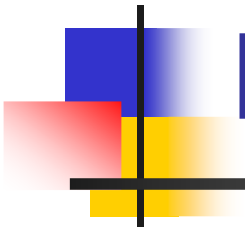
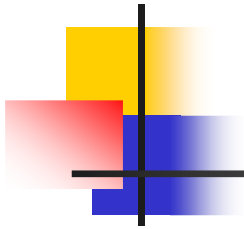


Next-use Information

- for register and temporary allocation
- remove variables from registers if not used
- statement X = Y op Z defines X and uses Y and Z
- scan each basic blocks backwards
- assume all temporaries are dead on exit and all user variables are live on exit

Algorithm to compute next use information



Suppose we are scanning
i : X := Y op Z in backward scan

1. **Attach to statement i the information currently found in the symbol table regarding the next use and live ness of x, y and z.**
2. **In the symbol table, set x to "not live" and "no next use".**
3. **In the symbol table, set y and z to "live" and the next uses of y and z to i. Note that the order of steps (2) and (3) may not be interchanged because x may be y or z.**

Example

1: t1 = a * a

2: t 2 = a * b

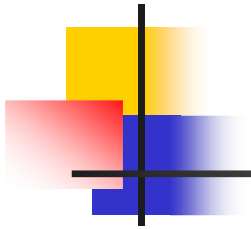
3: t3 = 2 * t2

4: t4 = t 1 + t3

5: t5 = b * b

6: t6 = t 4 + t5

7: X = t 6



Names	Liveliness	Next-use
t1	dead	4
t2	dead	3
t3	dead	4
t4	dead	6
t5	dead	6
t6	dead	7

1: t 1 = a * a

2: t 2 = a * b

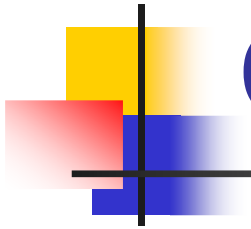
3: t2 = 2 * t2

4: t1 = t 1 + t2

5: t2 = b * b

6: t1 = t1 + t 2

7: X = t1



Code Generator

The code-generation uses descriptors to keep track of register contents and addresses for names.

- **Register descriptor**

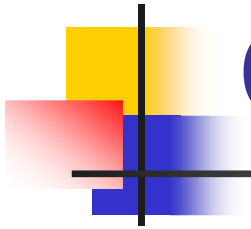
- Keep track of what is currently in each register.

-Initially all the registers are empty

- **.Address descriptor**

- Keep track of location where current value of the name can be found at runtime

- The location might be a register, stack, memory address or a set of those



Code generation algorithm

for each X = Y op Z do

invoke a function getreg to determine location L where X must be stored. Usually L is a register.

Consult address descriptor of Y to determine Y'. Prefer a register for Y'. If value of Y not already in L generate

Mov Y', L

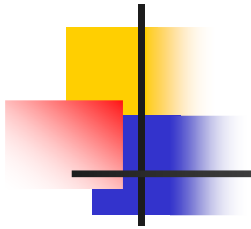
Generate

op Z', L

Again prefer a register for Z. Update address descriptor of X to indicate X is in L. If L is a register update its descriptor to indicate that it contains X and remove X from all other register descriptors.

If current value of Y and/or Z have no next use and are dead on exit from block and are in registers, change register descriptor to indicate that they no longer contain Y and/or Z

Function getreg



1. If Y is in register (that holds no other values) and Y is not live and has no next use after

   X = Y op Z

   then return register of Y for L.
2. Failing (1) return an empty register
3. Failing (2) if X has a next use in the block or op requires register then get a register R, store its content into M (by Mov R, M) and use it.
4. else select memory location X as L

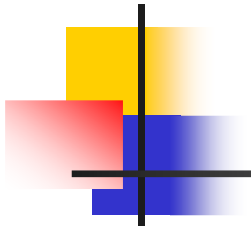
d := (a - b) + (a - c) + (a - c)

t1 = a - b

t 2 = a - c

t 3 = t 1 + t2

d = t 3 + t2



Statements	***Code generated***	***Register descriptor***	***Address descriptor***	***Cost***
		Registers empty		
$t1 := a - b$	MOV *a*, *R*0	*R*0 contains *t*1	*t*1 in *R*0	2
	SUB *b*, *R*0			2
$t2 := a - c$	MOV *a*, *R*1	*R*0 contains *t*1	*t*1 in *R*0	2
	SUB *c*, *R*1	*R*1 contains *t*2	*t*2 in *R*1	2
$t3 := t1 + t2$	ADD *R*1, *R*0	*R*0 contains *t*3	*t*2 in *R*1	1
		*R*1 contains *t*2	*t*3 in *R*0	
$d := t3 + t2$	ADD *R*1, *R*0	*R*0 contains *d*	*d* in *R*0	1
	MOV *R*0, *d*		*d* in *R*0 and memory	2
			Total Cost	12