Методи на Транслация

Генерация на код

доц. д-р Александър Пенев

Генерация на Код

Генерация

По време на анализа и обработката на извлечените от входната програма данни, транслаторът обикновено преминава през няколко етапа на генериране на междинни структури от данни:

- Генериране на абстрактно синтактично дърво и символна таблица – това е вид (вътрешно) представяне на входната програма. То е зависимо основно от входния език и е на сравнително „високо“ ниво на абстракция;
- Генериране на три адресен код (TAC, SSA и др.) – това е вид (вътрешно) представяне на изходната програма. Прави се на базата на абстрактното дърво и символната таблица. Зависи основно от възможностите на бВИМ т.е. изходният език;
- Генериране (и запис) на изходната програма – генерация на код;

Отговорности на генерацията на код

1. Генерация на машинни инструкции
   - избор на правилните инструкции
   - избор на правилния вид адресация
2. Транслация на контролните структури (if, while, ...) в преходи
3. Определяне на стек фреймовете за локалните променливи
4. Възможна оптимизация
5. Изходен файл

Известни стратегии

1. Изучаване на целевата машина
   - регистри, формат на данните, видове адресации, инструкции, формат на инструкциите, ...
2. Дизайн на структурите, използващи се по време на изпълнение
   - разпределяне на стек фреймове, разпределяне на глобалната област за данни, ...
3. Реализиране на буфер на кода
   - кодиране на инструкциите, ...
4. Реализиране на разпределяне на регистрите
   - не е пряко свързано с SimpleC#, защото се използва стекова машина.

Известни стратегии

5. Реализиране на генерация на код:
   - зареждане на стойности и адреси в регистри (или върху стека за изчисления);
   - обработка на означения за достъп до елементи (x.y, a[i], …);
   - транслация на изрази;
   - управление на преходи и етикети;
   - транслация на изречения;
   - транслация на методи/подпрограми и предаване на параметри;

Зареждане на Стойности и Адреси

Зареждане на стойности

Дадено: Типът на операнда

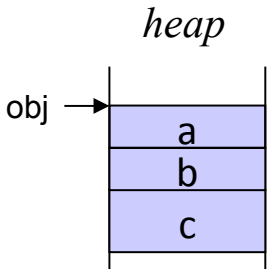
Търси се: Подходяща инструкция, която да зарежда стойността на операнда на върха на стека (или в подходящ регистър при регистрова бВИМ)

Обработка на Означения за Достъп до Елементи

Достъп до поделементи на обекти/структури, масиви и др.

Инстанции на класове

```
class X {
  int a, b;
  double c;
}
X obj = new X;
```

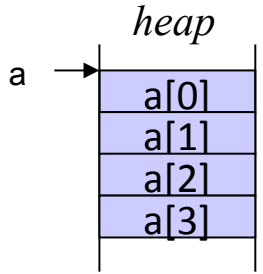


Адресирането става чрез отместване с размера на полето от началото на обекта

Масиви

```
int[] a = new int[4];
x = a[i];
```

```
ldloc a
ldloc i
ldelem
stloc x
```



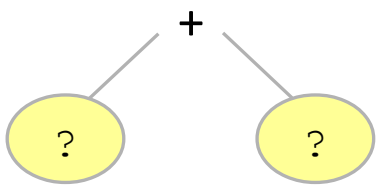
Адресирането става чрез отместване с размера на полето от началото на обекта

Транслация на Изрази

Операнди по време на генерация на код

Пример:

Трябва да съберем двете стойности



Нуждаем се от следната последователност от инструкции:

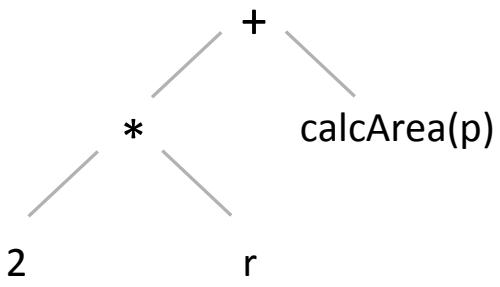
зареди операнд 1
зареди операнд 2
събери

В зависимост от вида на операндите трябва да се генерират различни инструкции за зареждане.

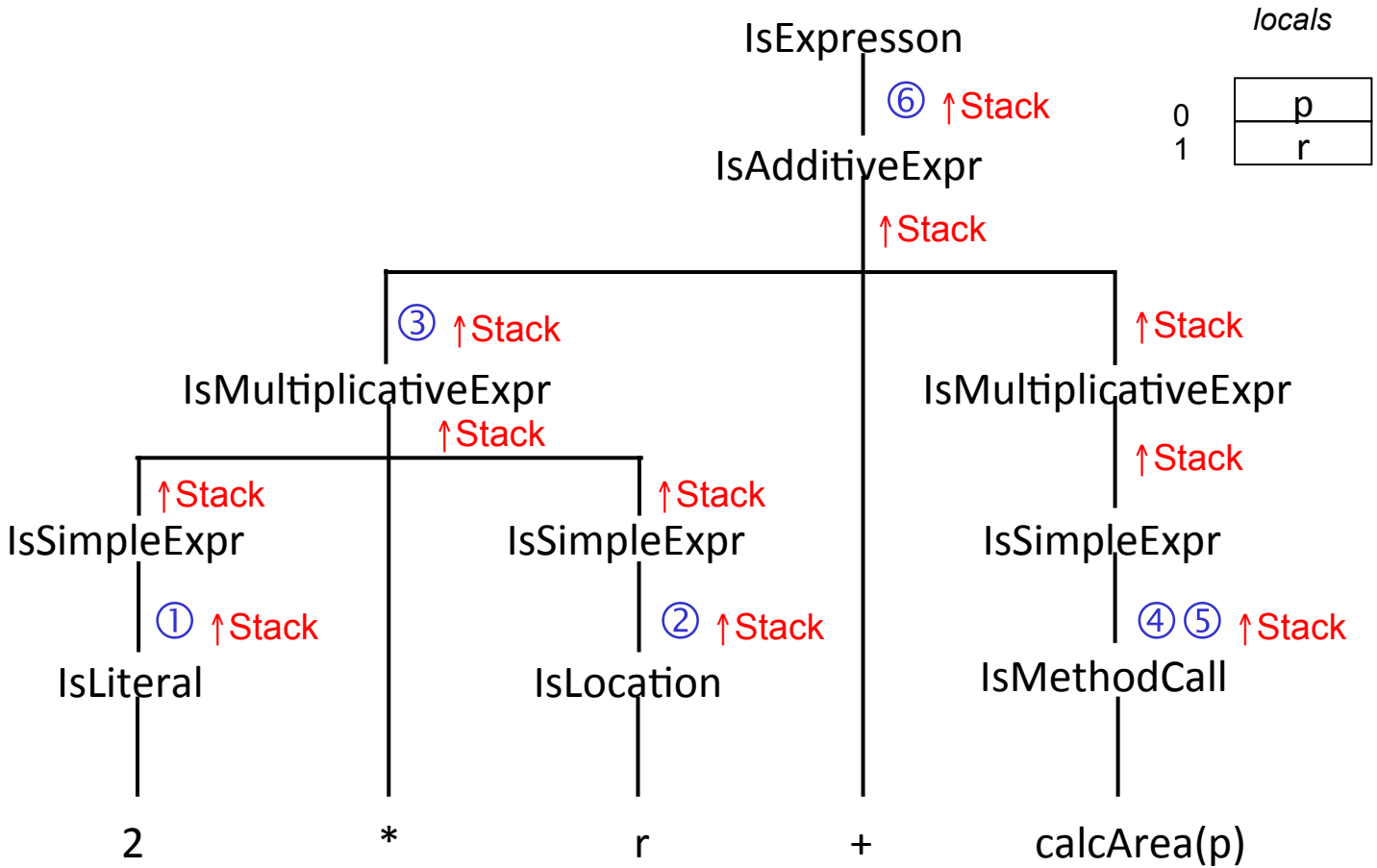
Вид на операнд	*инструкция за генерация*
❖ константи (целочислени)	`ldc.i4 c`
❖ аргументи на методи	`ldarg.s arg`
❖ локални променливи	`ldloc.s loc`
❖ глобални променливи	`ldsfld` T_{fld}
❖ елементи на масив	`ldelem`

Пример

```
2 * r + calcArea(p)
```



① `ldc.i4.2`
② `ldloc.1`
③ `mul`
④ `ldloc.0`
⑤ `call T.calcArea(float)`
⑥ `add`



Присвояване

Има няколко възможни случая на присвояване, зависещи от вида на лявата страна.
Ако лявата страна е:

- аргумент
- локална променлива
- глобална променлива
- елемент на масив

arg = expression;	local = expression;	global = expression;	a[i] = expression;
... load expr ... **`starg arg`**	*... load expr ...* **`stloc local`**	*... load expr ...* **`stsfld`** T_{global}	**`ldloc`** a **`ldloc`** i *... load expr ...* **`stelem.i2`** **`i4`** **`ref`**

Зависи от типа на елементите
(char, int, object reference)

Не трябва да се забравят проверките за контекстните условия като:

- съвместимост на типове
- дефинирана ли е променливата
- ...

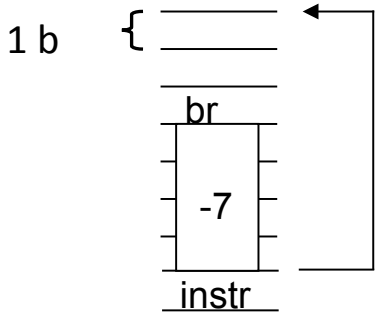
Управление на Преходи и Етикети

Преходи и Етикети

- При повечето управляващи конструкции се налага да се използват инструкции за условен и безусловен преход;
- Техният аргумент обикновено е адрес в паметта, където да продължи изпълнението на програмата. Адресацията на тези инструкции може да е:
  - Относителна;
  - Абсолютна;
- И в двата случая има проблеми с адресите, които се намират след текущия (т.е преходите „напред“);
- Тези преходи трябва да бъдат изчислени по-късно. Това се решава чрез дефиниране на т.нар. Етикети – места в програмата, които маркират различни адреси;

Преходи назад и напред

Преходи назад

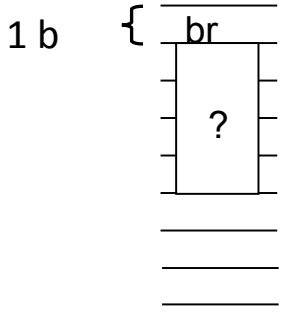


Адресът (етикета) към който ще бъде прехода е известен (защото инструкцията на тази позиция е вече генерирана)

Размер на прехода:

4 байта (CIL също има кратка форма с 1 байтов адрес) Адресът е относителен, считан от първия байт на следващата инструкция (т.е края на инструкцията за преход)

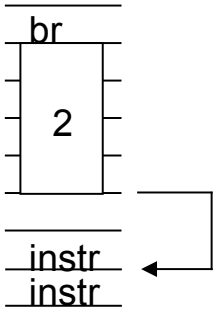
Преходи напред



Адресът-цел все още не се знае

⟹ оставя се празен

⟹ запомня се във "fixup address"



оправя се когато адресът цел стане известен (fixup)

Транслация на Изречения

Транслация на изречения

- Присвояване („=“, Assignment);
- Цикли, break и continue;
- Условни и безусловни преходи;
- Съставен оператор (Compound Statement);
- Други;

Assign Statement

Целевият псевдо-код

```
Цел = Израз
```

```
// [подготовка на адреса на резултата]
... код за Израз ...
// инстркцията зависи
// от вида на Целта. Ако тя е локална пром.:
stloc Цел
```

Пример

```
a = a - 2;
```

1 ldloc.0
2 ldc.i4.2
3 sub
4 stloc.0

while Statement

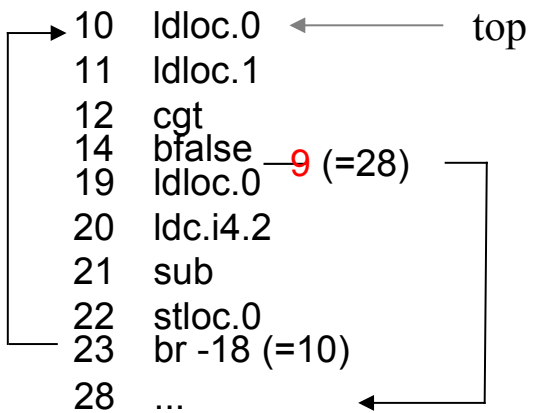
Целевият псевдо-код

```
while(Условие)
  Block
```

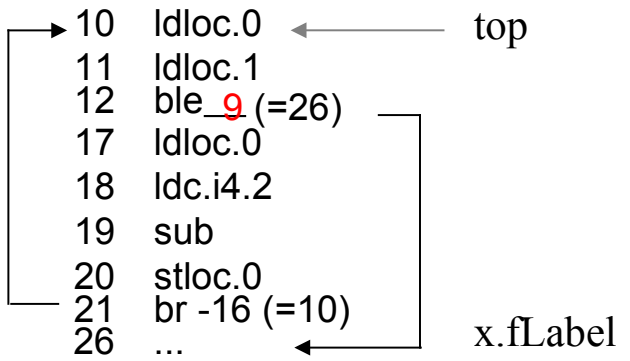
```
top:
    ... код за Условие ...
    brfalse end
    ... код за Block ...
    br top
end:
```

Пример

```
while (a > b) a = a - 2;
```



или



if Statement

Целеви псевдо-код

```
if (Условие)
Block
```

```
    ... Условие ...
    brfalse end
    ... Block ...
end:
```

```
if (Условие)

Block

else
Block
```

```
    ... Условие ...
    brfalse else
    ... Block ...
    br end
else:
    ... Block ...
end:
```

Пример

```
if (a > b) max = a; else max = b;
```

```
10  ldloc.0
11  ldloc.1
12  cgt
14  bfalse  7 (=26)
19  ldloc.0
20  stloc.2
21  br  2 (=28)
26  ldloc.1
27  stloc.2
28  ...
```

x.fLabel

end

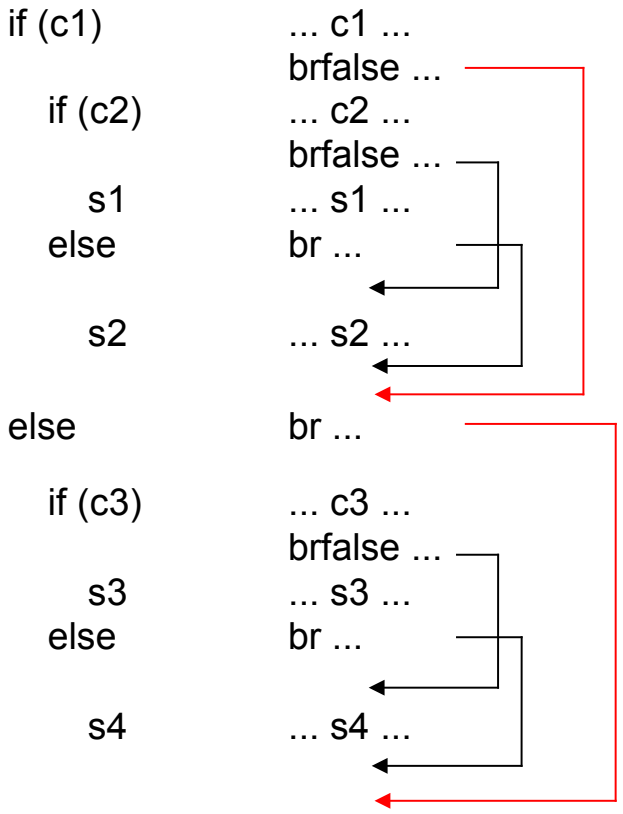
или

```
10  ldloc.0
11  ldloc.1
12  ble  7 (=24)
17  ldloc.0
18  stloc.2
19  br  2 (=26)
24  ldloc.1
25  stloc.2
26  ...
```

x.fLabel

end

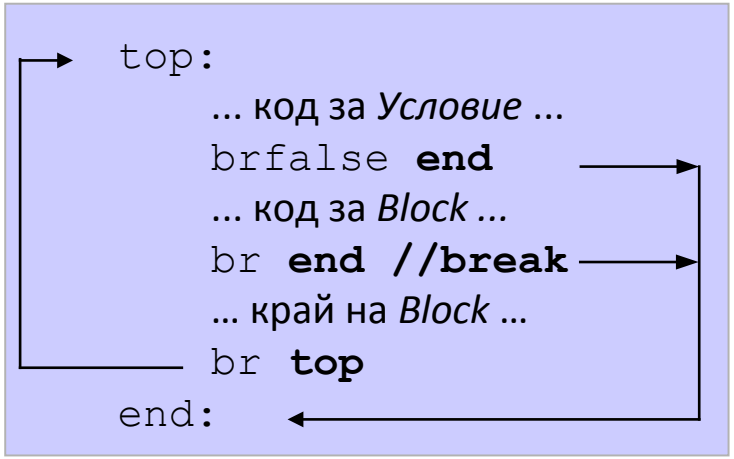
Схемата сработва и при вложени if



- Етикетите на всеки оператор трябва да се генерират уникално, за да не се получи дублиране и проблеми;
- Ако се приложи същата схема за използване на етикети и за другите видове оператори, то влагането на произволни по вид и брой оператори обикновено не води до проблеми;

break Statement

```
while(Условие)
  Block с break
```



Използва се за изход от цикъл

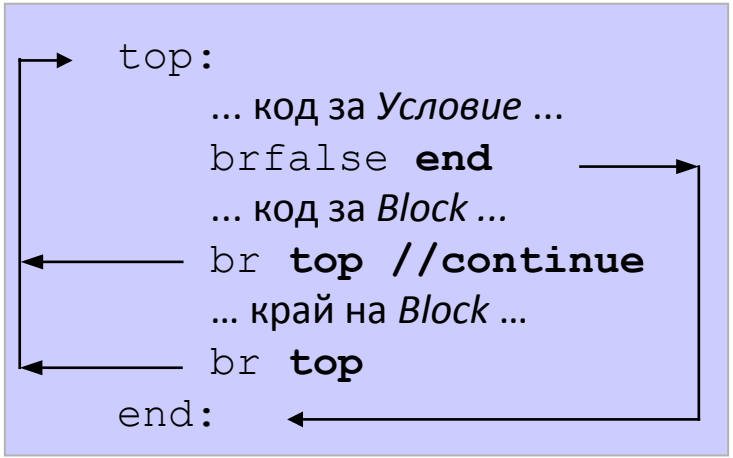
- дефинира се етикет след края на цикъла (след безусловния преход към условието);
- ако се срещне break в цикъла се прави преход на етикета;

Вложени цикли

- всеки цикъл се нуждае от свой собствен етикет;
- затова е необходимо да се поддържа стек на етикетите или етикета да се предава като атрибут;

continue Statement

```
while(Условие)
  Block с continue
```



Използва се за преход към началото на цикъла т.е. следваща итерация:

- дефинира се етикет в началото на цикъла (преди кода на условието);
- ако се срещне continue в цикъла се прави преход на етикета;

Вложени цикли:

- всеки цикъл се нуждае от свой собствен етикет;
- затова е необходимо да се поддържа стек на етикетите или етикета да се предава като атрибут;

Съкратено оценяване на булеви изрази

- Булевите изрази могат да съдържат оператори като && и ||
- Оценката на булев израз спира когато резултатът е известен

```
a && b && c
```

```
a
brfalse
b
brfalse
c
```

```
a || b || c
```

```
a
brtrue
b
brtrue
c
```

```
if (a && b  ||  c && d  ||  e && f) stat;
```

```
a
brfalse
b
brtrue
c
brfalse
d
brtrue
e
brfalse
f
brfalse
stat
```

Условие

два недефинирани етикета на края на *Условие*

Съставен оператор

```
{
  Statement1;
  Statement2;
  // ...
  Label: StatementN;
}
```

```
    ... код за Statement1 ...

    ... код за Statement2 ...

    //…

Label:
    ... код за StatementN ...
```

- Простата последователност от оператори (statements) се превежда като проста последователност от изпълнимия код на всеки един от тях, в реда на срещането им;
- Ако оператора е маркиран с етикет по някакъв начин в езика, то в началото на кода се поставя етикет съответстващ на този от входния език;

Транслация на Подпрограми и Предаване на Параметри

Извикване на подпрограми

Изпълнението на подпрограми се разделя на три:

- Предаване на параметри (актуални на формални);
- Извикване и изпълнение на подпрограмата;
- Връщане от подпрограма и получаване на резултатите;

Подпрограми

```
SubRoutine(Expr1,
           Expr2,
           ...
           ExprN);
```

```
... код за Expr1 ...
... код за Expr2 ...
//  ...
... код за ExprN ...
call SubRoutine
 // Result handle
 [pop]
```

- В зависимост от изразите и вида формални параметри, предаването на актуалните на формалните параметри може да стане по стойност или по адрес;
- Ако функцията не връща стойност т.е. типа и е void (или е процедура както се наричат този вид подпрограми в Паскал) то обработка на резултата не се извършва;
- Ако функцията връща стойност, но се извиква (извън израз) като процедура, евентуално полученият резултат трябва да се игнорира;

Буфер на Кода и Запис на Изходния Файл

Буфер на кода

- Генерираните изходни инструкции (на изходния език) се записват в буфера на кода (това в по-простите компилатори може да е и директно изходния файл);
- Когато програмата (модула) е готов, той се записва в изходния файл, като се форматира според правилата на изходния език (в текстов или бинарен вид);