DAG representation of basic blocks

- useful data structures for implementing transformations on basic blocks
- gives a picture of how value computed by a statement is used in subsequent statements
- good way of determining common sub-expressions
- A DAG for a basic block has following labels on the nodes

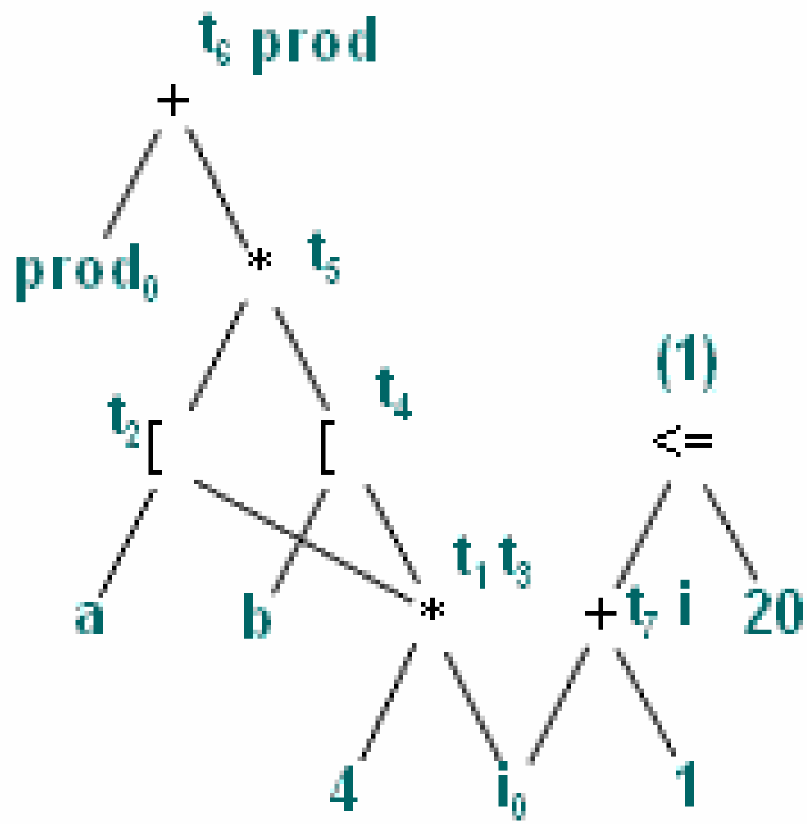
  - leaves are labeled by unique identifiers, either variable names or constants

  - interior nodes are labeled by an operator symbol

  - nodes are also optionally given a sequence of identifiers for labels

DAG representation: example

1. $t_1 := 4 * i$
2. $t_2 := a[t_1]$
3. $t_3 := 4 * i$
4. $t_4 := b[t_3]$
5. $t_5 := t_2 * t_4$
6. $t_6 := prod + t_5$
7. $prod := t_6$
8. $t_7 := i + 1$
9. $i := t_7$
10. if i <= 20 goto (1)



Three-Address Code	***Code from DAG***
$t1 := 4 * i$	$t1 := 4 * i$
$t2 := a[t1]$	$t2 := a[t1]$
$t3 : 4 * i$	
$t4 := b[t3]$	$t4 := b[t1]$
$t5 := t2 * t4$	$t5 := t2 * t4$
$t6 :=$ prod $+ t5$	prod $:=$ prod $+ t5$
prod $:= t6$	
$t7 := i + 1$	$i := i + 1$
$i := t7$	
if $i <= 20$ goto (1)	if $i <= 20$ goto (1).

Code Optimization

- The term “Code Optimization” refers to techniques a compiler can employ in an attempt to produce a better object language program.
- Optimizing Compilers : Compilers that apply code improving transformations
- Two categories
  - Machine independent
  - Machine dependent

Code Optimization (cont.)

- Properties
  - Preserve meaning
  - Speed up program
  - Transformation must be worth the effort
- Places of improvement
  - Source level
  - Intermediate code level
  - Target program level

Principle Sources of Optimization

- Local optimization – within basic block
  - Function Preserving Optimization
    - Common sub-expression elimination
    - Copy propagation
    - Dead code elimination
    - Constant folding
- Global optimization – across basic block
- Loop optimization – local
    - Code motion
    - Induction variable elimination
    - Strength reduction
- Optimization of basic blocks
  - Structure preserving
  - Use pf algebraic identities

Loop Optimization

The running time of a program may be improved if we decrease the number of instructions in an inner loop, even if we increase the amount code outside the loop.

Three techniques for loop optimization are

(i) Code motion

(ii) Induction variable elimination

(iii) Reduction in strength.

Code Motion

This transformation takes an expression that yields the same result independent of the number of times a loop is executed. (loop-invariant computation) and places the expression before the loop.

For example:

while ($i < = n$ - 2)

{

. . . .

}

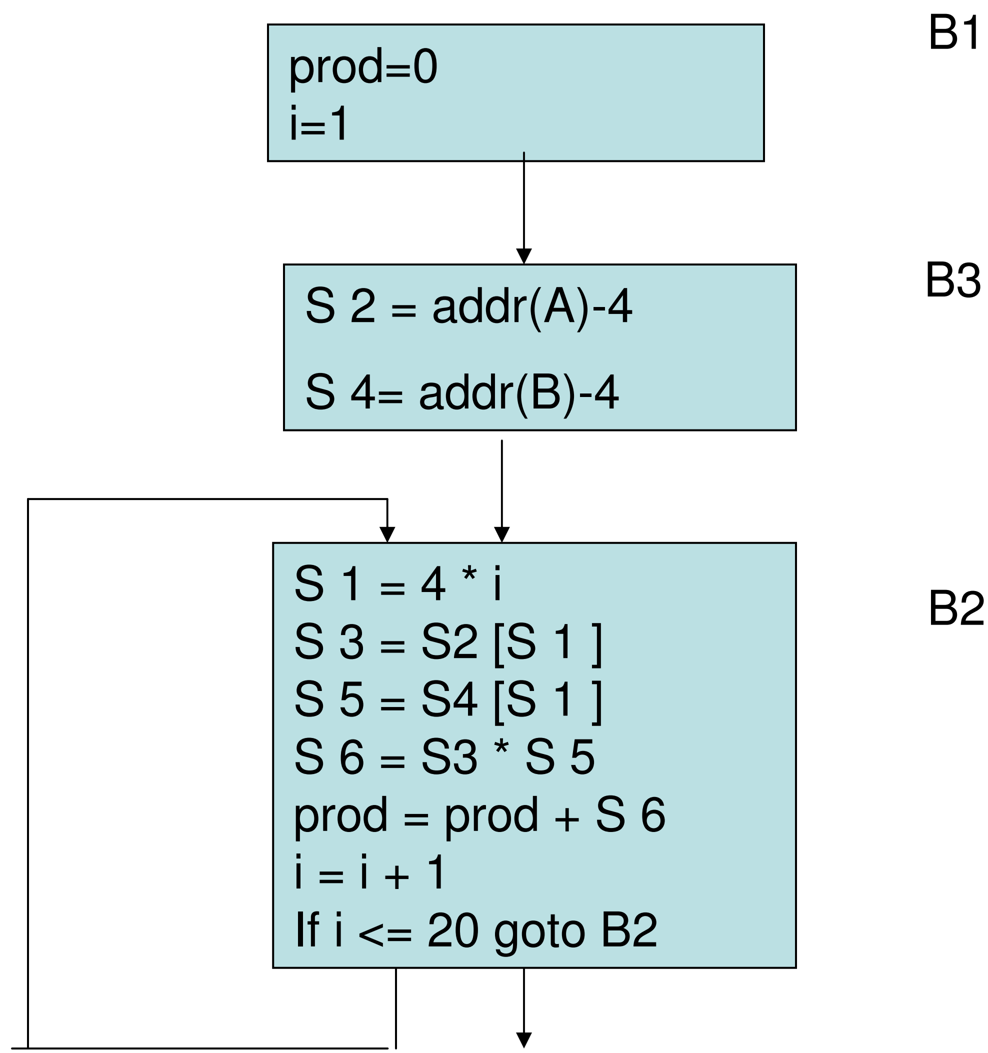
Evaluation of n - 2 is a loop-invariant computation in the above while statements. So this can be placed outside the loop.

Code motion will result in the, equivalent of

$t = n – 2$,

while ($i < = t$).

Induction variable elimination



Loop Optimization

When I takes 1,2,---,20

S1 takes 4,8,---80

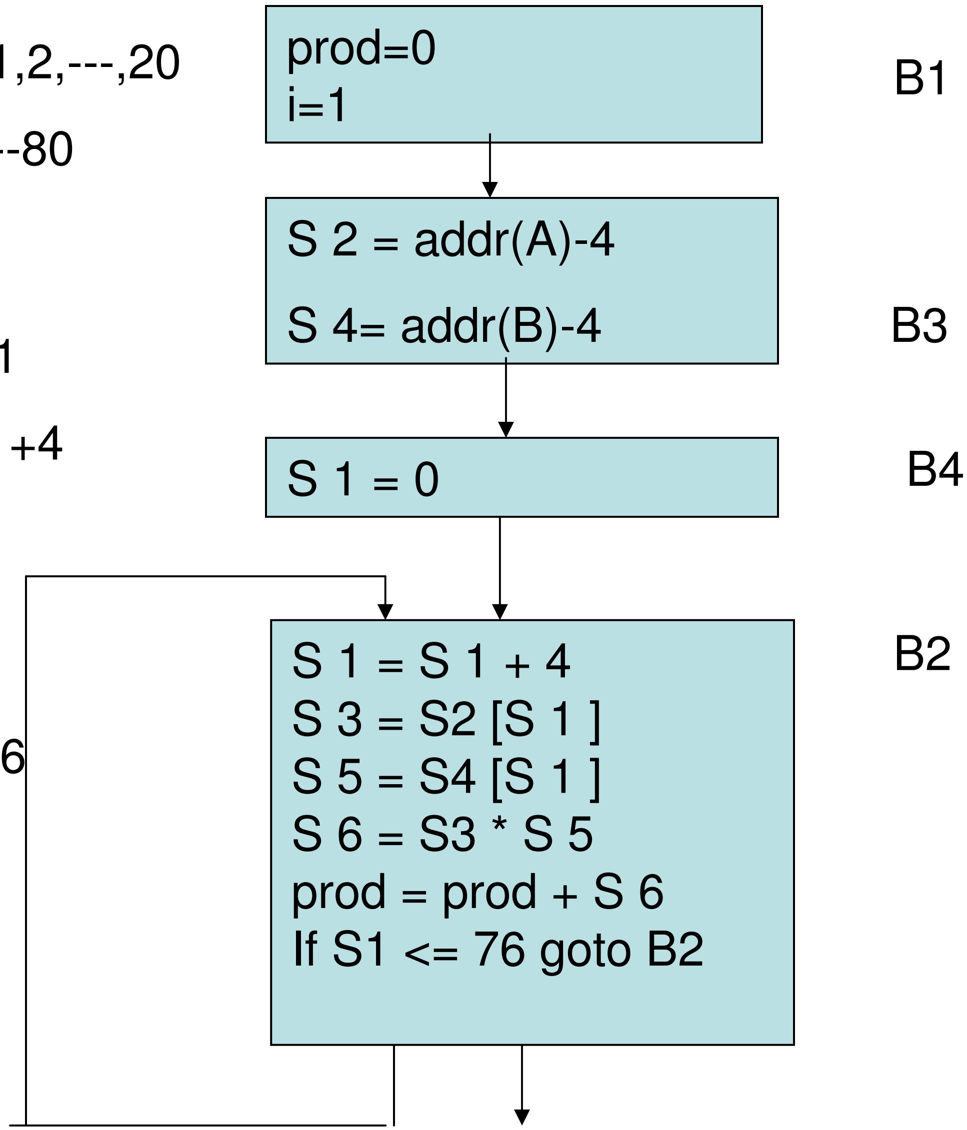
Replace 4*I

I=I+1

With s1=s1+4

i=i+1

i<=20 → S1<=76



Reduction in Strength

The higher strength operators can be replaced by lower strength operators.

(e.g.,) Replacement of multiplication by repeated addition.
For example:

```
for (i = 1 ; i < = 50, i ++ )
{
- - -
C = i × 7.
- - -
}
```

This code can replaced by using strength reduction as follows:

```
t = 7;
for (i = 1; i < = 50; i ++)
{
- - -
c = t;
t = t + 7;
}
```

Peephole Optimization

- target code often contains redundant instructions and suboptimal constructs
- examine a short sequence of target instruction (peephole) and replace by a shorter or faster sequence
- peephole is a small moving window on the target systems

Peephole optimization examples

Redundant loads and stores

Consider the code sequence

1. Move R0 , a   2.Move a, R0

Instruction 2 can always be removed if it does not have a label.

Unreachable code

- Consider following code sequence

```
#define debug 0
if (debug) {
        print debugging info
}
```

this may be translated as

```
        if debug = 1 goto L1
        goto L2
L1: print debugging info
L2:
```

Eliminate jump over jumps

```
        if debug <> 1 goto L2
                print debugging information
L2:
```

Unreachable code example

- constant propagation

```
if 0 <> 1 goto L2
        print debugging information
L2:
```

Evaluate boolean expression. Since if condition is always true the code becomes

```
goto L2
print debugging information
L2:
```

The print statement is now unreachable. Therefore, the code becomes

```
L2:
```

Peephole optimization examples.

- flow of control: replace jump sequences

```
    goto L1                         goto L2
    ...                                ...
    ...            by                  ...
L1 : goto L2                    L1: goto L2
```

- Simplify algebraic expressions

remove x := x+0 or x:=x*1

Peephole optimization examples.

Strength reduction

Replace X^2 by X*X

Replace multiplication by left shift

Replace division by right shift

Use faster machine instructions

```
replace    Add #1,R
by           Inc    R
```