Classical Optimizations

Overview

- Last lecture
  - → Code generation
    - High-level languages
  - → Optimization
    - Goals, approaches
- This lecture
  - → Classical optimizations
    - Common subexpression elimination
    - Forward substitution / copy propagation
    - Loop invariant code motion
    - Strength reduction
    - Loop test elimination, induction variable elimination
    - Constant propagation
    - Dead code elimination
  - → Basic block construction
  - → Basic block DAG construction

Example Code

Fortran Source Code:

```
        .
        .
        .
        sum = 0
        do 10 i = 1, n
10      sum = sum + a(i) * a(i)
        .
        .
        .
```

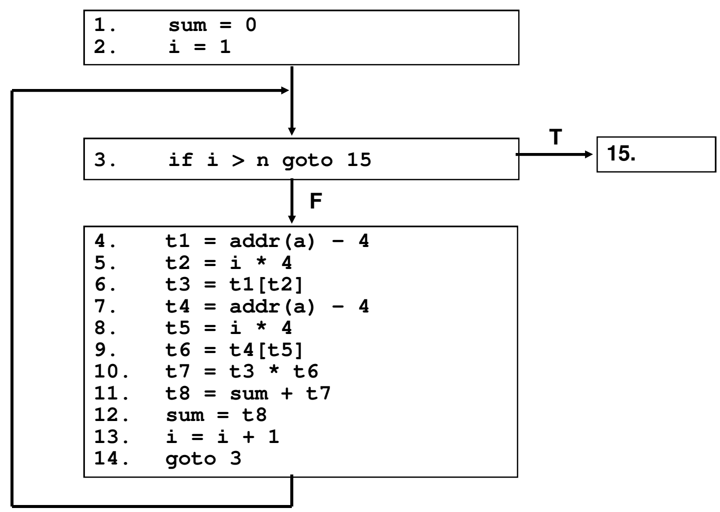
CS430 3

3-address code

1.	`sum = 0`	**sum = 0**
2.	`i = 1`	**init for loop and check limit**
3.	`if i > n goto 15`	
4.	`t1 = addr(a) - 4`	**a[i]**
5.	`t2 = i * 4`	
6.	`t3 = t1[t2]`	
7.	`t4 = addr(a) - 4`	**a[i]**
8.	`t5 = i * 4`	
9.	`t6 = t4[t5]`	
10.	`t7 = t3 * t6`	**a[i] * a[i]**
11.	`t8 = sum + t7`	**increment sum**
12.	`sum = t8`	
13.	`i = i + 1`	**Incr. loop counter back to loop check**
14.	`goto 3`	
15.		

CS430 4

Control Flow Graph (CFG)



Common Subexpression Elimination

```
1.      sum = 0
2.      i = 1
3.      if i > n goto 15
4.      t1 = addr(a) – 4
5.      t2 = i * 4
6.      t3 = t1[t2]
7.      t4 = addr(a) – 4
8.      t5 = i * 4
9.      t6 = t4[t5]
10.     t7 = t3 * t6
10a.    t7 = t3 * t3
11.     t8 = sum + t7
12.     sum = t8
13.     i = i + 1
14.     goto 3
15.
```

common subexpression elimination

Forward Substitution (Copy Propagation)

```
1.      sum = 0
2.      i = 1
3.      if i > n goto 15
4.      t1 = addr(a) – 4
5.      t2 = i * 4
6.      t3 = t1[t2]
10a.    t7 = t3 * t3
11.     t8 = sum + t7
12.     sum = t8                 forward substitution
12a.    sum = sum + t7
13.     i = i + 1
14.     goto 3
15.
```


Invariant Code Motion

```
1.      sum = 0
2.      i = 1
2a.     t1 = addr(a) – 4
3.      if i > n goto 15
4.      t1 = addr(a) – 4         invariant code motion
5.      t2 = i * 4
6.      t3 = t1[t2]
10a.    t7 = t3 * t3
12a.    sum = sum + t7
13.     i = i + 1
14.     goto 3
15.
```

Strength Reduction

```
1.      sum = 0
2.      i = 1
2a.     t1 = addr(a) - 4
2b.     t2 = i * 4
3.      if i > n goto 15
5.      t2 = i * 4                  strength reduction
6.      t3 = t1[t2]
10a.    t7 = t3 * t3
12a.    sum = sum + t7
12b.    t2 = t2 + 4
13.     i = i + 1
14.     goto 3
15.
```


Loop Test Adjustment

```
1.      sum = 0
2.      i = 1
2a.     t1 = addr(a) - 4
2b.     t2 = i * 4
2c.     t9 = n * 4
3.      if i > n goto 15            loop test adjustment
3a.     if t2 > t9 goto 15
6.      t3 = t1[t2]
10a.    t7 = t3 * t3
12a.    sum = sum + t7
12b.    t2 = t2 + 4
13.     i = i + 1
14.     goto 3a
15.
```

Induction Variable Elimination

```
1.      sum = 0
2.      i = 1
2a.     t1 = addr(a) - 4
2b.     t2 = i * 4
2c.     t9 = n * 4
3a.     if t2 > t9 goto 15
6.      t3 = t1[t2]
10a.    t7 = t3 * t3
12a.    sum = sum + t7
12b.    t2 = t2 + 4
13.     i = i + 1                    induction variable elimination
14.     goto 3a
15.
```


Constant Propagation

```
1.      sum = 0
2.      i = 1
2a.     t1 = addr(a) - 4
2b.     t2 = i * 4                   constant propagation
2d.     t2 = 4
2c.     t9 = n * 4
3a.     if t2 > t9 goto 15
6.      t3 = t1[t2]
10a.    t7 = t3 * t3
12a.    sum = sum + t7
12b.    t2 = t2 + 4
14.     goto 3a
15.
```

Dead Code Elimination

```
1.      sum = 0
2.      i = 1
2a.     t1 = addr(a) - 4
2d.     t2 = 4
2c.     t9 = n * 4
3a.     if t2 > t9 goto 15
6.      t3 = t1[t2]
10a.    t7 = t3 * t3
12a.    sum = sum + t7
12b.    t2 = t2 + 4
14.     goto 3a
15.
```

dead code elimination

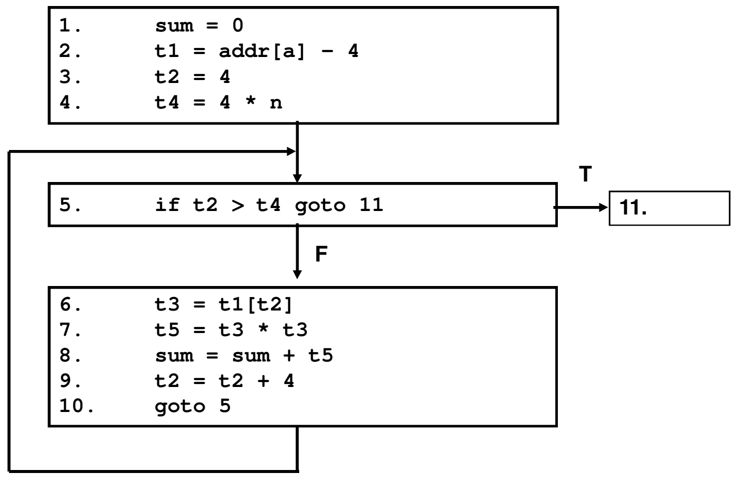
Final Optimized Code

```
1.      sum = 0
2.      t1 = addr(a) - 4
3.      t2 = 4
4.      t4 = n * 4
5.      if t2 > t4 goto 11
6.      t3 = t1[t2]
7.      t5 = t3 * t3
8.      sum = sum + t5
9.      t2 = t2 + 4
10.     goto 5
11.
```

unoptimized: 8 temps, 11 stmts in innermost loop
optimized: 5 temps, 5 stmts in innermost loop

1 index addressing	2 index addressing
1 multiplication	3 multiplications
2 additions	2 additions & 2 subtractions
1 jump	1 jump
1 test	1 test
	1 copy

CFG of Final Optimized Code



Basic Block Construction

- Find leader statements
  - First program statement
  - Targets of conditional or unconditional gotos
  - Any statement following a conditional goto
- ($\forall x \in$ leaders) construct Bx, basic block headed by x: include all statements following x until next leader or end of program is reached.
- At end of algorithm, any statements not in some basic block are unreachable from program entry and are therefore, <u>dead code.</u>

3-address code example

leader

```
1.   sum = 0
2.   i = 1
3.   if i > n goto 15
4.   t1 = addr(a) – 4
5.   t2 = i * 4
6.   t3 = t1[t2]
7.   t4 = addr(a) – 4
8.   t5 = i * 4
9.   t6 = t4[t5]
10.  t7 = t3 * t6
11.  t8 = sum + t7
12.  sum = t8
13.  i = i + 1
14.  goto 3
15.
```


3-address code example

leader

```
1.   sum = 0
2.   i = 1
3.   if i > n goto 15
4.   t1 = addr(a) – 4
5.   t2 = i * 4
6.   t3 = t1[t2]
7.   t4 = addr(a) – 4
8.   t5 = i * 4
9.   t6 = t4[t5]
10.  t7 = t3 * t6
11.  t8 = sum + t7
12.  sum = t8
13.  i = i + 1
14.  goto 3
15.
```

3-address code example

leader

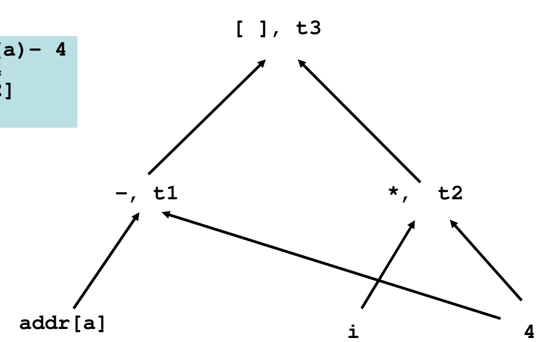
basic block

```
B1    1.   sum = 0
      2.   i = 1
B3    3.   if i > n goto 15
B4    4.   t1 = addr(a) – 4
      5.   t2 = i * 4
      6.   t3 = t1[t2]
      7.   t4 = addr(a) – 4
      8.   t5 = i * 4
      9.   t6 = t4[t5]
      10.  t7 = t3 * t6
      11.  t8 = sum + t7
      12.  sum = t8
      13.  i = i + 1
      14.  goto 3
B15   15.
```


Basic Block DAG Construction

B4

```
4.      t1 = addr(a)– 4
5.      t2 = i * 4
6.      t3 = t1[t2]
    . . .
```

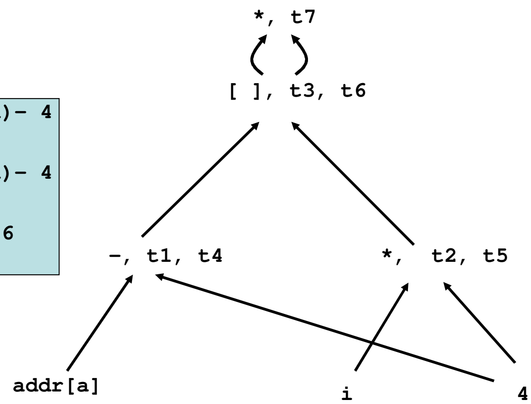


Reference: ASU, p.548

Basic Block DAG Construction

B4

```
4.      t1 = addr(a)- 4
5.      t2 = i * 4
6.      t3 = t1[t2]
7.      t4 = addr(a)- 4
8.      t5 = i * 4
9.      t6 = t4[t5]
10.     t7 = t3 * t6
    . . .
```



Reference: ASU, p.548

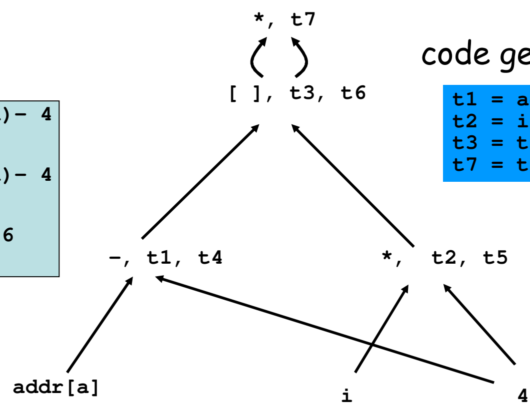
CS430

21

Basic Block DAG Construction

B4

```
4.      t1 = addr(a)- 4
5.      t2 = i * 4
6.      t3 = t1[t2]
7.      t4 = addr(a)- 4
8.      t5 = i * 4
9.      t6 = t4[t5]
10.     t7 = t3 * t6
    . . .
```



code generated:

```
t1 = addr[a]-4
t2 = i * 4
t3 = t1[t2]
t7 = t3 * t3
```

Reference: ASU, p.548

CS430

22

Basic Block DAG Construction

How to add a subexpression into a partially constructed DAG:

A = B + C

Is there a node already for B + C?

- If so, add A to its list of labels.
- If not:
  - is there a node labeled B already?

    If not, create a leaf labeled B.
  - Is there a node labeled C already?

    If not, create a leaf labeled C.
  - Create a node labeled A, for +, with left child B and right child C.

How to do this? HASHING <op, node(opd1), node(opd2)>

Reference: ASU, p.548

Basic Block DAG Construction

How to add a subexpression into a partially constructed DAG:

A = B + C

Is there a node already for B + C? <+,node(B),node(C)> defined?

- If so, add A to its list of labels.
- If not:
  - is there a node labeled B already? node(B) defined?

    If not, create a leaf labeled B.
  - Is there a node labeled C already? node(C) defined?

    If not, create a leaf labeled C.
  - Create a node labeled A, for +, with left child B and right child C. Create node(+) with children node(B), node(C)

How to do this? HASHING <op, node(opd1), node(opd2)>

Reference: ASU, p.548