

A code-generation algorithm

The algorithm takes as input a sequence of three-address statements constituting a basic block. For each three-address statement of the form $x := y \text{ op } z$, perform the following actions:

1. Invoke a function `getreg` to determine the location L where the result of the computation $y \text{ op } z$ should be stored.
2. Consult the address descriptor for y to determine y' , the current location of y . Prefer the register for y' if the value of y is currently both in memory and a register. If the value of y is not already in L , generate the instruction `MOV y' , L` to place a copy of y in L .
3. Generate the instruction `OP z' , L` where z' is a current location of z . Prefer a register to a memory location if z is in both. Update the address descriptor of x to indicate that x is in location L . If x is in L , update its descriptor and remove x from all other descriptors.
4. If the current values of y or z have no next uses, are not live on exit from the block, and are in registers, alter the register descriptor to indicate that, after execution of $x := y \text{ op } z$, those registers will no longer contain y or z .

Generating Code for Assignment Statements:

- The assignment $d := (a-b) + (a-c) + (a-c)$ might be translated into the following three-address code sequence:

Code sequence for the example is:



```
t := a - b
u := a - c
v := t + u
d := v + u
```

with d live at the end.

Code sequence for the example is:

Statements	Code Generated	Register descriptor	Address descriptor
		Register empty	
t := a - b	MOV a, R0 SUB b, R0	R0 contains t	t in R0
u := a - c	MOV a, R1 SUB c, R1	R0 contains t R1 contains u	t in R0 u in R1
v := t + u	ADD R1, R0	R0 contains v R1 contains u	u in R1 v in R0
d := v + u	ADD R1, R0 MOV R0, d	R0 contains d	d in R0 d in R0 and memory

Generating Code for Indexed Assignments

The table shows the code sequences generated for the indexed assignments $a := b[i]$ and $a[i] := b$

Statements	Code Generated	Cost
$a := b[i]$	MOV b(Ri), R	2
$a[i] := b$	MOV b, a(Ri)	3

Generating Code for Pointer Assignments

The table shows the code sequences generated for the pointer assignments $a := *p$ and $*p := a$

Statements	Code Generated	Cost
$a := *p$	MOV *Rp, a	2
$*p := a$	MOV a, *Rp	2

Generating Code for Conditional Statements

Statement	Code
if $x < y$ goto z	CMP x, y CJ< z /* jump to z if condition code is negative */
$x := y + z$	MOV y, R0

if x < 0 goto z ADD z, R0

MOV R0,x

CJ< z

