



代码生成： 寄存器计算机及其代码生成

编译原理

华保健

bjhua@ustc.edu.cn

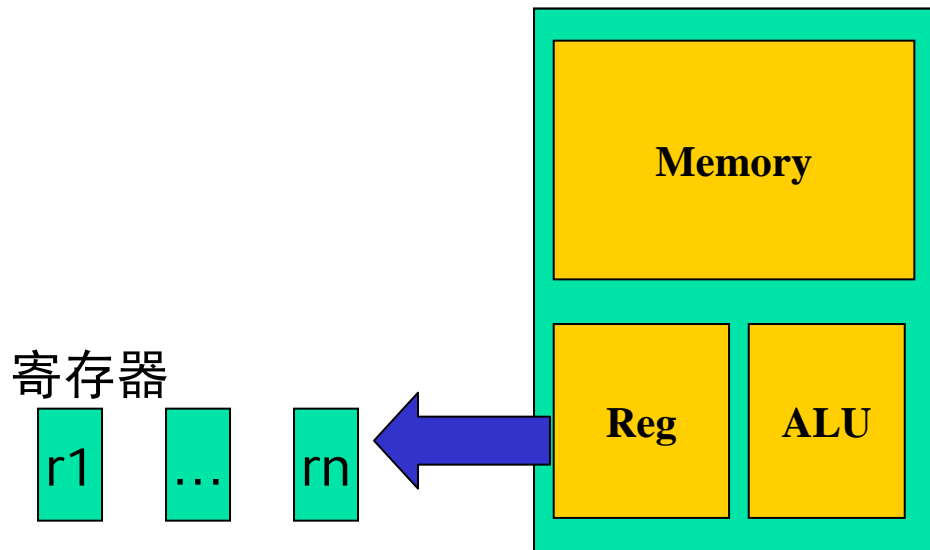


寄存器计算机

- 寄存器计算机是目前最流行的机器体系结构之一
 - 效率很高
 - 机器体系结构规整
- 机器基于寄存器架构：
 - 典型的有16、32或更多个寄存器
 - 所有操作都在寄存器中进行
 - 访存都通过load/store进行
 - 内存不能直接运算

寄存器计算机Reg的结构

- 内存
 - 存放溢出的变量
- 寄存器
 - 进行运算的空间
 - 假设有无限多个
- 执行引擎
 - 指令的执行



寄存器计算机的指令集

// 指令的语法

s -> movn n, r

| mov r1, r2

| load [x], r

| store r, [x]

| add r1, r2, r3

| sub r1, r2, r3

| times r1, r2, r3

| div r1, r2, r3

} 数据移动

} 访存

} 算术运算

寄存器

r1

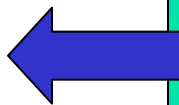
...

rn

Memory

Reg

ALU





变量的寄存器分配伪指令

- Reg机器只支持一种数据类型int，并且给变量x分配寄存器的伪指令是：
 - .int x
- 在代码生成的阶段，假设Reg机器上有无限多个寄存器
 - 因此每个声明变量和临时变量都会占用一个（虚拟）寄存器
 - 把虚拟寄存器分配到物理寄存器的过程称为寄存器分配

递归下降代码生成算法： 从C--到Reg

```
P -> D S
D -> T id; D
    |
T -> int
    | bool
S -> id = E
    | printi (E)
    | printb (E)
E -> n
    | id
    | true
    | false
    | E + E
    | E && E
```

// 要写如下几个

// 递归函数：

```
void Gen_P(D S);
void Gen_D(T id; D);
void Gen_T(T);
void Gen_S(S);
R_t Gen_E(E);
```

// 指令的语法

```
s -> movn n, r
    | mov r1, r2
    | load [x], r
    | store r, [x]
    | add r1, r2, r3
    | sub r1, r2, r3
    | times r1, r2, r3
    | div r1, r2, r3
```

递归下降代码生成算法： 表达式的代码生成

// 不变式：表达式的值在函数返回的寄存器中

```
R_t Gen_E(E e)
    switch (e)
        case n: r = fresh();
                emit ("movn n, r");
                return r;
        case id: r = fresh ();
                emit ("mov id, r");
                return r;
        case true: r = fresh ();
                  emit ("movn 1, r");
                  return r;
        case false: r = fresh ();
                    emit ("movn 0, r");
                    return r;
```

// 下页继续

```
P -> D S
D -> T id; D
    |
T -> int
    | bool
S -> id = E
    | printi (E)
    | printb (E)
E -> n
    | id
    | true
    | false
    | E + E
    | E && E
```

递归下降代码生成算法： 表达式的代码生成

// 不变式：表达式的值在函数返回的寄存器中

```
R Gen_E(E e)
  switch (e)
    case e1+e2: r1 = Gen_E(e1);
                r2 = Gen_E(e2);
                r = fresh();
                emit ("add r1, r2, r");
                return r;
    case e1&&e2: r1 = Gen_E(e1);
                r2 = Gen_E(e2);
                r = fresh();
                emit ("and r1, r2, r");
                return r; // 非短路
```

```
P -> D S
D -> T id; D
    |
T -> int
    | bool
S -> id = E
    | printi (E)
    | printb (E)
E -> n
    | id
    | true
    | false
    | E + E
    | E && E
```


递归下降代码生成算法： 语句的代码生成

```
Gen_S(S s)
  switch (s)
    case id=e: r = Gen_E(e);
               emit("mov r, id");
               break;
    case printi(e): r = Gen_E(e);
                   emit ("printi r");
                   break;
    case printb(e): r = Gen_E(e);
                   emit ("printb r");
                   break;
```

```
P -> D S
D -> T id; D
   |
T -> int
   | bool
S -> id = E
   | printi (E)
   | printb (E)
E -> n
   | id
   | true
   | false
   | E + E
   | E && E
```

递归下降代码生成算法： 类型的代码生成

// 不变式：只生成.int类型

```
Gen_T(T t)
    switch (t)
        case int: emit (".int");
                  break;
        case bool: emit (".int");
                  break;
```

```
P -> D S
D -> T id; D
    |
T -> int
    | bool
S -> id = E
    | printi (E)
    | printb (E)
E -> n
    | id
    | true
    | false
    | E + E
    | E && E
```

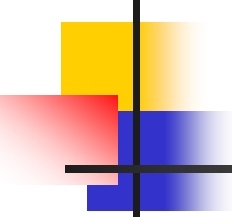
递归下降代码生成算法： 变量声明的代码生成

// 不变式：只生成.int类型

```
Gen_D(T id; D)
  Gen_T (T);
  emit (" id");
  Gen_D (D);
```

```
P -> D S
D -> T id; D
    |
T -> int
    | bool
S -> id = E
    | printi (E)
    | printb (E)
E -> n
    | id
    | true
    | false
    | E + E
    | E && E
```

递归下降代码生成算法： 程序的代码生成



```
Gen_P(D S)
  Gen_D (D);
  Gen_S (S);
```

```
P -> D S
D -> T id; D
    |
T -> int
    | bool
S -> id = E
    | printi (E)
    | printb (E)
E -> n
    | id
    | true
    | false
    | E + E
    | E && E
```



示例

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

```
x = 1+2+3+4;  
y = 5;  
z = x + y;  
y = z * x;
```

```
.int x  
.int y  
.int z
```

```
1:  movn 1, r1  
3:  movn 2, r2  
4:  add r1, r2, r3  
5:  movn 3, r4  
6:  add r3, r4, r5  
7:  movn 4, r6  
8:  add r5, r6, r7  
9:  mov r7, x  
10: movn 5, r8  
11: mov r8, y
```



如何运行生成的代码？

- 写一个虚拟机（解释器）
- 在真实的物理机器上运行：
 - 需进行寄存器分配