Konstrukcija kompilatora

— Troadresni kod —

Milena Vujošević Janičić

Matematički fakultet, Univerzitet u Beogradu

Sadržaj

Na slajdovima obavezno pogledati animacije koje su ovde izostavljene!

1 Troadresni kod

Troadresni kod

- Troadresni kod - međureprezentacija visokog nivoa u kojoj svaki operacija ima najviše tri operanda
- U trenutku generisanja troadresnog koda nije bitno misliti na njegovu optimizaciju
- U redu je da taj kod sadrži dodele viška i redudantna izračunavanja - to će biti eliminisano u fazi optimizacije
- Razmatraćemo stek izvršavanja i virtuelne tabele

Sample TAC Code

```
int a;
int b;
int c;
int d;

a = b + c + d;
b = a * a + b * b;
```

```
_t0 = b + c;
a = _t0 + d;
_t1 = a * a;
_t2 = b * b;
b = _t1 + _t2;
```

Privremene promenljive

- Tri u troadresnom kodu označava broj operanada proizvoljen instrukcije
- Evaluacija izraza sa više od tri podizraza zahteva uvođenje privremenih promenljivih
- Videćemo postupak uskoro

1.1 Aritmetičke i bulovske operacije

Sample TAC Code

```
int a;
int b;

a = 5 + 2 * b;
```

TAC allows for instructions with two operands.

```
_t0 = 5;
_t1 = 2 * b;
a = _t0 + _t1;
```

Troadresni kod

- Razmatraćemo pojednostavljeni troadresni kod u kojem su dozvoljene naredne dodele

```
var   = constant;
var_1 = var_2 ;
var_1 = var_2 op var_3 ;
var_1 = constant op var_2 ;
var_1 = var_2 op constant;
var   = constant_1 op constant_2 ;
```

- Dozvoljeni operatori $+, -, *, /, \%$
- Kako biste preveli `y = -x;` ?

```
y = 0 - x;
y = -1 * x;
```

One More with `bool`s

```
int x;
int y;
bool b1;
bool b2;
bool b3;

b1 = x + x < y
b2 = x + x == y
b3 = x + x > y
```

```
_t0 = x + x;
_t1 = y;
b1 = _t0 < _t1;

_t2 = x + x;
_t3 = y;
b2 = _t2 == _t3;

_t4 = x + x;
_t5 = y;
b3 = _t5 < _t4;
```

TAC sa bool vrednostima

- Bulovske promenljive se predstavljaju kao celobrojne vrednosti koje mogu imati vrednost nula ili vrednost različitu od nule
- Pored aritmetičkih operatora, podržavamo i $<$, `==`, `||` i `&&`
- Kako bi moglo da se prevede `b = (x <= y)` ?

```
_t0 = x < y;
_t1 = x == y;
b = _t0 || _t1;
```

1.2 Kontrola toka

Labele

- Imenovane labele označavaju određene delove koda na koje se može skočiti
- Mogu postojati razne instrukcije za kontrolu toka
- `Goto label;` — bezuslovni skok
- `ifZ value goto label;` — uslovni skok, `ifZ` je uvek upareno sa `Goto label`

Control Flow Statements

```
int x;
int y;
int z;

if (x < y)
   z = x;
else
   z = y;

z = z * z;
```

```
    _t0 = x < y;
    IfZ _t0 Goto _L0;
    z = x;
    Goto _L1;
_L0:
    z = y;
_L1:
    z = z * z;
```

Control Flow Statements

```
int x;
int y;

while (x < y) {
   x = x * 2;
}

y = x;
```

```
_L0:
    _t0 = x < y;
    IfZ _t0 Goto _L1;
    x = x * 2;
    Goto _L0;
_L1:
    y = x;
```

1.3 Funkcije i stek okviri

Kompilacija funkcija

- Funkcije se sastoje iz četiri dela:
  - Labela koja označava početak funkcije

– Instrukcija `BeginFunc N;` koja rezerviše N bajtova za prostor za lokalne i privremene promenljive
– Telo funkcije
– Instrukcija `EndFunc` koja obeležava kraj funkcije, kada se do nje dođe ona je zadužena da počisti stek okvir i da vrati kontrolu na odgovarajuće mesto

A Complete Decaf Program

```
void main() {
   int x, y;
   int m2 = x * x + y * y;

   while (m2 > 5) {
      m2 = m2 - x;
   }
}
```

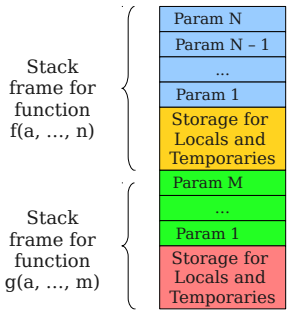
```
main:
   BeginFunc 24;
   _t0 = x * x;
   _t1 = y * y;
   m2 = _t0 + _t1;
_L0:
   _t2 = 5 < m2;
   IfZ _t2 Goto _L1;
   m2 = m2 - x;
   Goto _L0;
_L1:
   EndFunc;
```

Upravljanje stekom u TACu

Podsetnik:

- Funkcija pozivaoc je odgovorna za smeštanje argumenata funkcije na stek
- Pozvana funkcija je odgovorna za smeštanje svojih lokalnih i privremenih promenljivih

A Logical Decaf Stack Frame



Upravljanje stekom u TACu

- Instrukcija `BeginFunc N;` rezerviše prostor za lokalne i privremene promenljive
- Instrukcija `EndFunc;` vraća prostor koji je rezervisan sa `BeginFunc N;`
- Jedna parametar je gurnut na stek od strane pozivaoca korišćenjem instrukcije `PushParam var`
- Prostor je oslobođen od strane pozivaoca korišćenjem instrukcije `PopParams N;`
- N se meri u bajtovima, ne po broju argumenata!

Compiling Function Calls

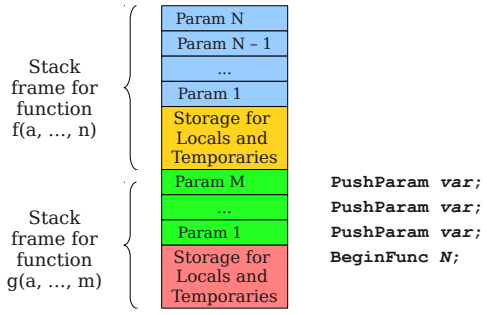
```
void SimpleFn(int z) {
    int x, y;
    x = x * y * z;
}

void main() {
   SimpleFunction(137);
}
```

```
_SimpleFn:
   BeginFunc 16;
   _t0 = x * y;
   _t1 = _t0 * z;
   x = _t1;
   EndFunc;

main:
   BeginFunc 4;
   _t0 = 137;
   PushParam _t0;
   LCall _SimpleFn;
   PopParams 4;
   EndFunc;
```

A Logical Decaf Stack Frame

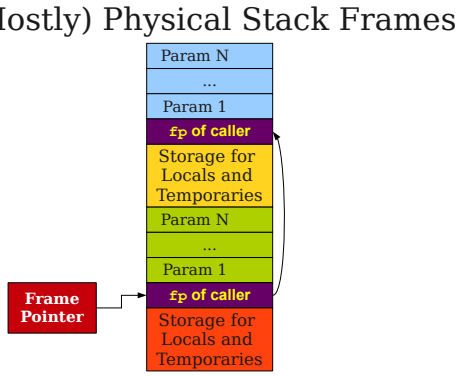


1.4 Dodatne informacije u stek okviru

Frame pointer

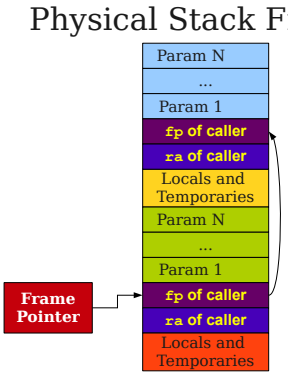
- Prethodni prikazi su bili prikazi logičkih stek okvira

- Da bi se implementirao stek, potrebno je čuvati dodatne informacije
- Pre svega, potrebno je znati gde se nalaze lokalne promenljive, parametri i privremene promenljive
- One se čuvaju u odnosu na *frame pointer* fp



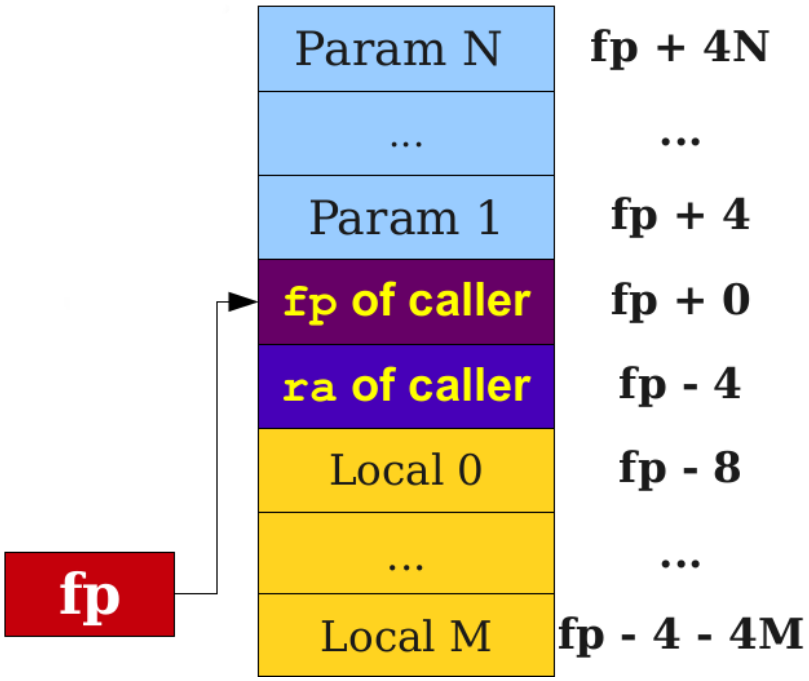
Dodatne informacije u stek okviru

- Interno, procesor ima specijalni registar koji se naziva brojač instrukcija *program counter* (PC) koji čuva adresu naredne instrukcije koja treba da se izvrši
- Kad kôd funkcije završi sa radom, potrebno je da se PC podesi tako da se nastavi izvršavanje funkcije tamo gde je ono bilo prekinuto
- Adresa gde je potrebno funkcija da se vrati se drugačije čuva na različitim platformama. Na primer u okviru MIPS platforme ona se čuva u okviru specijalnog registra koji se zove `ra` (*return address*)
- To omogućava MIPS funkcijama da budu pozvane: svaka funkcija treba da sačuva prethodnu vrednost od ra

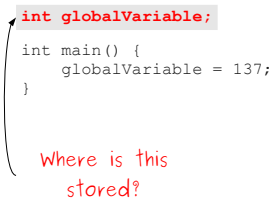


Gde se čuvaju vrednosti parametara

- Parametri počinju od adrese fp+4 i rastu naviše
- Lokalne promenljive i privremene promenljive počinju od adrese fp-8, i rastu naniže



And One More Thing...



Global pointer

- MIPS takođe ima registar koji se naziva *golbal pointer* (gp) koji čuva adresu globalnog prostora
- Memorija na koju pokazuje globalni pointer se tretira kao niz vrednosti koji raste naviše
- Potrebno je za svaku globalnu promenljivu odrediti pomeraj u ovom nizu tj gde se ona tačno u tom nizu čuva

Rezime izgleda memorije

- Većina detalja je apstrahovana IR formatom
- Parametri počinju na adresi fp + 4 i rastu naviše
- Lokanle promenljive počinju na adresi fp - 8 i rastu naniže
- Globalne promenljive počinju od adrese gp + 0 i rastu naviše

2 Troadresni kod za objekte

2.1 Poziv metoda

TAC for Objects, Part I

```
class A {
    void fn(int x) {
        int y;
        y = x;
    }
}

int main() {
    A a;
    a.fn(137);
}
```

```
_A.fn:
    BeginFunc 4;
    y = x;
    EndFunc;

main:
    BeginFunc 8;
    _t0 = 137;
    PushParam _t0;
    PushParam a;
    LCall _A.fn;
    PopParams 8;
    EndFunc;
```

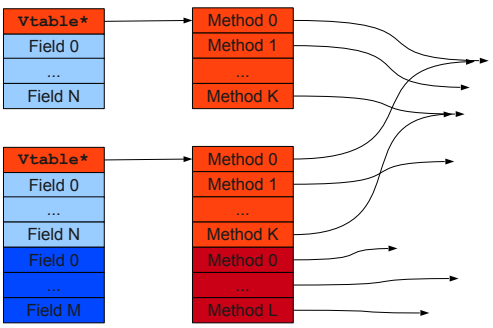
2.2 Pristup podacima

Pristup memoriji

- Potrebno je da proširimo TAC tako da ima mogućnost pristupa memoriji, tj potrebno je da pristup poljima objekta pretvorimo u relativne memorijske adrese

```
var_1 = *var_2
var_1 = *(var_2 + constant)
*var_1 = var_2
*(var_1 + constant) = var_2
```

A Reminder: Object Layout



TAC for Objects, Part II

```
class A {
    int y;
    int z;
    void fn(int x) {
        y = x;
        x = z;
    }
}

int main() {
    A a;
    a.fn(137);
}
```

```
_A.fn:
    BeginFunc 4;
    *(this + 4) = x;
    x = *(this + 8);
    EndFunc;

main:
    BeginFunc 8;
    _t0 = 137;
    PushParam _t0;
    PushParam a;
    LCall _A.fn;
    PopParams 8;
    EndFunc;
```

2.3 Dinamičko razrešavanje poziva

OOP in TAC

- Kako obezbediti dinamičko razrešavanje poziva metoda?
- Adresa virtuelne tabele objekta može da se referencira kroz ime dodeljeno virtuelnoj tabeli, obično je to ime isto kao ime objekta, npr `_t0 = Base;`
- Kada se kreira objekat, mora prvo da se postavi pokazivač na virtuelnu tabelu
- Instrukcija `ACall` može da se koristi za poziv metoda kada je za njega dat pokazivač na prvu instrukciju

TAC for Objects, Part III

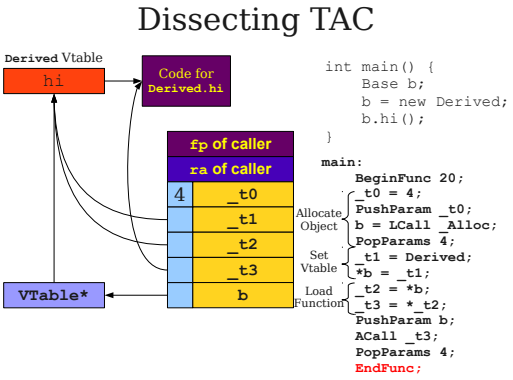
```
class Base {
  void hi() {
    Print("Base");
  }
}

class Derived extends Base{
  void hi() {
    Print("Derived");
  }
}

int main() {
    Base b;
    b = new Derived;
    b.hi();
}
```

```
_Base.hi:
    BeginFunc 4;
    _t0 = "Base";
    PushParam _t0;
    LCall _PrintString;
    PopParams 4;
    EndFunc;
Vtable Base = _Base.hi,
    ;

_Derived.hi:
    BeginFunc 4;
    _t0 = "Derived";
    PushParam _t0;
    LCall _PrintString;
    PopParams 4;
    EndFunc;
Vtable Derived = _Derived.hi,
    ;
```



3 Generisanje troadresnog koda

Generisanje troadresnog koda

- U ovom stadijumu kompilacije, imamo na raspolaganju
  - AST
  - koji je anotiran sa informacijama o dosegu
  - koji je anotiran i sa informacijama o tipovima
- Da bi se generisao TAC, potrebno je još jednom obići rekruzivno AST
  - Generiši TAC za svaki podizraz ili podnaredbu
  - Koristeći generisani TAC za podizraze/naredbe, generiši TAC za kompletne izraze i naredbe

3.1 Generisanje TACa za izraze

Generisanje TACa za izraze

- Definiši funkciju `cgen(expr)` koja generiše TAC koji računa izraz, čuva njenu vrednost u privremenoj promenljivoj i vraća ime te promenljive
- Definiši `cgen(expr)` direktno za atomičke izraze (konstante, this, identifikatore i slično)
- Definiši `cgen(expr)` rekurzivno za složene izraze (binarne operatore, pozive funkcija)

cgen for Basic Expressions

cgen(k) = { // k is a constant
    Choose a new temporary t
    Emit($t = k$);
    Return t
}
cgen(id) = { // id is an identifier
    Choose a new temporary t
    Emit($t = id$)
    Return t
}

cgen for Binary Operators

cgen($e_1 + e_2$) = {
    Choose a new temporary t
    Let t_1 = **cgen**(e_1)
    Let t_2 = **cgen**(e_2)
    Emit($t = t_1 + t_2$)
    Return t
}

An Example

cgen(5 + x) = {
    Choose a new temporary t
    Let t_1 = {
        Choose a new temporary t
        Emit(t = 5)
        return t
    }
    Let t_2 = {
        Choose a new temporary t
        Emit(t = x)
        return t
    }
    Emit ($t = t_1 + t_2$)
    Return t
}

```
_t0 = 5
_t1 = x
_t2 = _t0 + _t1
```

3.2 Generisanje TACa za naredbe

Generisanje naredbi

- Možemo proširiti funkciju `cgen` tako da radi sa naredbama
- Za razliku od `cgen` za izraze, `cgen` za naredbe ne vraća ime privremene promenljive koja čuva vrednost

cgen for Simple Statements

cgen(*expr;*) = {
    cgen(*expr*)
}

cgen for `while` loops

cgen(`while` `(`*expr*`)` *stmt*) = {
    Let L_{before} be a new label.
    Let L_{after} be a new label.
    Emit(L_{before}`:`)
    Let *t* = **cgen**(*expr*)
    Emit(`IfZ` *t* `Goto` L_{after})
    cgen(*stmt*)
    Emit(`Goto` L_{before})
    Emit(L_{after}`:`)
}

4 Literatura

Literatura

- (The Dragon Book) Compilers: Principles, Techniques, and Tools Alfred V. Aho,Monica S. Lam, Ravi Sethi, and Jeffrey D. Ullman
- Kompajleri Stanford https://web.stanford.edu/class/archive/cs/cs143/cs143.1128/