CPSC 323 Compilers and Languages

Instructor: Susmitha Padda

Credits : Doina Bein

Chapter 7. Object Code Generation

7.1 Introduction

7.2 Blind Generation

7.3 Context Sensitive Generation

- a) Live/next-use
- b) Register Management
- c) OC Generation

7.4 Instruction Sequencing

7.1 Introduction

- Object Code is a phase where the Intermediate Code is translated into the language of the target machine.
- In our case 3AC -> Assembly Code
- Types of Assembly Instructions:
  - Memory-to-Register
  - Register-to-Register
  - Memory-to-Memory
- In our case , assume NO M-M instructions
- Need at least one register per instruction
- Issues are:

1. Which instruction to use
2. How to avoid redundant operations
3. How to manage the register

7.2 Blind Generation

- Idea: For each 3AC instruction, we have a “template” of assembly instructions
- Example:

  3AC: x = y + z
- Template:

MOV Rx, y
ADD Rx, z
STO x, Rx
- Problem is that we could have redundant instructions:
- Example:

a = b+c;
d = a+b;
- So, we generate 6 instructions:

MOV Rx, b
ADD Rx, c
STO a, Rx
MOV Rx, a (NOT necessary – not an “intelligent use”)
ADD Rx, b
STO d, Rx

7.3 Context Sensitive Generation

- IDEA: Makes use of values that exist in the register, and if a value is in a register and is going to be used later, keep it in the register otherwise clear the register.
- Issues: How do we keep track of?

a) Which registers are used and what do they hold?

b) Where the current value of a variable can be found?

c) Which variable will be used later and where?

d) Which variables in the register must be saved for later purpose, if it has to give up the register?

- Solutions:

a) Register Association table hold information about the variables

b) Address table holds all infos about the variables (where the value is stored (R or M. And also which R)

However, to address (c) and (d) we need to have the concepts of live and next-use.

a) Live/Next-Use

- *Live*: A variable is said to be “Live” if it is going to be used again
- *Next-Use*: where it is going to be used

- Q: How do we find this information?

  One way: By searching the instructions from the top

  Example: source d= (a-b) + (c-a) – (d+b)*(c+1)

3AC:

1. u = a-b u(L, 3), a(L, 2), b(L, 4)
2. v = c-a
3. w = u+v
4. x = d+b
5. y = c+1
6. z = x*y
7. d = w-z

However, the operation is O(N*N)

- More efficient way: Start from the last instruction backwards and utilize symbol table
- Step 1. In the symbol table, mark all user-created variables “live” and temporaries with “dead”
- Step 2. For each 3 AC instruction a = b (op) c backwards do
  - look-up the symbol table and attach Live/next-use information
  - Update the symbol table as follows
    - mark the target “a” to be “dead”
    - mark the operands “b” and “c” “ to be “live”
    - and set next-use to be the current instruction

u = a-b
v = c-a
w = u+v
x = d+b
y = c+1
z = x*y
d = w-z

Ex. source: d = (a-b) + (c-a) – (d+b)*(c+1)

1. u = a-b u(L, 3), a(L, 2), b(L, 4)
2. v = c-a v(L,3), c(L, 5), a(L,N)
3. w = u+v w(L, 7), U(D, N), v(D, N)
4. x = d+b x(L, 6), d(D, N), b(L, N)
5. y = c+1 y(L, 6), c(L, N)
6. z = x*y z(L, 7), x(D, N), y(D, N)
7. d = w-z d(L, N), w(D, N), z(D,N)

<u>Symbol Table (initial)</u>

Variable	Liveness	Next-use
a	L	N
b	L	N
c	L	N
d	L	N
u	D	N
v	D	N
w	D	N
x	D	N
y	D	N
z	D	N

7.3 Context Sensitive Generation

- IDEA: Makes use of values that exist in the register, and if a value is in a register and is going to be used later, keep it in the register otherwise clear the register.
- Issues: How do we keep track of?

a) Which registers are used and what do they hold?

b) Where the current value of a variable can be found?

c) Which variable will be used later and where?

d) Which variables in the register must be saved for later purpose, if it has to give up the register?

- Solutions:

a) Register Association table hold information about the variables

b) Address table holds all infos about the variables (where the value is stored (R or M. And also which R)

However, to address (c) and (d) we need to have the concepts of live and next-use.

a) Live/Next-Use

- *Live*: A variable is said to be "Live" if it is going to be used again
- *Next-Use*: where it is going to be used

- Q: How do we find this information?

  One way: By searching the instructions from the top

  Example: source d= (a-b) + (c-a) – (d+b)*(c+1)

3AC:
1. u = a-b u(L, 3), a(L, 2), b(L, 4)
2. v = c-a
3. w = u+v
4. x = d+b
5. y = c+1
6. z = x*y
7. d = w-z

However, the operation is O(N*N)

- More efficient way: Start from the last instruction backwards and utilize symbol table
- Step 1. In the symbol table, mark all user-created variables “live” and temporaries with “dead”
- Step 2. For each 3 AC instruction a = b (op) c backwards do
  - look-up the symbol table and attach Live/next-use information
  - Update the symbol table as follows
    - mark the target “a” to be “dead”
    - mark the operands “b” and “c” “ to be “live”
    - and set next-use to be the current instruction

u = a-b
v = c-a
w = u+v
x = d+b
y = c+1
z = x*y
d = w-z

Ex. source: d = (a-b) + (c-a) – (d+b)*(c+1)

1. u = a-b u(L, 3), a(L, 2), b(L, 4)
2. v = c-a v(L,3), c(L, 5), a(L,N)
3. w = u+v w(L, 7), U(D, N), v(D, N)
4. x = d+b x(L, 6), d(D, N), b(L, N)
5. y = c+1 y(L, 6), c(L, N)
6. z = x*y z(L, 7), x(D, N), y(D, N)
7. d = w-z d(L, N), w(D, N), z(D,N)

<u>Symbol Table (initial)</u>

Variable	Liveness	Next-use
a	L	N
b	L	N
c	L	N
d	L	N
u	D	N
v	D	N
w	D	N
x	D	N
y	D	N
z	D	N

2.Register Management

Since we need at least one register for operation (RM, RR instruction), we have to manage the limited number of registers.

/* Find an available register for a 3AC instruction a = b (op) c */

```
function GetReg(b)
{
    If (b is in register R) and (b is dead) and (R does not hold
another variable) then return R
    else if there is an unused register R, then return R
    else /* all register are used */
        Select an occupied register R for use
        Store the content of R
        Return R
}
```

3.Context Sensitive Generation of the Code

Algorithm for Code Generation

```
Procedure GenCode (3AC: a = b (op) c, or  a =b)
{
   R = GetReg(b);
   If R is empty,  then generate “L   R, b”
   If 3AC is “a =b” then change the symbol table so that R holds also
        “a” and return;
   Find “C” using the address table
   If “c” is in memory then generate “ op    R, c”
   else if “c” is in a Register S, then
            {
            Generate “opR   R, S”
            If  “c” is dead then free the Register S
            }
  Update symbol table
  }
```

Example:

Source code : d = (a-b) + (c-a) – (d+b) * (c+1)

1. u = a-b	u(L,3), a(L,2), b(L,4)
2. r = c-a	v(L,3), c(L,5), a(L,N)
3. w = u+v	w(L,7), U(D, N), V(D,N)
4. x = d+b	x(L,6), d(D,N), b(L,N)
5. y = c+1	y(L,6), C(L,N)
6. z = x*y	z(L,7), x(D,N), y (D, N)
7. d = w – z	d (L, N), w(D, N), Z (D,N)

Register ASSN Table **Address Table**

2	a m
3	b m
4	c m
5	d m
6	u -
7	v -
8	w -
9	x -
10	y -
	z -

Procedure GenCode(3ac: a = b (op) c, or a =b)
{
R = GetReg(b);
If R is empty, then generate "L R, B"
If 3ac is "a =b" then, change the tables so that R holds also "a" and return;
Find "C" using the address table
If "C" is in memory then generate " op R, c"
Else if "C" is in a Register S, then
{
Generate "opR R, S"
If "C" is dead then free the Register S
}
Update tables
}

* Find an available register for a 3AC instruction a = b (op) c */
unction GetReg(b)

If (b is in register R) and (b is dead) and (R does not hold another variable) then return R
Else if there is an unused register R, then return R
Else /* all register are used */
Select an occupied register R for use
Store the content of R
Return R

(1) u = a –b
GetReg(a) = 2
L 2, a
S 2, b

(2) v = c –a
GetReg(c) = 3
L 3, c
S 3, a

1. u = a-b	u(L,3), a(L,2), b(L,4)
2. r = c-a	v(L,3), c(L,5), a(L,N)
3. w = u+v	w(L,7), U(D, N), V(D,N)
4. x = d+b	x(L,6), d(D,N), b(L,N)
5. y = c+1	y(L,6), C(L,N)
6. z = x*y	z(L,7), x(D,N), y (D, N)
7. d = w – z	d (L, N), w(D, N), Z (D,N)

```
Procedure GenCode(3ac: a = b (op) c, or  a =b)
{
   R = GetReg(b);
   If R is empty,  then generate "L   R, B"
   If 3ac is "a =b" then, change the tables so that R holds also
        "a" and return;
   Find "C" using the address table
   If "C" is in memory then generate " op    R, c"
   Else if "C" is in a Register S, then
           {
           Generate "opR   R, S"
           If  "C" is dead then free the Register S
           }
 Update tables
 }
```

```
/* Find an available register for a 3AC instruction a = b (op) c  */
function GetReg(b)
{
   If (b is in register R) and (b is dead) and (R does not hold another
         variable) then return R
   Else if there is an unused register R, then return R
   Else /* all register are used */
      Select an occupied register R for use
      Store the content of R
      Return R
}
```

(3) w = u + v
GetReg(u) = 2
AR 3,2   /* R2 can be freed */

(4) x = d + b
GetReg(d) = 4
L 4, d
A 4, b

1. u = a-b	u(L,3), a(L,2), b(L,4)
2. r = c-a	v(L,3), c(L,5), a(L,N)
3. w = u+v	w(L,7), U(D, N), V(D,N)
4. x = d+b	x(L,6), d(D,N), b(L,N)
5. y = c+1	y(L,6), C(L,N)
6. z = x*y	z(L,7), x(D,N), y (D, N)
7. d = w – z	d (L, N), w(D, N), Z (D,N)

```
Procedure GenCode(3ac: a = b (op) c, or  a =b)
{
   R = GetReg(b);
   If R is empty,  then generate "L   R, B"
   If 3ac is "a =b" then, change the tables so that R holds also
       "a" and return;
   Find "C" using the address table
   If "C" is in memory then generate " op    R, c"
   Else if "C" is in a Register S, then
            {
            Generate "opR   R, S"
            If  "C" is dead then free the Register S
            }
 Update tables
}
```

```
/* Find an available register for a 3AC instruction a = b (op) c  */
function GetReg(b)
{
    If (b is in register R) and (b is dead) and (R does not hold another
          variable) then return R
    Else if there is an unused register R, then return R
    Else /* all register are used */
       Select an occupied register R for use
       Store the content of R
       Return R
}
```

1. u = a-b	u(L,3), a(L,2), b(L,4)
2. r = c-a	v(L,3), c(L,5), a(L,N)
3. w = u+v	w(L,7), U(D, N), V(D,N)
4. x = d+b	x(L,6), d(D,N), b(L,N)
5. y = c+1	y(L,6), C(L,N)
6. z = x*y	z(L,7), x(D,N), y (D, N)
7. d = w – z	d (L, N), w(D, N), Z (D,N)

(5) y = c+1

GetReg(c) = 5

L 5, c

A 5, = F'1'

(6) z = x * y   /* need even-odd registers for multiply */

GetReg(x) = 4

MR 4,5

Procedure GenCode(3ac: a = b (op) c, or a =b)
{
R = GetReg(b);
If R is empty, then generate "L R, B"
If 3ac is "a =b" then, change the tables so that R holds also "a" and return;
Find "C" using the address table
If "C" is in memory then generate " op R, c"
Else if "C" is in a Register S, then
{
Generate "opR R, S"
If "C" is dead then free the Register S
}
Update tables
}

/* Find an available register for a 3AC instruction a = b (op) c */
function GetReg(b)
{
If (b is in register R) and (b is dead) and (R does not hold another variable) then return R
Else if there is an unused register R, then return R
Else /* all register are used */
Select an occupied register R for use
Store the content of R
Return R
}

1. u = a-b	u(L,3), a(L,2), b(L,4)
2. r = c-a	v(L,3), c(L,5), a(L,N)
3. w = u+v	w(L,7), U(D, N), V(D,N)
4. x = d+b	x(L,6), d(D,N), b(L,N)
5. y = c+1	y(L,6), C(L,N)
6. z = x*y	z(L,7), x(D,N), y (D, N)
7. d = w – z	d (L, N), w(D, N), Z (D,N)

(7) d = w-z
GetReg(w) =3
SR 3, 4

In total we get 11 instructions VS. 21 instructions in BLIND generation => About 45 % instruction savings

7.4 Instruction Sequencing

Sethi-Ullman Method or Minimum Use of Registers

If we need one register per 3AC instruction, it is important to minimize the usage of the registers.

Idea: We change the sequence of 3AC instructions such that the new sequence uses less number of registers

Assume We have an AST (!!), otherwise we have to create one.

Step1: Label (Give a count to) the nodes in AST as follows:

Leaves: The left node gets a count of 1 and right node gets a count of 0

Roots: If the subtrees have the same count K, then K+1

else the greater count of two

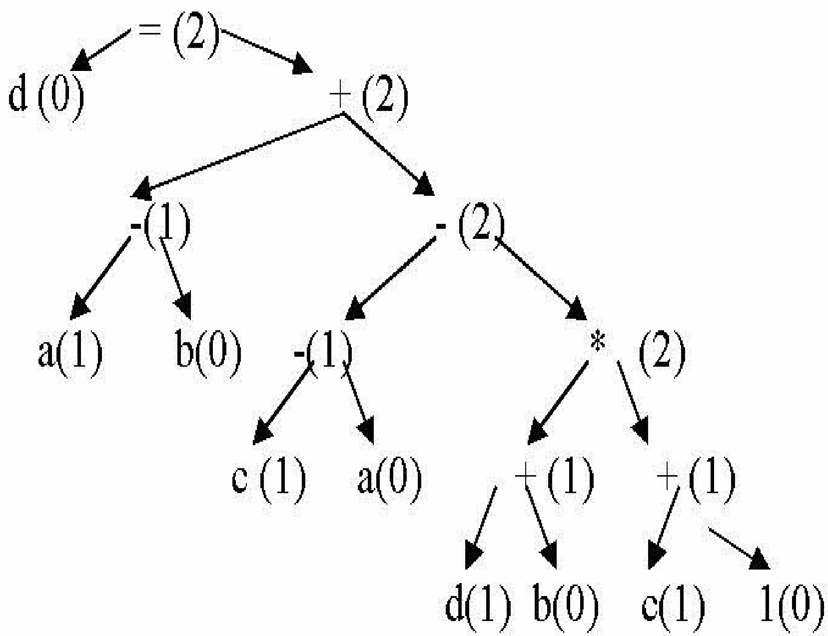
Step 2: Generate 3Acs such that more expensive subtree is evaluated first.

If the subtrees have the same count, then favor the right subtree (or left) – consistent

Step 3: Generate the OC from the new sequence

EX: $d = (a-b) + (c-a) - (d+b)*(c+1)$

Step1:



u = a-b
v = c-a
w = u+v
x = d+b
y = c+1
z = x*y
d = w-z

Step2:

1. t1 = c +1
2. t2 = d +b
3. t3 = t2 * t1
4. t4 = c –a
5. t5 = t4 – t3
6. t6 = a –b
7. d = t6 + t5

Step 3:
We can see that we need two registers only