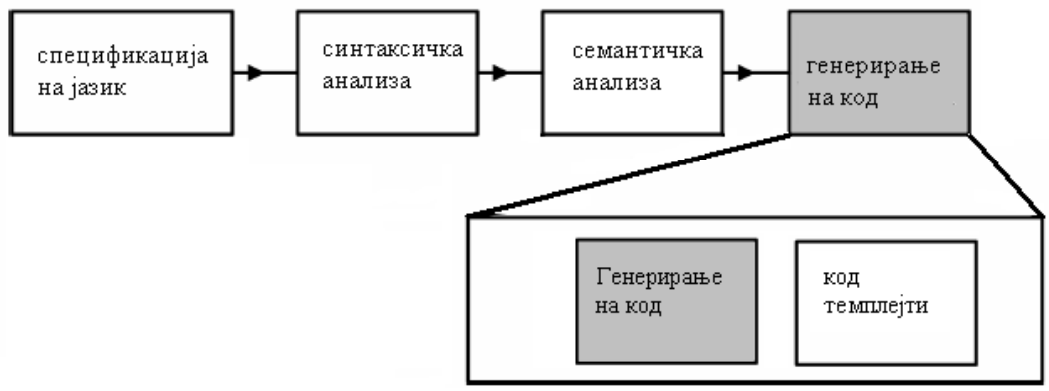
Генерирање на код

Генерирање на код

- После семантичката анализа би требало во кодот веќе да нема гершки, ако има, тогаш скоро секогаш генерирањето на кодот ќе падне.

- Овде се дава осврт на генерирањето на код во компајлерот на Inger.
  - Имлементација (асембли код) за секој оператор кој се јавува во Inger,
  - Зачувувањето на типот на податоци
  - Пресметка на низата од офсетови и повикувањето на функции
- Темплејтите од код во асембли јазик за секој оператор во Inger.

Генерирање на код



Вовед

- Најмистичен аспект во конструирањето на компајлерот
- Не е премногу тешко, но побарува големо посветување на внимание на деталите
- Пристапот во компајлерот на Inger е да се напише темплејт за секоја операција
- Сите темплејти можат да се менуваат во било кој редослед (се смета дека редоследот е точен затоа што претходно се е поминато низ фазите на синтаксичка и семантичка анализа
- Пример int a = (b + 0x20);
  - За генерирање на овој код се повикуваат 4 темплејти
  - Редоследот е определен со редоследот по кој се евалуираат операциите, т.е. Редоследот по кој се поврзани јазлите во апстрактното дрво
    1. Собирање: се пресметува b + 0x20
    2. Дереференцирање: наоѓање на мемориска локација за изразот во заградите
    3. Декларација: променливата се декларира како тип integer, или во стекот (ако е локална променлива) или во купот (ако е глобална променлива)
    4. Доделување: вредноста која се определува со темплејтот за деференцирање се зачувува во локацијата која се врача од страна на темплејтот за декларација.

Вовед

- Ако темплејтите се напишани доволно внимателно и доволно се тестирани, може да се креира компајлер кој ги поддржува и се грижи за редоследот на темплејтите
- Наредно прашање е како темплејтите да се поврзат еден со друг.
  - Ова се прави со означување на еден регистер (eax), како регистер за резултатот.
  - Секој темплејт го зачувува неговиот резултат во тој регистер eax, без разлика дали е вредност или покажувач.
  - Што означува вредноста зачувана во eax е определено со темплејтот кој ја зачувува таа вредност.

Предложен код

- Компајлерот на Inger генерира асембли код, па неопходно е да се спакува во формат кој го очекува асемблерот
- Користат GNU AT&T асемблер, кој ја користи синтаксата на јазикот AT&T (слична на Intel), но со некои посебни карактеристики
- Пример
  - асембли синтаксата на Intel : MOV EAX, EBX ја копира вредноста која се наоѓа во регистерот EBX во регистерот EAX
  - Во синтаксата на GNU AT&T ова е: movl %ebx, %eax
- Неколку разлики:
  - имињата на регистрите се со мала буква и со префикс процент (%) кој индицира дека се работи за регистри, а не имиња на глобални променливи;
  - Редоследот на операндите е обратен
  - На инструкцијата за паметење mov и се става префикс со големината на операндите (4 bytes, long).

Предложен код

- Секој фајл во GNU Асемблерот има барем еден податочен дел (назначен со .data), и еден дел со код (назначен со .text).
  - Податочниот дел содржи глобални променливи и стринг константи
  - Делот со код го содржи кодот
- Делот за кодот не се пишува се додека податочниот дел се модифицира.
- Глобалните променливи се декларираат со специфицирање на нивната големина, и опционо типот и усогласување и се секогаш од тип @object
- Наспроти нив функциите се од тип @function
- Мора да се декларира барем една функција (main), која се користи како појдовна точка на програмата.
  - Нејзиниот тип е секогаш @function.

Глобални декларации

- Кодот за една програма во Inger се генерира со повеќекратно поминување по дрвото
  - Првиот пат е обавезен за да се најдат сите глобални декларации
  - Како што се минува дрвото, модулот за генерирање на код ги проверува јазлите за декларација
  - Кога ќе најде таков јазол, од симболната табела се превзема симболот кој припаѓа на таа декларација
  - Ако овој симбол е глобална променлива, се зема информацијата за типот и се генерира код.
  - Во текот на ова поминување се изоставаат локалните променливи и параметрите на функциите.

Пресметка на ресурси

- Во текот на второто поминување се генерира вистинскиот код
- При ова поминување се креираат и имплементации за функциите
- Пред да се генерира кодот, модулот за генерирање на код мора да ја знае локацијата на сите параметри на функциите и сите локални променливи на стекот
- Ова се прави со брзо скенирање на телото на функцијата за да се најдат локалните декларации.
- Секогаш кога ќе се најде декларација, се определува нејзината позиција во стекот и тоа се зачувува во самиот симбол, во симболната табела.
- На овој начин референците кон локалните променливи лесно можат да се конвертираат во локации на стекот кога се имплементира генерирање на кодот.

Меѓурезултати на изрази

- Генерирањето на код во Inger е имплементирано на многу едноставен, директен начин.
- Не се инволвирани вистински регистри, и сите меѓу вредности и резултати од изрази се зачувуваат во EAX регистерот
- Ова може да доведе до многу неоптимизиран код, но е лесно за пишување

Пример

```
module simple;

extern printInt : int i → void;

int a, b;

start main : void → void
{
    a = 16;
    b = 32;
    printInt ( a * b );
}
```

```
.data
.globl a
    .align 4
    .type   a,@object
    .size   a,4
a:
    .long   0
.globl b
    .align 4
    .type   b,@object
    .size   b,4
b:
    .long   0
.text
    .align 4
.globl main
    .type   main,@function
main:
    pushl   %ebp
    movl    %esp, %ebp
    subl    $0, %esp
    movl    $16, %eax
    movl    %eax, a
    movl    $32, %eax
    movl    %eax, b
    movl    a, %eax
    movl    %eax, %ebx
    movl    b, %eax
    imul    %ebx
    pushl   %eax
    call    printInt
    addl    $4, %esp
    leave
    ret
```

Меѓурезултати на изрази

- eax регистерот може да содржи или вредности или повикувач, зависно од темплејтот кој ја става вредноста во eax.
  - Пример ако е собирање, eax ќе содржи нумеричка вредност, (или реална или integer). Ако е темплејт за адресирачки оператор (&), eax ќе содржи адреса.
- Бидејќи модулот за генерирање на код смета дека кодот е и синтаксички и семантички точен, вредноста во eax не е битно каква е, води сметка само вредноста во eax коректно и ефикасно да помине низ темплејтите.

Повикување на функции

- Повикувањето на функциите се врши со користење на Intel наредбата **call**. GNU е асемблер од високо ниво и треба само да се повика името на функцијата, а линкерот ќе се погрижи за точната адреса
  - Пример call printInt
- Параметрите на функциите се поминуваат со користење на стек
  - Повикувачот е одговорен за ставање и за вадење на параметрите од стекот
- Локалните променливи исто така се стават на стек
  - Редоследот по кој се стават локалните променливи и параметрите на стекот се нарекува рамка на стекот
- Овие работи се слични како во C.

Повикување на функции

- Функцијата која се повикува го користи ESP регистерот за да покаже на врвот на стекот.
- EBP регистерот е основниот покажувач кон рамката на стекот
- Параметрите се пуштаат на стекот од десно на лево (последниот се пушта прв)
- Вредноста која се враќа (ако е од 4 бајти и помала) се зачувува во EAX регистерот
  - За поголеми, повикувачот пуште екстра прв аргумент кон функцијата која се повикува, кој е адреса на локацијата каде вредноста што се враќа е зачувана.

```
/* vec3 is a structure of
 * 3 floats (12 bytes). */
struct vec3
{
    int x, y, z;
};

/* f is a function that returns
 * a vec3 struct: */
vec3 f( int a, int b, int c );
```

```
v = f ( 1, 0, 3 );
```

се трансформира во

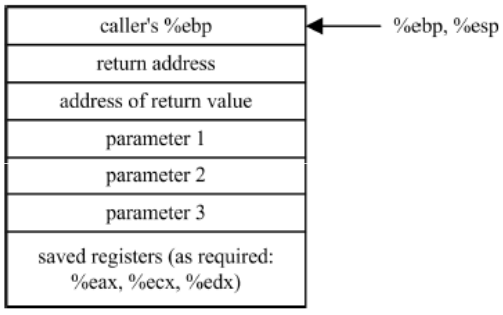
```
f( &v , 1, 0, 3 );
```

Повикување на функции

- Секоја функција си има свој сопствен стек, па содржината во него е прилично сигурна
- Но регистрите ќе бират презапишани од страна на функцијата која се повикува, па повикувачот мора сите потребни вредности пак да ги пушти во стекот
  - Ако повикувачот сака да ги зачува eax, ecx и edx регистрите, прво треба да ги пушти на стекот
  - Потоа ги пушта аргументите (од десно на лево)
  - На крај, кога се повикува инструкцијата call, eip регистерот исто така се пушта во стекот, што значи дека вратената адреса е на врвот на стекот.

Повикување на функции

- Иако повикувачот сработува голем дел од рабоата за пополнување на стекот, функцијата која се повикува треба да сработи уште неколку работи:
- Повиканата функција креира простор за локалните променливи
  - И ги става нивните иницијални вредности ако ги има
- Потоа ја зачувува содржината на ebx, esi и edi и ги става esp и ebp да покажуваат на врвот и дното на стекот, соодветно



Повикување на функции

- За да додели меморија за локалните променливи и привремените складови, повиканата функција само од esp го одзема бројот на бајтови кои се потребни за доделувањето.
- На крај ги пушта ebx, esi и edi на стекот,
- Во текот на извршувањето на функцијата, покажувачот на стекот esp оди горе долу, но ebp регистерот е фиксен, па функцијата секогаш може да се повика на него
  - Пример за првиот аргумент [ebp+8].

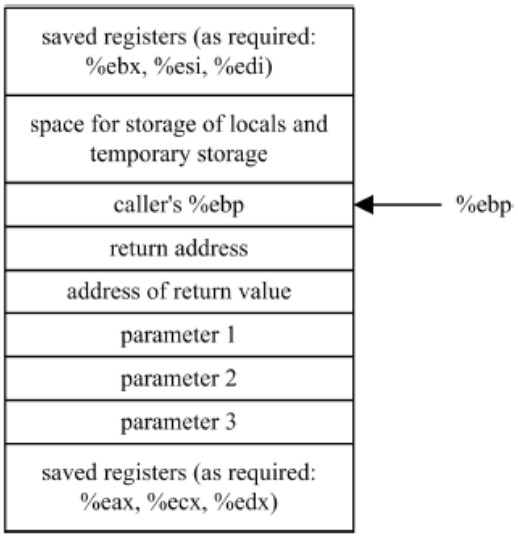


Figure 12.2: Stack Frame With Local Variables

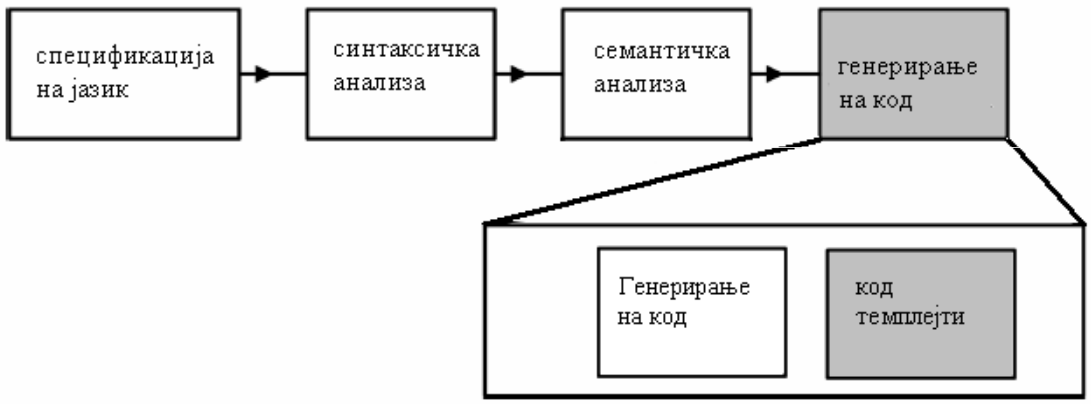
Повикување на функции

- Повиканата функција сега го чисти стекот
  - За чистење на параметрите на функцијата е задолжен повикувачот
- Повикуваната функција го прави следново
  - ги зачувува вратените вредности во eax, или во екстра параметар;
  - Ги враќа ebx, esi и edi регистрите ако е тоа потребно.
- Стекот треба да се врати во првобитната форма.

Структури за контрола на протокот

- За генерирање на споредби и условни прескокнувања, модулот за генерирање на код работи со if/then/else структури
  - Скоковите одат кон места кои претходно се обележани (пред блоковите then и else.
- Јамките се имплементирани на тој начин што прво генерираат лабела до која треба да скокнат после секоја итерација.
  - После тоа се генерира кодот за споредба, потполно исто како за if изразите.
  - Потоа блокот за код на лупата се генерира
  - По него следува скокот до лабелата која е веднаш пред кодот за споредба
  - Лупата завршува со последна лабела каде пто треба да се скокне кога се излегува од лупата.

Код Темплејт



Собирање

- Inger
  - expr + expr
- Пример
  - 3 + 5
- Асемблер
  1. Левиот израз се евалуира и се зачувува во eax.
  2. movl %eax, %ebx
  3. Десниот израз се евалуира и се зачувува во eax.
  4. addl %ebx, %eax
- Опис: Резултатот од левиот израз се додава на резултатот од десниот и се зачувува во eax.

Одземање

- Inger
  - expr - expr
- Пример
  - 8-3
- Асемблер
  1. Левиот израз се евалуира и се зачувува во eax.
  2. movl %eax, %ebx
  3. Десниот израз се евалуира и се зачувува во eax.
  4. subl %ebx, %eax
- Опис: Резултатот од десниот израз се одзема од резултатот на левиот и се зачувува во eax.

Множење

- Inger
  - expr * expr
- Пример
  - 8*3
- Асемблер
  1. Левиот израз се евалуира и се зачувува во eax.
  2. movl %eax, %ebx
  3. Десниот израз се евалуира и се зачувува во eax.
  4. imul %ebx
- Опис: Резултатот од левиот израз се множи со резултатот од десниот и се зачувува во eax.

Делење

- Inger
  - expr / expr
- Пример
  - 32/8
- Асемблер
  1. Левиот израз се евалуира и се зачувува во eax.
  2. movl %eax, %ebx
  3. Десниот израз се евалуира и се зачувува во eax.
  4. xchgl %eax, %ebx
  5. xorl %edx, %edx
  6. idiv %ebx
- Опис: Резултатот од левиот израз се дели со резултатот од десниот и резултатот се зачувува во eax.

Модули

- Inger
  - expr % expr
- Пример
  - 14%3
- Асемблер
  1. Левиот израз се евалуира и се зачувува во eax.
  2. movl %eax, %ebx
  3. Десниот израз се евалуира и се зачувува во eax.
  4. xchgl %eax, %ebx
  5. xorl %edx, %edx
  6. idiv %ebx
  7. movl %edx, %eax
- Опис: Резултатот од левиот израз се дели со резултатот од десниот и остатокот од делењето се зачувува во eax.

Негација

- Inger
  - - expr
- Пример
  - -3
- Асемблер
  1. Изразот се евалуира и се зачувува во eax.
  2. neg %eax
- Опис: Резултатот од изразот се негира и се зачувува во eax.

Леви поместувања

- Inger
  - expr << expr
- Пример
  - 256 << 2
- Асемблер
  1. Левиот израз се евалуира и се зачувува во eax.
  2. movl %eax, %ebx
  3. Десниот израз се евалуира и се зачувува во eax.
  4. xchgl %eax, %ecx
  5. sall %cl, %eax
- Опис: Резултатот од левиот израз се поместува n битови на лево, каде што n е резултатот од десниот израз. Резултатот се зачувува во eax.

If-Then-Else

Inger

```
if ( expr )
{
    // blok
}
// opcionen del
else
{
   // blok
}
```

Пример

```
int a = 2;
if ( a == 1 )
{
    a = 5;
}
else
{
    a = a - 1;
}
```

- Асемблер
- Кога има само блок then:
  1. Изразот се евалуира и се зачувува во eax.
  2. **cmpl** $0, %**eax**
  3. **je** .LABEL0
  4. Се генерира кодот за блокот.
  5. .LABEL0:
- Кога има блок else:
  1. Изразот се евалуира и се зачувува во eax.
  2. **cmpl** $0, %**eax**
  3. **je** .LABEL0
  4. Се генерира кодот за блокот.
  5. **jmp** .LABEL1
  6. .LABEL0:
  7. Се генерира кодот за блокот else.
  8. .LABEL1:

Опис: извршувањето на условниот код се реализира со условно прескокнување до лабелите

Inger

```
while( expr ) do
{
    // Код блок
}
```

Пример

```
int i = 5;
while( i > 0 ) do
{
    i = i - 1;
}
```

- Асемблер
  1. Изразот се евалуира и зачувува во eax.
  2. .LABEL0:
  3. cmpl $0, %eax
  4. je .LABEL1
  5. Се генерира кодот за блокот
  6. jmp .LABEL0
  7. .LABEL1

Опис: Се евалуира изразот и додека резултатот е точен се извршува кодот за блокот.

Користење на функција

- Inger
  func(arg1 , arg2 , argN);
- Пример
  printInt (4);
- Асемблер
  1. се евалуираат изразите за секој од аргументите, се зачувуваат во eax, и се пуштаат по стекот.
  2. movl %ebp, %ecx
  3. Се определува локацијата на стекот.
  4. Се повикува printInt (името на функцијата)
  5. Се пресметува бројот на бајти за секој аргумент
  6. addl $4, %esp (4 е бројот на бајти 4)

Имплементација на функција

- Inger

  func : type ident1 , type ident2 , type identN ! returntype
  {

  implementacija
  }

- Пример

  square : int i -> int
  {
  return(i * i);
  }

Асемблер

1. .globl square (square е ime na funkcijata)
2. .type square , @function
3. square :
4. pushl %ebp
5. movl %esp, %ebp
6. Се бројат бројот на бајти потребни за секој параметар
7. subl $4, %esp (бројот на потребни бајти е 4)
8. Се генерира кодот
9. Leave
10. Ret

Опис

Бројот на потребни бајти се пресметува и се вади од esp регистерот за да лоцира простор во стекот.

Идентификатор

- Inger

  identifier
- Пример

  i
- Асемблер

  за глобална променлива
  1. Изразот се евалуира и се зачувува во eax
  2. movl i, %eax (името на променливата е i)

  За локална променлива
  1. movl %ebp, %ecx
  2. Се определува локацијата на стекот
  3. addl $4, %ecx (офсетот на стекот е 4)
  4. movl (%ecx), %eax

- Опис

  За глобална промелнива е лесно да се генерира кодот затоа што се користи само имете на идентификаторот. За локачни променливи треба да се определи неговата позиција во стекот.

Доделување

- Inger
  identifier = expr;
- Пример
  i = 12;
- Асемблер
  за глобална променлива
  1. Изразот се евалуира и се зачувува во eax
  2. movl i, %eax (името на идентификаторот е i)

  За локална променлива
  1. Изразот се евалуира и се зачувува во eax
  2. Се определува локацијата на стекот
  3. movl %eax, 4(%ebp) (офсетот на стекот е 4)
  4. movl (%ecx), %eax

Декларација на глобални променливи

- Inger
  type identier = initializer ;
- Пример
  int i = 5;
- Асемблер
  За глобална променлива:
  1. .data
  2. .globl i (името на идентификаторот е i)
  3. .type i ,@object)
  4. .size i ,4 (типот има големина од 4 бајти)
  5. a:
  6. .long 5 (иницијалната вредност е 5)

Еднаквост

- Inger
  expr == expr
- Пример
  i == 3
- Асемблер
  1. Левиот израз се евалуира и се става во eax.
  2. movl %eax, %ebx
  3. Десниот израз се евалуира и се става во eax.
  4. cmpl %eax, %ebx
  5. movl $0, %ebx
  6. movl $1, %ecx
  7. cmovne %ebx, %eax
  8. cmove %ecx, %eax

- Опис: Се пресметуваат изразите и се споредуваат. Ако се исти во eax се додава 1, инаку се додава 0.