Architektury systemów komputerowych

Wykład 1: Struktura programu

Krystian Bacławski

Instytut Informatyki
Uniwersytet Wrocławski

2 marca 2022

Kod trójkowy

Zastosowanie

Kod trójkowy (ang. *three-address code*) to postać pośrednia stosowana przez kompilatory przy translacji z języka wysokiego poziomu do asemblera. W większości przypadków można ją bezpośrednio przetłumaczyć na kod maszynowy procesora.

Kod wyrażony w kodzie trójkowym składa się z **adresów** i **instrukcji**.

Czego nie ma w TAC?

- wysokopoziomowych instrukcji sterujących (`for`, `while`, `switch`, ...)
- typów złożonych (`struct`, `union`, `enum`, ...)
- procedur
- zasięgów leksykalnych

Adresy

- stała,
- nazwa – zmiennej, funkcji, etykiety,
- zmienna tymczasowa.

Instrukcje

- `x := y` *binop* `z` – gdzie *binop* jest operatorem binarnym
- `x :=` *unop* `z` – gdzie *unop* jest operatorem unarnym
- `x := y` – kopiowanie danej
- `goto L` – skok bezwarunkowy do etykiety `L`
- `if b goto L` – skok do etykiety `L`, jeśli `b` jest prawdą
- `if x` *relop* `y goto L` – skok do `L`, jeśli `x` jest w relacji *relop* do `y`
- `x := &y` – wyznaczenie wkaźnika do zmiennej (referencja)
- `x := *y`, `*x := y` – dereferencja wskaźnika
- `param x` – użyj `x` jako parametru procedury
- `call p, n` – wołanie procedury `p` z `n` argumentami
- `return n` – zwróć `n` z procedury

binop $\in$ { `+`, `-`, `*`, `/`, ..., `&&`, `||`, ..., `&`, `|`, `^`, ... }
unop $\in$ { `-`, `!`, `~` ...}
relop $\in$ { `==`, `!=`, `<=`, `<`, ...}

Więcej na ten temat w §6.2 Compilers: Principles, Techniques & Tools; Aho, Lam, Sethi, Ullman.

Przykład 1

Oblicz jedno rozwiązanie równania kwadratowego

```
x = (-b + sqrt(b*b - 4*a*c)) / (2*a)
```

```
t1 := b * b
t2 := 4 * a
t3 := t2 * c
t4 := t1 - t3
param t4
t5 := call sqrt,1
t6 := - b
t7 := t5 + t6
t8 := 2 * a
x  := t7 / t8
```

Przykład 2

Wskaźniki w TAC

Aby uprościć zapis możemy wprowadzić następujące dwie instrukcje:

- `x := a[i]` jest tym samym co `t := a + i; x := *t`
- `a[i] := x` jest tym samym co `t := a + i; *t := x`

Arytmetyka na wskaźnikach w TAC jest beztypowa!

`a[i]` nie oznacza dostępu do `i`-tego elementu tablicy `a`, tylko do adresu `a + i`

Zachowujemy typ wskaźnika, by odwołać się do słowa określonego rozmiaru!

Znajdź element niemniejszy niż `v`

```
uint32_t *a;
...
int i = 0;
while (a[i] < v) {
  i++;
}
```

```
   i  := 0
L: t1 := i * 4
   t2 := a[t1]
   if t2 >= v goto E
   i  := i + 1
   goto L
E:
```

Jak kompilator języka C reprezentuje program w pamięci?

Graf przepływu sterowania

Graf skierowany reprezentujący wszystkie ścieżki programu, które można przejść w trakcie jego wykonania. Wierzchołkami są **bloki podstawowe**.

Blok podstawowy

Sekwencja instrukcji za wyjątkiem skoków, kończąca się instrukcją skoku warunkowego albo bezwarunkowego albo powrotu z procedury. Instrukcje w bloku podstawowym zawsze zaczynamy wykonywać od pierwszej. Innymi słowy, blok podstawowy ma jeden punkt wejścia i jeden punkt wyjścia.

Graf przepływu sterowania

```
FOR I := 1 TO n - 1 DO
  FOR J := 1 TO I DO
    IF A[J] > A[J+1] THEN
    BEGIN
      Temp := A[J]
      A[J] := A[J + 1]
      A[J + 1] := Temp
    END
  DONE
DONE
```

```
        I := 1                      ; <<B1>>
        goto ITest
ILoop:  J := 1                      ; <<B2>>
        goto JTest
JLoop:  t1 := 4 * J                 ; <<B3>>
        t2 := A[t1]                 ; A[J]
        t3 := J + 1
        t4 := 4 * t3
        t5 := A[t4]                 ; A[J + 1]
        if t2 <= t5 goto JPlus
        t6 := 4 * J                 ; <<B4>>
        Temp := A[t6]               ; Temp := A[J]
        t7 := J + 1
        t8 := 4 * t7
        t9 := A[t8]                 ; A[J + 1]
        t10 := 4 * J
        A[t10] := t9                ; A[J] := A[J + 1]
        t11 := J + 1
        t12 := 4 * t11
        A[t12] := Temp              ; A [J + 1] := Temp
JPlus:  J := J + 1                  ; <<B5>>
JTest:  if J <= I goto JLoop        ; <<B6>>
IPlus:  I := I + 1                  ; <<B7>>
ITest:  t13 := n - 1
        if I <= t13 goto ILoop      ; <<B8>>
```

Źródło: Optimization: Introduction and Control Flow Analysis