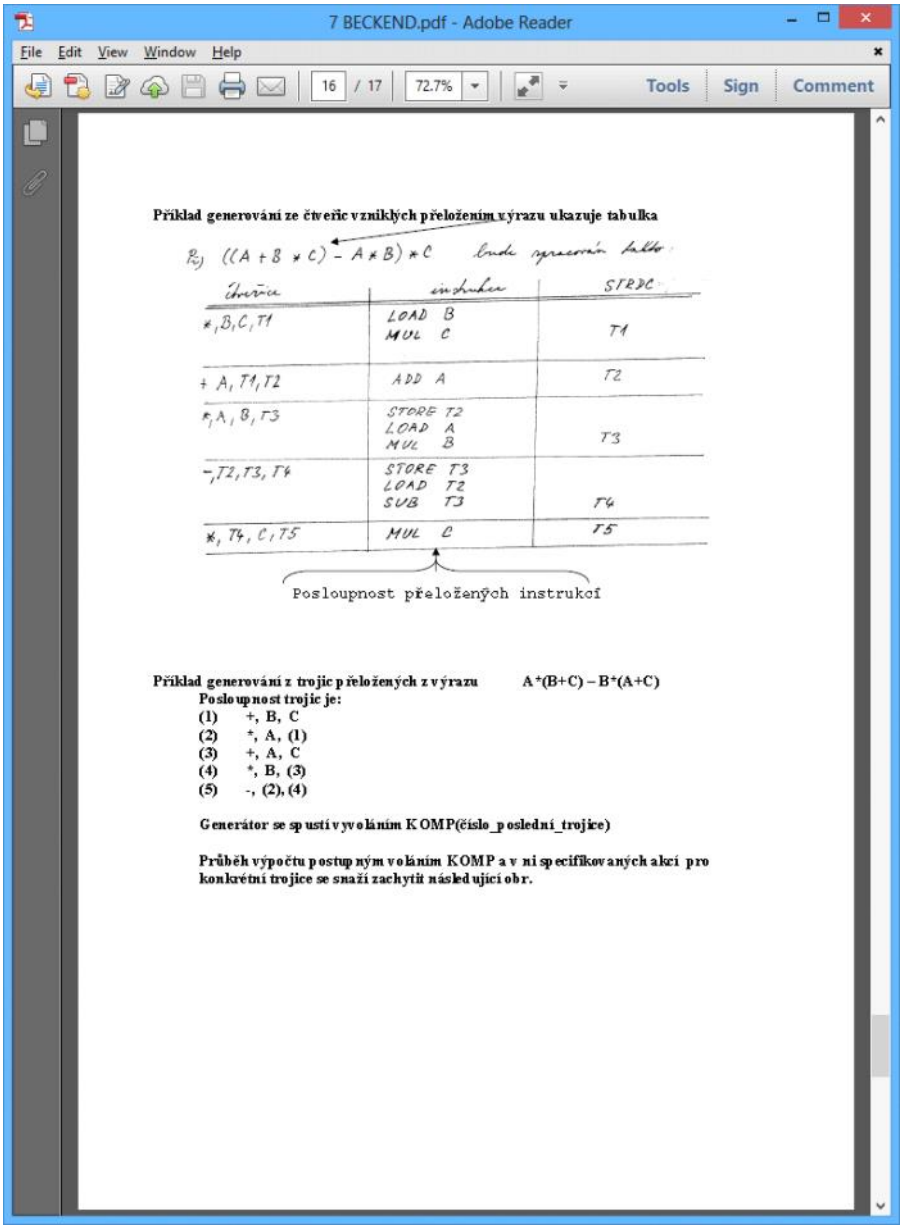
2. Generování z trojic popíšeme rozhodovací tabulkou COMP

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 (pohodáří jej najdou na posl.str.)



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 ní specifikovaných akcí pro konkrétní trojice se snaží zachytit následující obr.