40-414 Compiler Design

Local Optimizations

Lecture 11

Exercise

Question?

Which of the following are valid local optimizations for the given basic block? Assume that only g and x are referenced outside of this basic block.

```
1   a := 1
2   b := 3
3   c := a + x
4   d := a * 3
5   e := b * 3
6   f := a + b
7   g := e - f
```

- [ ] Copy propagation: Line 4 becomes d := a * b.
- [ ] Common subexpression elimination: Line 5 becomes e := d.
- [ ] Dead code elimination: Line 3 is removed.
- [ ] After many rounds of valid optimizations, the entire block can be reduced to g := 5.

Question?

Which of the following are valid local optimizations for the given basic block? Assume that only g and x are referenced outside of this basic block.

Copy propagation → (lines 1, 2)

```
1   a := 1
2   b := 3
3   c := a + x
4   d := a * 3
5   e := b * 3
6   f := a + b
7   g := e - f
```

- ☐ Copy propagation: Line 4 becomes d := a * b.
- ☐ Common subexpression elimination: Line 5 becomes e := d.
- ☐ Dead code elimination: Line 3 is removed.
- ☐ After many rounds of valid optimizations, the entire block can be reduced to g := 5.

Question?

Which of the following are valid local optimizations for the given basic block? Assume that only g and x are referenced outside of this basic block.

```
1   a := 1
2   b := 3
3   c := 1 + x
4   d := 1 * 3
5   e := 3 * 3
6   f := 1 + 3
7   g := e - f
```

- [ ] Copy propagation: Line 4 becomes d := a * b.
- [ ] Common subexpression elimination: Line 5 becomes e := d.
- [ ] Dead code elimination: Line 3 is removed.
- [ ] After many rounds of valid optimizations, the entire block can be reduced to g := 5.

Question?

Which of the following are valid local optimizations for the given basic block? Assume that only g and x are referenced outside of this basic block.

```
1   a := 1
2   b := 3
3   c := 1 + x
4   d := 1 * 3
5   e := 3 * 3
6   f := 1 + 3
7   g := e - f
```

Constant folding (lines 4, 5, 6)

- ☐ Copy propagation: Line 4 becomes d := a * b.
- ☐ Common subexpression elimination: Line 5 becomes e := d.
- ☐ Dead code elimination: Line 3 is removed.
- ☐ After many rounds of valid optimizations, the entire block can be reduced to g := 5.

Question?

Which of the following are valid local optimizations for the given basic block? Assume that only g and x are referenced outside of this basic block.

```
1  a := 1
2  b := 3
3  c := 1 + x
4  d := 3
5  e := 9
6  f := 4
7  g := e - f
```

- [ ] Copy propagation: Line 4 becomes d := a * b.
- [ ] Common subexpression elimination: Line 5 becomes e := d.
- [ ] Dead code elimination: Line 3 is removed.
- [ ] After many rounds of valid optimizations, the entire block can be reduced to g := 5.

Question?

Which of the following are valid local optimizations for the given basic block? Assume that only g and x are referenced outside of this basic block.

```
1   a := 1
2   b := 3
3   c := 1 + x
4   d := 3
5   e := 9
6   f := 4
7   g := e - f
```

Copy propagation (→ lines 5, 6)

- [ ] Copy propagation: Line 4 becomes d := a * b.
- [ ] Common subexpression elimination: Line 5 becomes e := d.
- [ ] Dead code elimination: Line 3 is removed.
- [ ] After many rounds of valid optimizations, the entire block can be reduced to g := 5.

Question?

Which of the following are valid local optimizations for the given basic block? Assume that only g and x are referenced outside of this basic block.

```
1   a := 1
2   b := 3
3   c := 1 + x
4   d := 3
5   e := 9
6   f := 4
7   g := 9 - 4
```

- ☐ Copy propagation: Line 4 becomes d := a * b.
- ☐ Common subexpression elimination: Line 5 becomes e := d.
- ☐ Dead code elimination: Line 3 is removed.
- ☐ After many rounds of valid optimizations, the entire block can be reduced to g := 5.

Question?

Which of the following are valid local optimizations for the given basic block? Assume that only g and x are referenced outside of this basic block.

```
1   a := 1
2   b := 3
3   c := 1 + x
4   d := 3
5   e := 9
6   f := 4
7   g := 9 - 4
```

Constant folding → line 7

- ☐ Copy propagation: Line 4 becomes d := a * b.
- ☐ Common subexpression elimination: Line 5 becomes e := d.
- ☐ Dead code elimination: Line 3 is removed.
- ☐ After many rounds of valid optimizations, the entire block can be reduced to g := 5.

Question?

Which of the following are valid local optimizations for the given basic block? Assume that only g and x are referenced outside of this basic block.

```
1   a := 1
2   b := 3
3   c := 1 + x
4   d := 3
5   e := 9
6   f := 4
7   g := 5
```

- ☐ Copy propagation: Line 4 becomes d := a * b.
- ☐ Common subexpression elimination: Line 5 becomes e := d.
- ☐ Dead code elimination: Line 3 is removed.
- ☐ After many rounds of valid optimizations, the entire block can be reduced to g := 5.

Question?

Which of the following are valid local optimizations for the given basic block? Assume that only g and x are referenced outside of this basic block.

Dead code elimination

```
1   a := 1
2   b := 3
3   c := 1 + x
4   d := 3
5   e := 9
6   f := 4
7   g := 5
```

- ☐ Copy propagation: Line 4 becomes d := a * b.
- ☐ Common subexpression elimination: Line 5 becomes e := d.
- ☐ Dead code elimination: Line 3 is removed.
- ☐ After many rounds of valid optimizations, the entire block can be reduced to g := 5.

Question?

Which of the following are valid local optimizations for the given basic block? Assume that only g and x are referenced outside of this basic block.

```
1
2
3
4
5
6
7   g := 5
```

- [ ] Copy propagation: Line 4 becomes d := a * b.
- [ ] Common subexpression elimination: Line 5 becomes e := d.
- [ ] Dead code elimination: Line 3 is removed.
- [ ] After many rounds of valid optimizations, the entire block can be reduced to g := 5.

Answer!

Which of the following are valid local optimizations for the given basic block? Assume that only g and x are referenced outside of this basic block.

```
1   a := 1
2   b := 3
3   c := a + x
4   d := a * 3
5   e := b * 3
6   f := a + b
7   g := e - f
```

- ☐ Copy propagation: Line 4 becomes d := a * b.
- ☐ Common subexpression elimination: Line 5 becomes e := d.
- ☐ Dead code elimination: Line 3 is removed.
- ☐ After many rounds of valid optimizations, the entire block can be reduced to g := 5.

Example: C code

```
void quicksort(m, n)
    int m, n;
    {
        int i, j;
        if (n <= m ) return;
        /* fragment begins here */
        i = m-1; j = n; v = a[n];
        while(1)    {
            do i = i+1; while( a[i] < v );
            do j = j-1; while( a[j] > v );
            if( i >= j ) break;
            x = a[i]; a[i] = a[j]; a[j] = x;
        }
        x = a[i]; a[i] = a[n]; a[n]= x;
        /* fragment ends here */
        quicksort(m, j); quicksort(i+1, n);
    }
```

Augmented 3AC

An augmented 3 address code language to simplify the code...

Let a be an array of integers starting at byte address a_0

a[add] on the left-hand-side of an assignment is the address a_0+add

a[add] on the right-hand-side of an assignment is the value of the element of the array at address a_0+add

*Since integers are stored in 4 bytes the offset address of an element a[i] is 4*i*

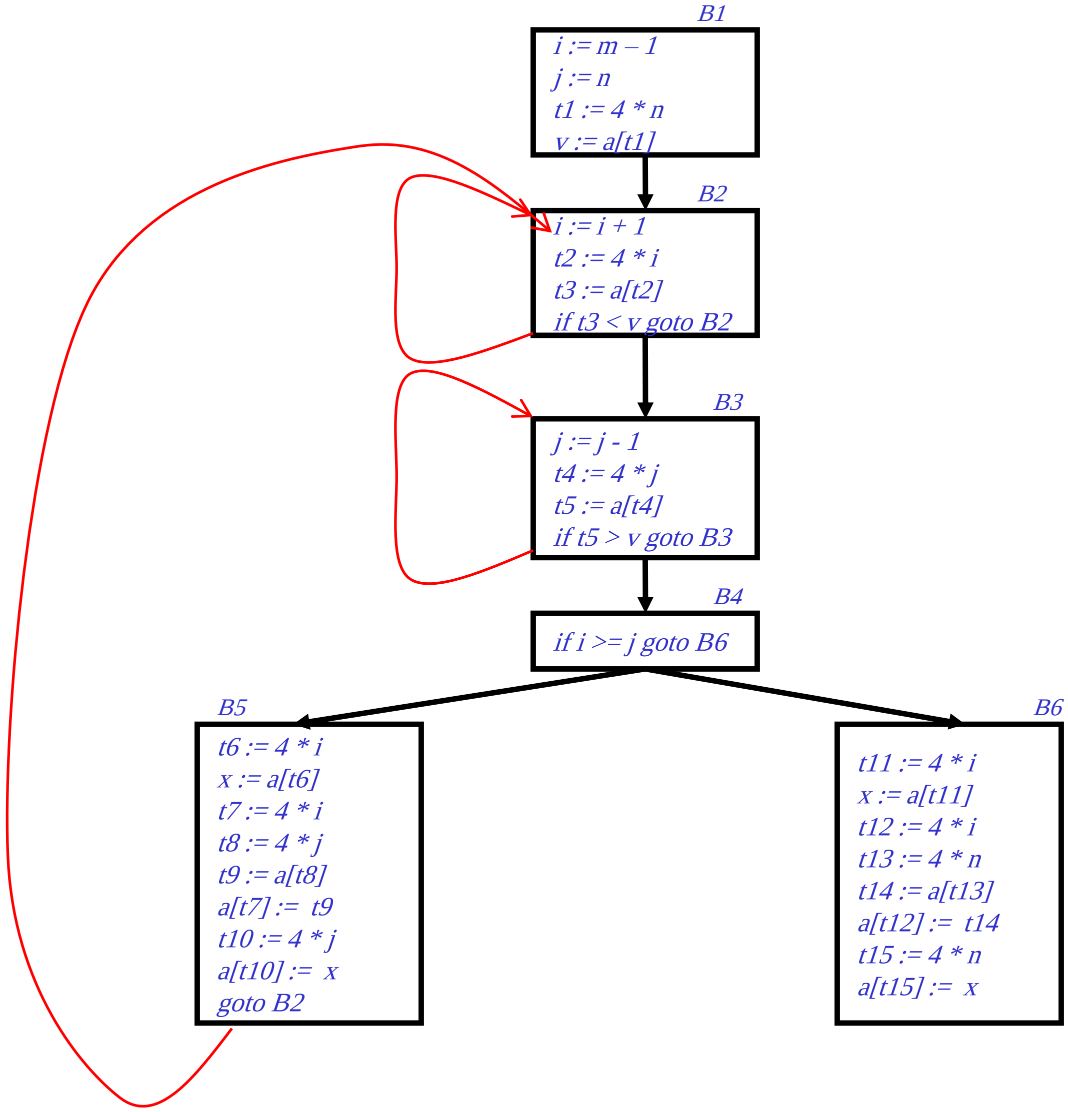
Augmented 3AC of the C code

```
01) i := m - 1
02) j := n
03) t1 := 4 * n
04) v := a[t1]
05) i := i + 1
06) t2 := 4 * i
07) t3 := a[t2]
08) if t3 < v goto 5
09) j := j – 1
10) t4 := 4 * j
11) t5 := a[t4]
12) if t5 > v goto 9
13) if i >= j goto 23
14) t6 := 4 * i
15) x := a[t6]
16) t7 := 4 * i
17) t8 := 4 * j
18) t9 := a[t8]
19) a[t7] := t9
20) t10 := 4 * j
21) a[t10] := x
22) goto 5
23) t11 := 4 * i
24) x := a[t11]
25) t12 := 4 * i
26) t13 := 4 * n
27) t14 := a[t13]
28) a[t12] := t14
29) t15 := 4 * n
30) a[t15] := x
```

Basic Blocks

```
01) i := m - 1
02) j := n
03) t1 := 4 * n
04) v := a[t1]
05) i := i + 1
06) t2 := 4 * i
07) t3 := a[t2]
08) if t3 < v goto 5
09) j := j - 1
10) t4 := 4 * j
11) t5 := a[t4]
12) if t5 > v goto 9
13) if i >= j goto 23
14) t6 := 4 * i
15) x := a[t6]
16) t7 := 4 * i
17) t8 := 4 * j
18) t9 := a[t8]
19) a[t7] := t9
20) t10 := 4 * j
21) a[t10] := x
22) goto 5
23) t11 := 4 * i
24) x := a[t11]
25) t12 := 4 * i
26) t13 := 4 * n
27) t14 := a[t13]
28) a[t12] := t14
29) t15 := 4 * n
30) a[t15] := x
```

Control Flow Graph



Local Optimizations

B5 before

```
t6 := 4 * i
x := a[t6]
t7 := 4 * i
t8 := 4 * j
t9 := a[t8]
a[t7] := t9
t10 := 4 * j
a[t10] := x
goto B2
```

B5 after

```
t6 := 4 * i
x := a[t6]
t7 := t6
t8 := 4 * j
t9 := a[t8]
a[t7] := t9
t10 := t8
a[t10] := x
goto B2
```

Common Subexpression Elimination

Local Optimizations

B5 before

```
t6 := 4 * i
x := a[t6]
t7 := t6
t8 := 4 * j
t9 := a[t8]
a[t7] := t9
t10 := t8
a[t10] := x
goto B2
```

B5 after

```
t6 := 4 * i
x := a[t6]
t7 := t6
t8 := 4 * j
t9 := a[t8]
a[t6] := t9
t10 := t8
a[t8] := x
goto B2
```

Copy propagation

Local Optimizations

B5 before

```
t6 := 4 * i
x := a[t6]
t7 := t6
t8 := 4 * j
t9 := a[t8]
a[t7] := t9
t10 := t8
a[t10] := x
goto B2
```

B5 after

```
t6 := 4 * i
x := a[t6]
t8 := 4 * j
t9 := a[t8]
a[t6] := t9
a[t8] := x
goto B2
```

Dead code elimination

Local Optimizations

B6 before

```
t11 := 4 * i
x := a[t11]
t12 := 4 * i
t13 := 4 * n
t14 := a[t13]
a[t12] := t14
t15 := 4 * n
a[t15] := x
```

B6 after

```
t11 := 4 * i
x := a[t11]
t12 := t11
t13 := 4 * n
t14 := a[t13]
a[t12] := t14
t15 := t13
a[t15] := x
```

Common Subexpression Elimination

Local Optimizations

B6 before	B6 after
t11 := 4 * i	t11 := 4 * i
x := a[t11]	x := a[t11]
t12 := t11	t12 := t11
t13 := 4 * n	t13 := 4 * n
t14 := a[t13]	t14 := a[t13]
a[t12] := t14	a[t11] := t14
t15 := t13	t15 := t13
a[t15] := x	a[t13] := x

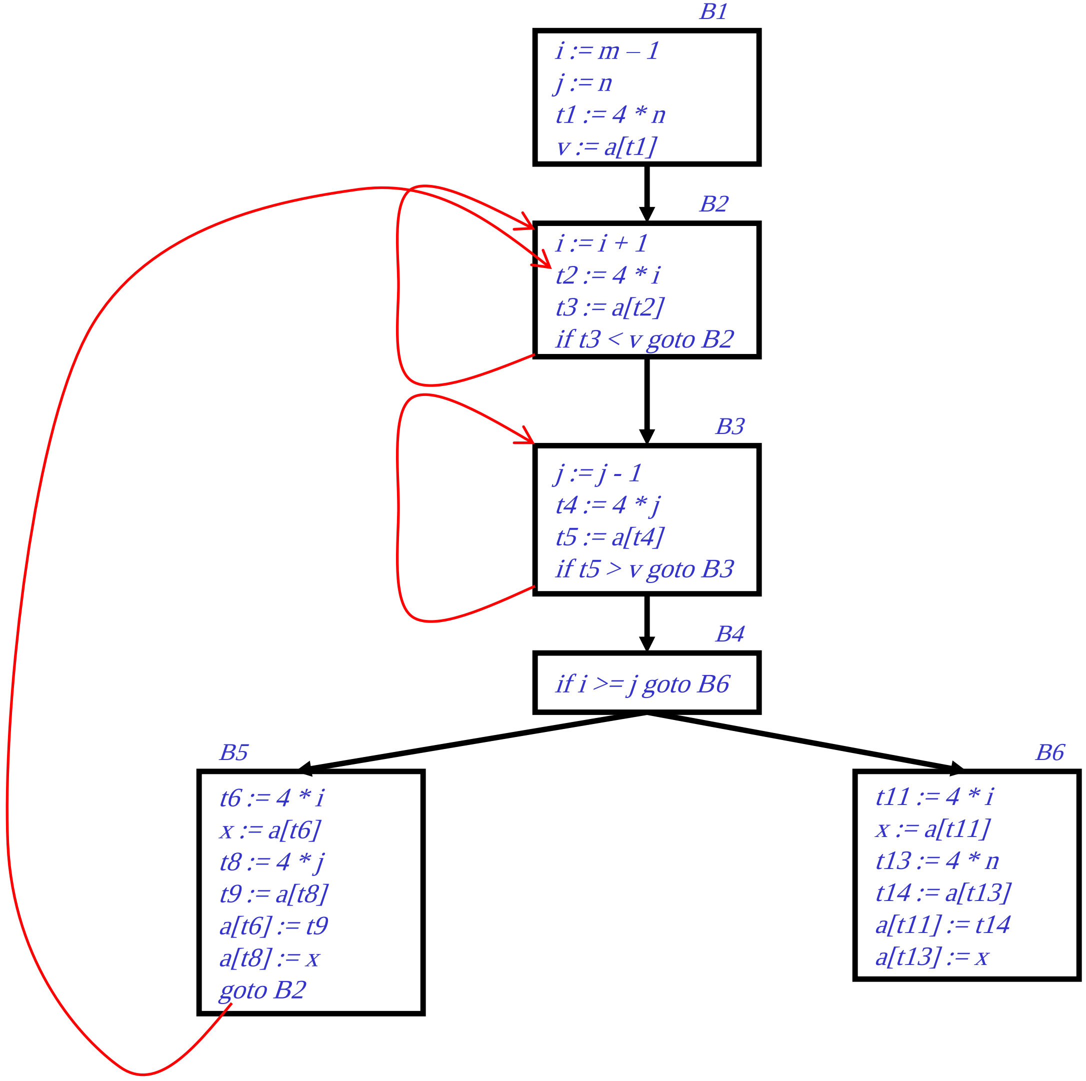
Copy Propagation

Local Optimizations

B6 before	B6 after
t11 := 4 * i	t11 := 4 * i
x := a[t11]	x := a[t11]
t12 := t11	t13 := 4 * n
t13 := 4 * n	t14 := a[t13]
t14 := a[t13]	a[t11] := t14
a[t11] := t14	a[t13] := x
t15 := t13	
a[t13] := x	

Dead code elimination

After Local Optimizations



Reduction in Strength

In B2 whenever i increases by 1, t2 increases by 4
In B3 whenever j decreases by 1, t4 decreases by 4

B1 Before

```
i := m - 1
j := n
t1 := 4 * n
v := a[t1]

B2:
  i := i + 1
  t2 := 4 * i
  t3 := a[t2]
  if t3 < v goto B2
B3:
  j := j - 1
  t4 := 4 * j
  t5 := a[t4]
  if t5 > v goto B2
```

B1 After

```
i := m - 1
j := n
t1 := 4 * n
v := a[t1]
t2 := 4 * i
t4 := 4 * j
B2:
  i := i + 1
  t2 := t2 + 4
  t3 := a[t2]
  if t3 < v goto B2
B3:
  j := j - 1
  t4 := t4 - 4
  t5 := a[t4]
  if t5 > v goto B3
```

Induction Variables Elimination

In B2 whenever i increases by 1, t2 increases by 4, i and t2 are called induction variables.

In B3 whenever j decreases by 1, t4 decreases by 4, j and t4 are induction variables, too.

If there are two or more induction variables in a loop, it may be possible to get rid of all but one

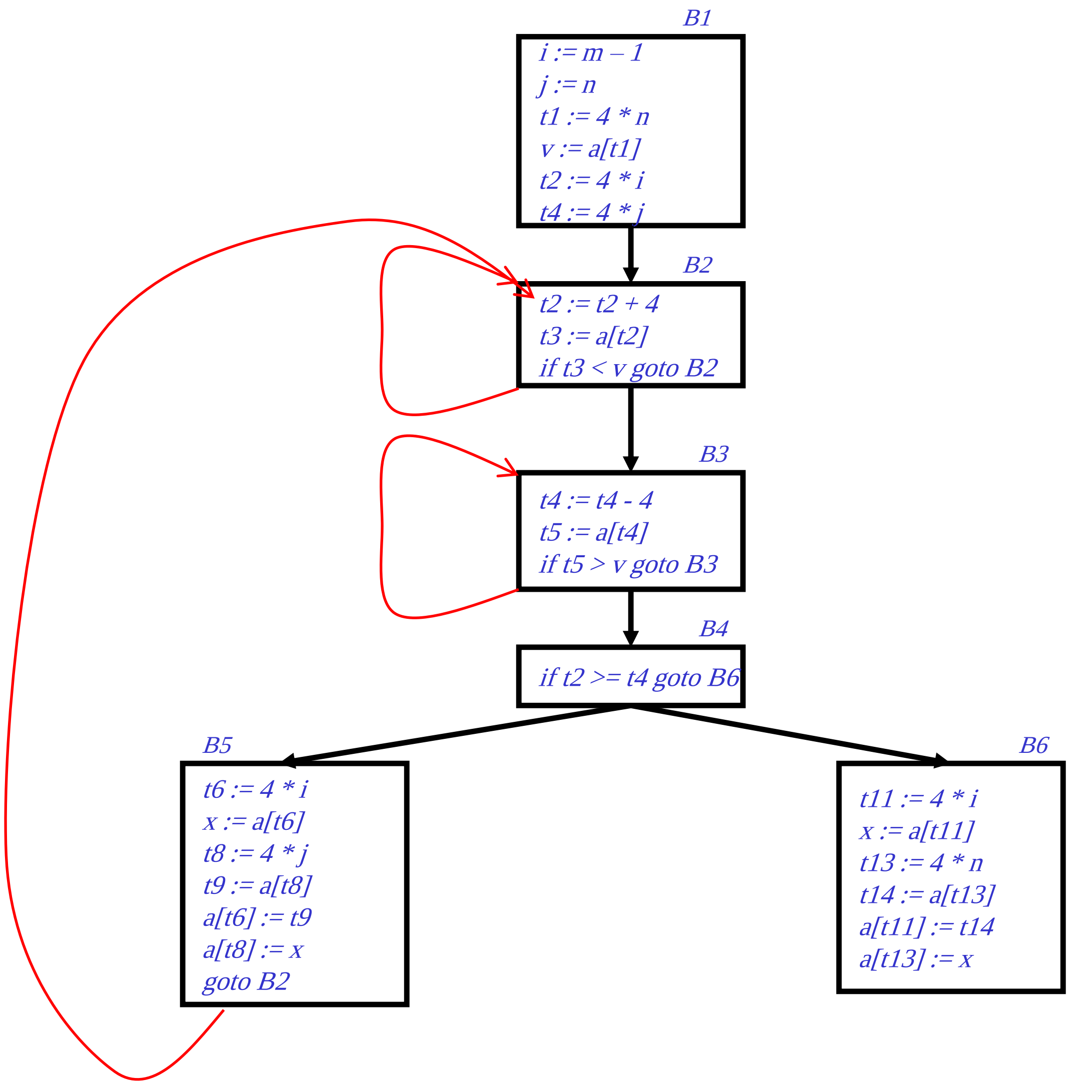
B4: Before

if i >= j goto B6

B4: After

if t2 >= t4 goto B6

After Loop Optimizations



After Global Optimizations