3.1 Lazy Code Motion

CFG for original code:

(1) x=x+3

(2) if(y>5)

(3) z=z+x

(4) x=4

(6) y=y-5

(7) z=z+x

Entry

(9) Return z

Exit

Pass 1: CFG with Anticipation

(1) x=x+3

(3) z=z+x

(6) y=y-5

Entry

(9) Return z

Exit

(2) if(y>5)

(7) z=z+x

(4) x=4

0/0/0

1/0/0

0/0/0

1/0/0

1/0/0

1/0/1

0/0/0

0/0/0

0/0/0

1/0/0

1/0/0

1/0/0

0/1/0

(z+x)/(x+3)/(y-5)

Pass 2: Early Placement Pass

(z+x)/(x+3)/(y-5)

t=z+x

z=t

(3) z=t

(6) y=y-5

Entry

(9) Return z

Exit

(2) if(y>5)

(7) z=t

(4) x=4

0

1

0

1

1

1

0

0

0

1

1

1

0

(1) x=x+3

Earliest z+x (anticipated here but not available.)

Pass 3: Lazy Code Motion

t=z+x can be postponed to

to IN of (3), IN of (6), OUT of

(6), IN of (7).

(3) z=t

(6) y=y-5

Entry

(9) Return z

Exit

(2) if(y>5)

(7) z=t

(4) x=4

0

1

0

1

1

1

0

0

0

1

1

1

0

(1) x=x+3

Earliest z+x (anticipated here but not available.)

t=z+x

t=z+x

1

According to algorithm

Z+x is moved to the most postponable

position as long as it is not used. But intuitively

it should be placed before the branch.

Pass 4: Cleanup Pass

Assumption that compiler moves the z+x above the

branch condition.

t=z+x

(3) z=t

(6) y=y-5

Entry

(9) Return z

Exit

(2) if(y>5)

(7) z=t

(4) x=4

0

1

0

1

1

1

0

0

0

1

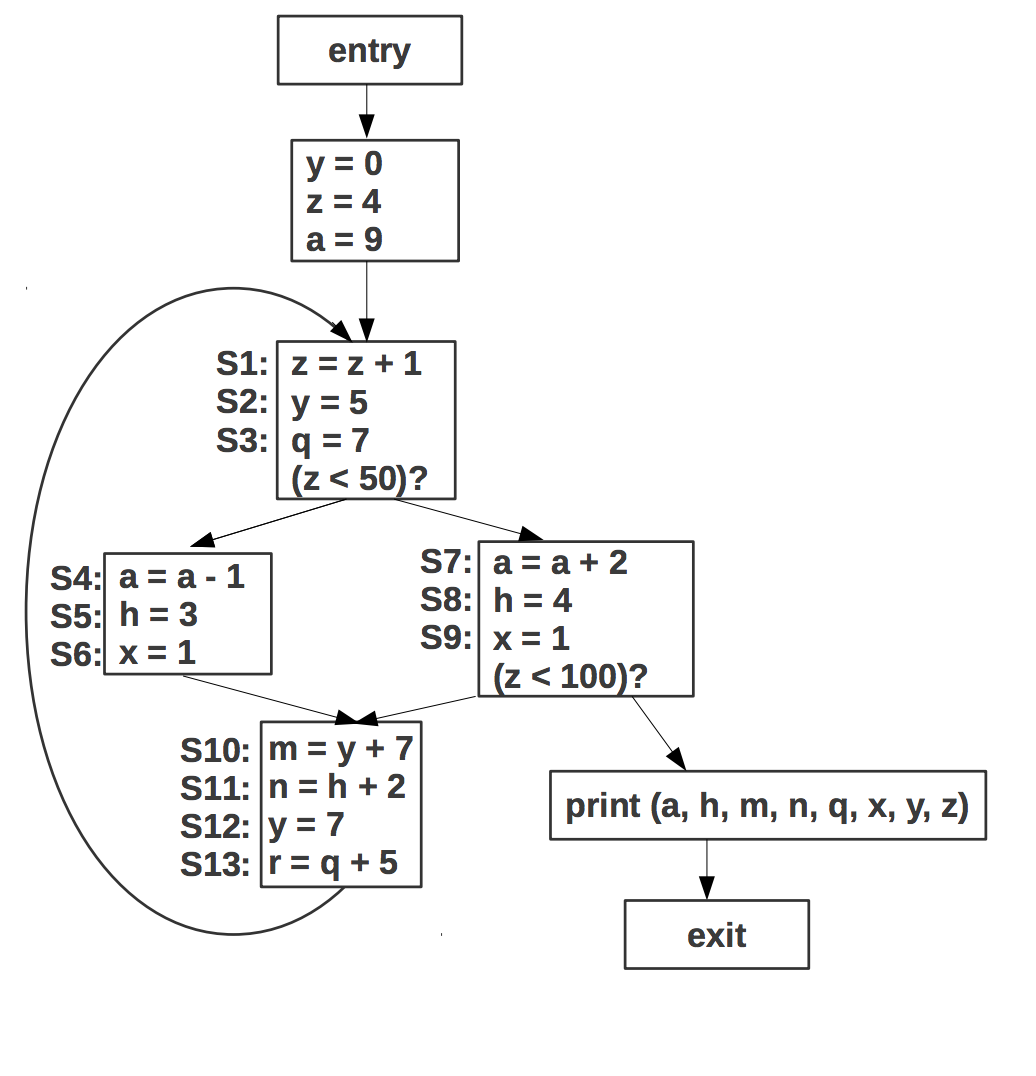
1

1

0

(1) x=x+3

3.2 LICM: Loop Invariant Code Motion



Loop Invariant Code Motion

Reaching Definitions:

Block 1:

IN={s1, s3, s4, s5, s6, s7, s8, s9, s10, s11, s12, s13, d1, d2, d3}

OUT={s1, s2, s3, s4, s5, s6, s7, s8, s9, s10, s11,, s13, d3}

Block2 (right):

IN={s1, s2, s3, s4, s5, s6, s7, s8, s9, s10, s11, s13, d3}

OUT={s1, s2, s3, s4, s5, s6, s10, s11, s13}

Block3 (Left):

IN={s1, s2, s3, s4, s5, s6, s7, s8, s9, s10, s11, s13, d3}

OUT={s1, s2, s3, s7, s8, s9, s10, s11, s13}

Block4:

IN={s1, s2, s3, s4, s5, s6, s7, s8, s9, s10, s11, s13}

OUT={s1, s3, s4, s5, s6, s7, s8, s9, s10, s11, s12, s13}

Loop Invariant Instructions:

S2, S3, S10, S12, S13

S2 is a constant definition of y.

S3 is a constant definition of q.

S10 is loop invariant since reaching definition of y is invariant.

S12 is a constant definition but it can be removed since it is killed by a redefinition of y at s2 and is not used anywhere else.

S13 is loop invariant since reaching definition of q is invariant.

Of the 6 only s2, s3, s12 and s13 can be moved to the pre-header by the loop invariant code motion pass.

S10 cannot be moved since there is a path to exit where the previous value of m is reached and is used.

After Lazy Code Motion Pass

y=0

z=4

a=9

Pre-header Block

S2: y=5

S12: q=7

S13: r=q+5

Entry

S1: z=z+1

(z<50)?

S4: a=a-1

S5: h=3

S6: x=1

S7: a=a+2

S8: h=4

S9: x=1

(z<100)?

Exit

S10: m=y+7

S11: n=h+2

Print (a, h, m, n, q, x, y, z)