Code Optimization

1 COPY PROPAGATION

It is code transformation technique used to improve the intermediate code. In copy propagation, if statement **x=y** exists in code, then use "y" instead of "x" in statements following x=y in the TAC.

Consider following program segment:

```
int a[20]
int main()
{
   a[0] = 3;
   a[1] = 4;
}
```

The Three Address Code for the above program segment will be: Assume bpw = 4

```
10:     t1 = 0 * 4// value of "i" represents reference, which is 0 in statement a[0]=3
20:     t2 =  add(a) – C //C = bpw
30:     t2[t1] = 3
40:     t3 = 1 * 4 // value of "I" represents reference, which is 1 in statement a[1]=3
50:     t4 = add(a) – C // C =bpw
60:     t4[t3] = 4
```

In the above code, copy propagation is possible for statement 10 and 40. Value of t1=0 and value of t3=4. The three address code after replacing copy propagation, will be:

```
10:     t1 = 0
20:     t2 =  add(a) – C //C = bpw
30:     t2[0] = 3
40:     t3 = 4
50:     t4 = add(a) – C // C =bpw
60:     t4[4] = 4
```

1.1 Dead Store elimination

In the above code statement 10 and 40, represents dead store. The value of dead store memory locations are not used in the three address code and can be used for optimization.

The optimized code is represented as:

```
20:     t2 =  add(a) – C //C = bpw
30:     t2[0] = 3
50:     t4 = add(a) – C // C =bpw
60:     t4[4] = 4
```

2 CONSTANT PROPAGATION

In the above example, the use of constant "0" was propagated in place of t0 and constant "4" was propagated in place of t3. This kind of propagation is referred as constant propagation.

It is also called as "constant folding optimization".

2.1 Variable Propagation: Consider following program segment

```
int demo(int a, int b, int c)
{
    int d,e,f;
    d = a;
    if(a>10)
      e=d+b;
   else
     e=d+c;
 f =d * e;
}
```

The Three Address code for above program segment will be:

```
10:     d = a
20      if (a>10) goto 40
30      goto 60
40      e = d+b
50      goto 70
60      e = d+c
70      f = d*e
```

The TAC will be modified as after performing of copy propagation:

```
10:     d = a
20      if (a>10) goto 40
30      goto 60
40      e = a+b  //variable propagation
50      goto 70
60      e = a+c  //variable propagation
70      f = a*e  //variable propagation
```

The statement number 10 represents "dead store", which can be also eliminated to perform further optimization.

```
20      if (a>10) goto 40
30      goto 60
40      e = a+b  //variable propagation
50      goto 70
60      e = a+c  //variable propagation
70      f = a*e  //variable propagation
```

3 DEAD CODE ELIMINATION

The part of program segment which is never reached during execution is called as "dead code". This code can be eliminated to perform optimization.
Consider the following program segment

```
int debug;
int demo(int a, int b, int c)
{
   int x,y,z;
   debug=0
   if(debug=1)
      z=a+b+c;
  v=a+b+c;
}
```

The Three Address code for above program segment will be:

```
10:    debug=0
20:    if(debug=1) goto 30
30:    goto 70
40:    t1=a+b
50:    t2=t1+c
60:    z=t1
70:    t3=a+b
80:    t4=t3+c
90:    v=t4
```

After copy propagation:

```
10:    debug=0
20:    if(0=1) goto 30
30:    goto 70
40:    t1=a+b
50:    t2=t1+c
60:    z=t1
70:    t3=a+b
80:    t4=t3+c
90:    v=t4
```

Dead Code elimination: Since "debug=0", the "if condition" will not be true any time. The code of statement "20,30,40,50" will be never executed and hence can be eliminated.
The three address code after dead code elimination:

```
10:    debug=0
70:    t3=a+b
80:    t4=t3+c
90:    v=t4
```

Example 2: Dead Code Elimination

```
int demo(int a, int b, int c)
{
  int x,y,z;
  x = a+b
  y = x/c
  return(y)
  x=y+z;
  z=z+1
  return(z)
}
```

In the above code segment the "return(y)" statement will generate transfer of control to main program. This will generate "dead code" in the segment.

The Three Address Code for above segment will be:

```
10:   t1=a+b
20:   x=t1
30:   t2=t1/c
40:   y=t2
50:   return(y)
60:   goto 120
70:   t1=y+z
80:   x=t1
90:   t2=z+1
100:  z=t2
110:  return(z)
120:  exit
```

The optimized TAC after elimination of dead code:

```
10:   t1=a+b
20:   x=t1
30:   t2=t1/c
40:   y=t2
50:   return(y)
```

4 ALGEBRAIC TRANSFORMATIONS

The quality of intermediate code can be improved by taking advantage of Algebraic transformations. Some of the commonly used identities are:

Name of identity	Example	Example
Additive identity	x+0=x	y=x+0 means y=x
Multiplicative identity	x*1=x	y=x*1 means y=x
Multiplication with zero	x*0=0	y=x*0 means y=0

This transformation reduces computation time. For example in place of addition instruction, assignment instruction will be used, which requires less time during execution. Sometimes it is also possible to replace arithmetic instructions with copy instruction.

4.1 Strength Reduction Transformation

This transformation replaces the costly operations with less expensive operations.
Example:
y = x * 2 can be replaced with y = x+x
y = x * 32 can be replaced with y = x << 5
y = x / 8 can be replaced with y = x >> 3