CS & IT Engineering

Compiler Design

Intermediate Code & Code Optimization

Deva sir

Lecture: 2

Topics to be covered:

- → 3-Address code Notations
- → Control Flow Graph (CFG)
- → Practice on Intermediate code
- *** → Code optimization

Three Address code Notations:

- Triple Notation
- Quadruple Notation
- Indirect Triple Notation

Advantage: Less space

Disadvantage: Computation takes much time

$x = a + b$
$y = x * c$
$z = x + y$
$w = z$

Triple Notation

	operator	operand1	operand2
1000	+	a	b
1010	*	1000	c
1020	+	1000	1010
1030	=	1020	

1000: (+, a, b)
1010: (*, 1000, c)
1020: (+, 1000, 1010)
1030: (=, 1020)

Three Address code Notations:

- Triple Notation
- Quadruple Notation
- Indirect Triple Notation

Advantage: Takes less time to compute

Disadvantage: More space

$x = a + b$

$y = x * c$

$z = x + y$

$w = z$

Quadruple Notation:

	operator	operand 1	operand 2	Result
1000	+	a	b	x
1015	*	x	c	y
1030	+	x	y	z
1045	=	z		w

Three Address code Notations:

- Triple Notation
- Quadruple Notation
- Indirect Triple Notation

Indirect Triple Notation

$x = a + b$
$y = x * c$
$z = x + y$
$w = z$

	operator	operand1	operand2
1000	+	a	b
1010	*	6000	c
1020	+	6000	7000
1030	=	8000	

Indirect	Actual Address
6000	1000
7000	1010
8000	1020
9000	1030

3-address code

```
t1 = b*c
t2 = a+t1
t3 = b*c
t4 = d/t3
t5 = t2-t4
```

Quadruples

op	arg_1	arg_2	result
*	b	c	t1
+	a	t1	t2
*	b	c	t3
/	d	t3	t4
-	t2	t4	t5

Triples

	op	arg_1	arg_2
0	*	b	c
1	+	a	(0)
2	*	b	c
3	/	d	(2)
4	-	(1)	(3)

Three Address Code for if else statement

```
if (x < y)
    z = x;
else
    z = y;

z = z * z;
```

```
    _t0 = x < y;
    IfZ _t0 Goto _L0;
    z = x;
    Goto _L1;
_L0:
    z = y;
_L1:
    z = z * z;
```

Three Address Code for while statement

```
while (x < y) {
   x = x * 2;
}

y = x;
```

```
_L0:
     _t0 = x < y;
     IfZ _t0 Goto _L1;
     x = x * 2;
     Goto _L0;
_L1:
     y = x;
```

Three Address Code for Array

Assume Declaration: A[n1,n2]

3AC for A[i,j] is:

$t_1 = n2 * i$
$t_2 = t_1 + j$
$t_3 = t_2 * w$
$t_4 = \text{Base Address}$
$t_5 = t_4 [t_3]$

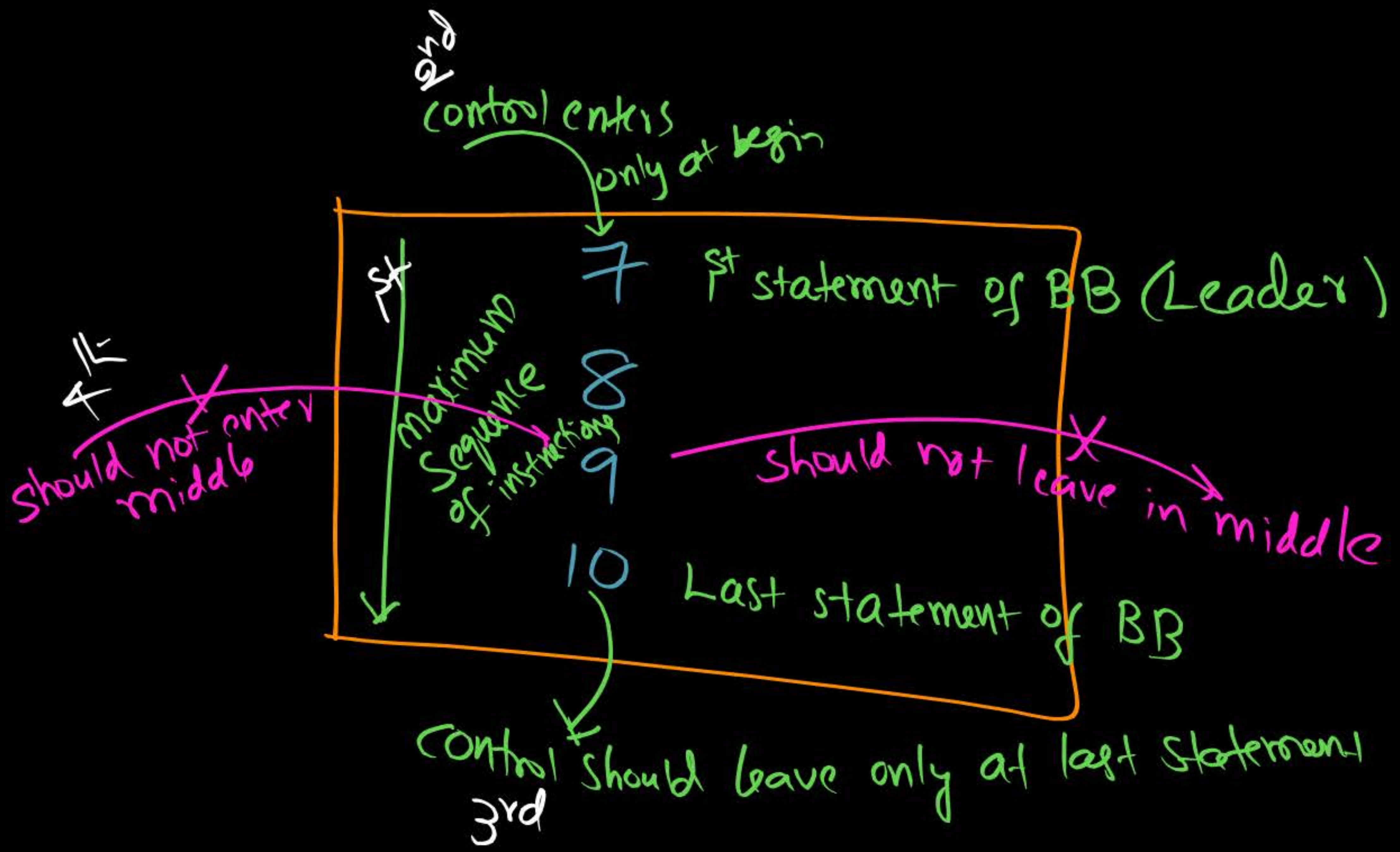
A[i, j] = Base Address + (n2*i + j) *w
Here, n2 is size of each row and w is size of each element.

*** Control Flow Graph

- It comprises of nodes and edges
- It is collection of $\boxed{\text{Basic Blocks}}$ and controls.
- It represents flow of program execution using Basic Blocks.

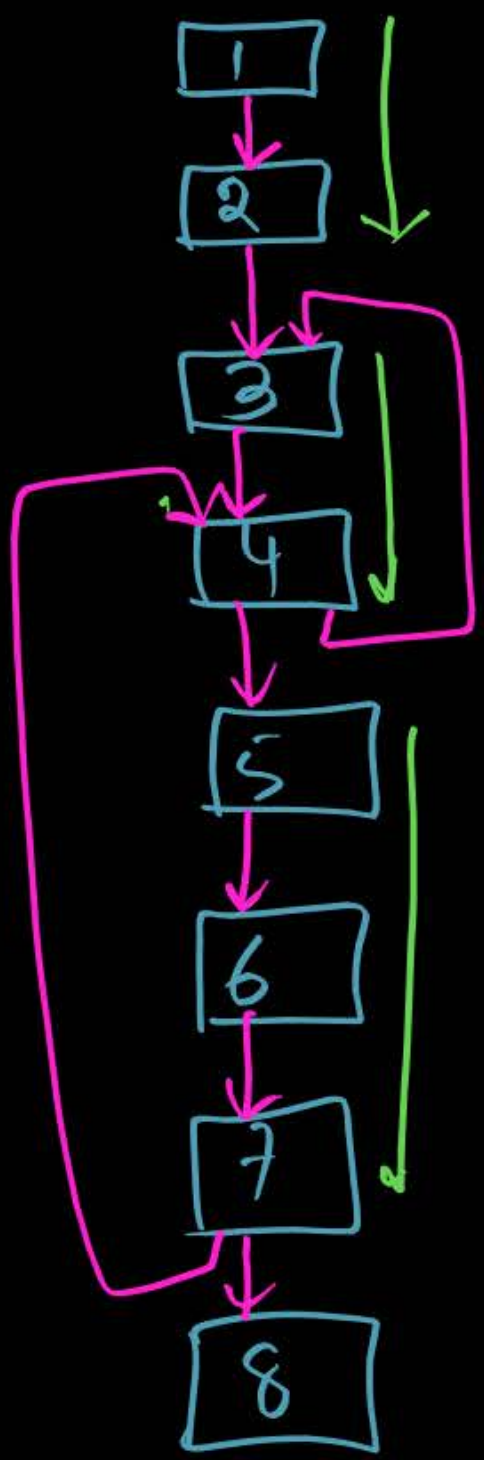
What is Basic Block (BB) ?

4 Imp points:

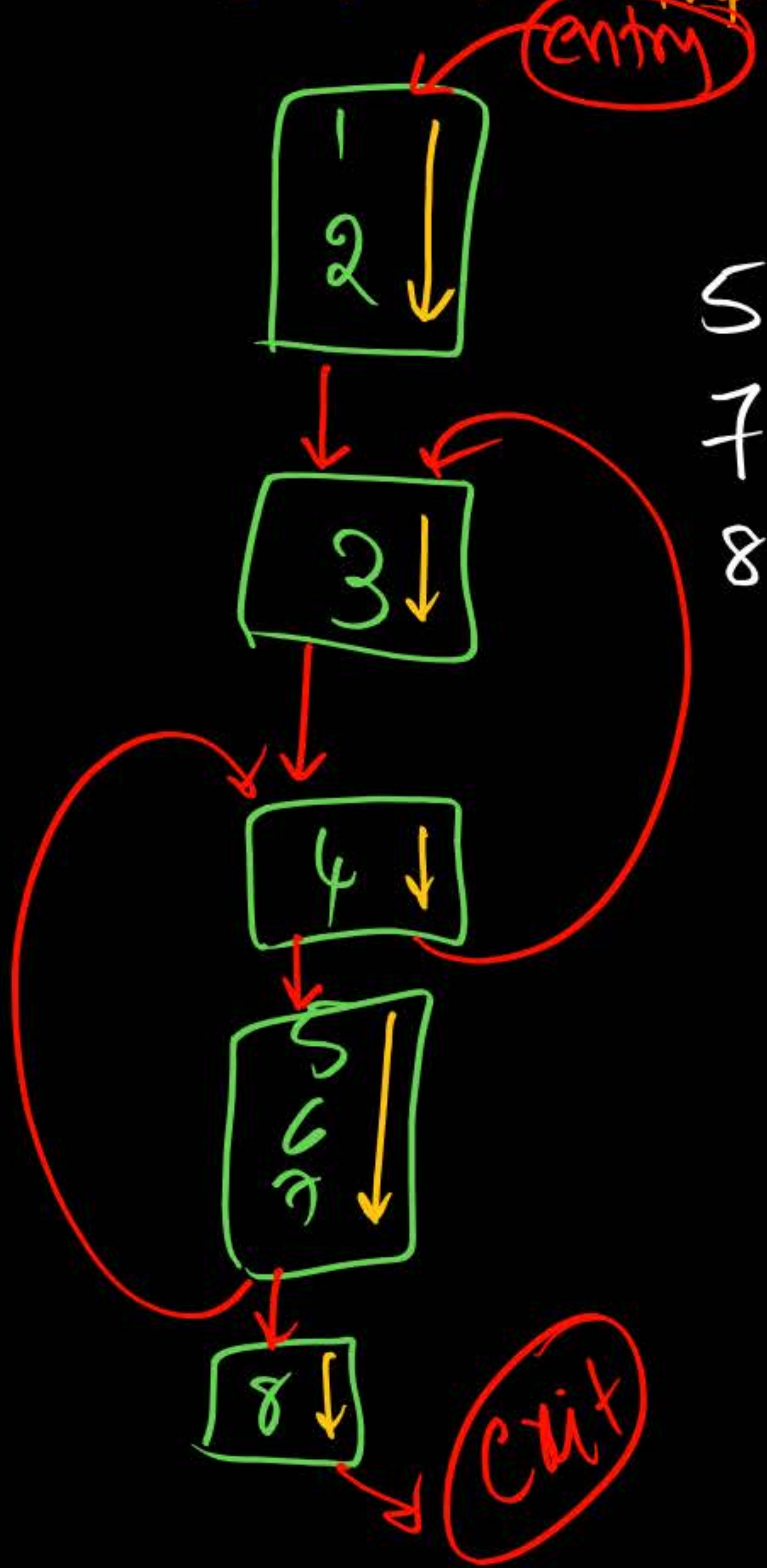


1. $x = a + b$
2. $y = x * y$
3. $z = x + y$
4. if $(z > d)$ goto 3
5. $z = x - y$
6. $z = z + a$
7. if $(z > 1000)$ goto 4
8. Print z

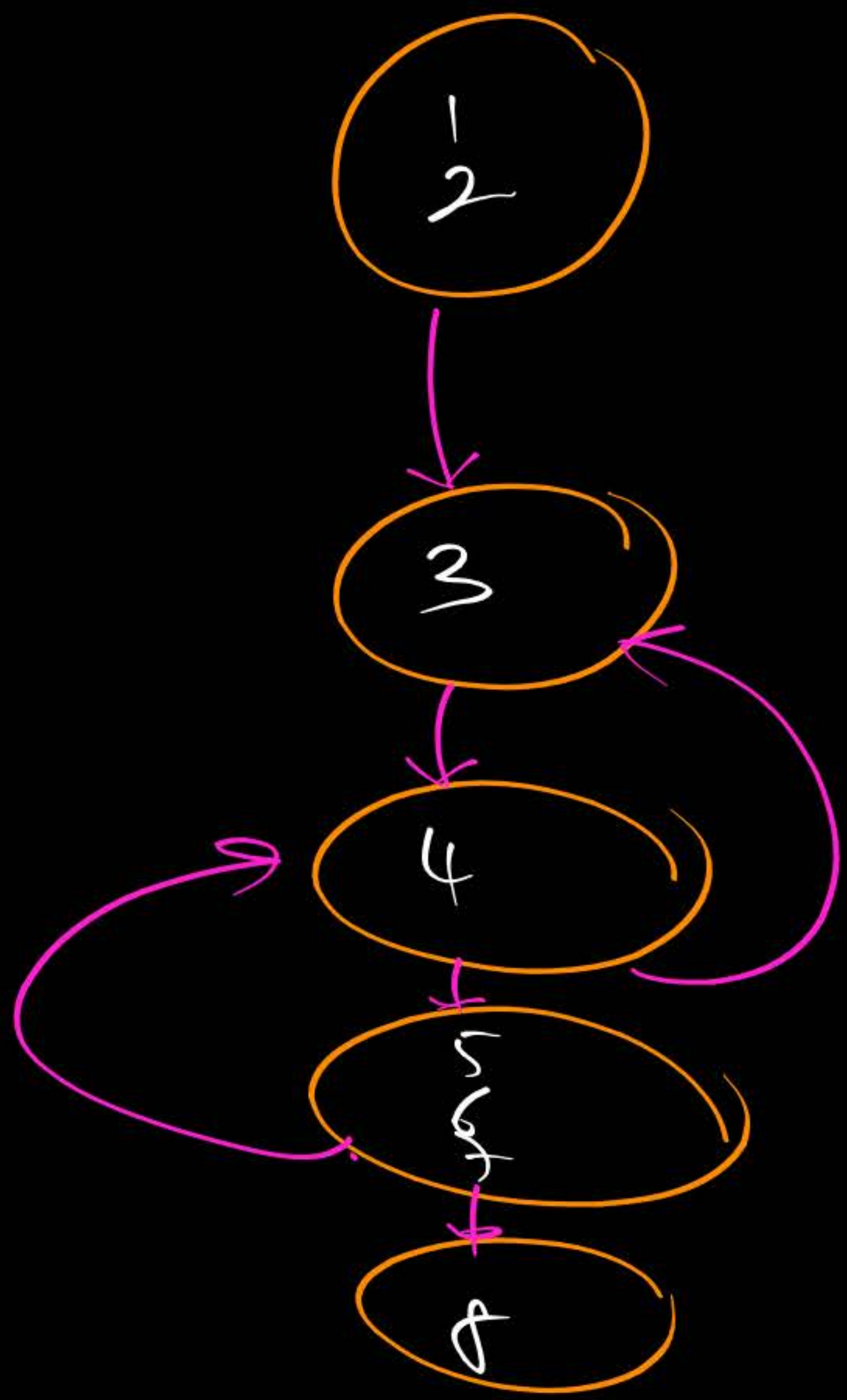
Flow



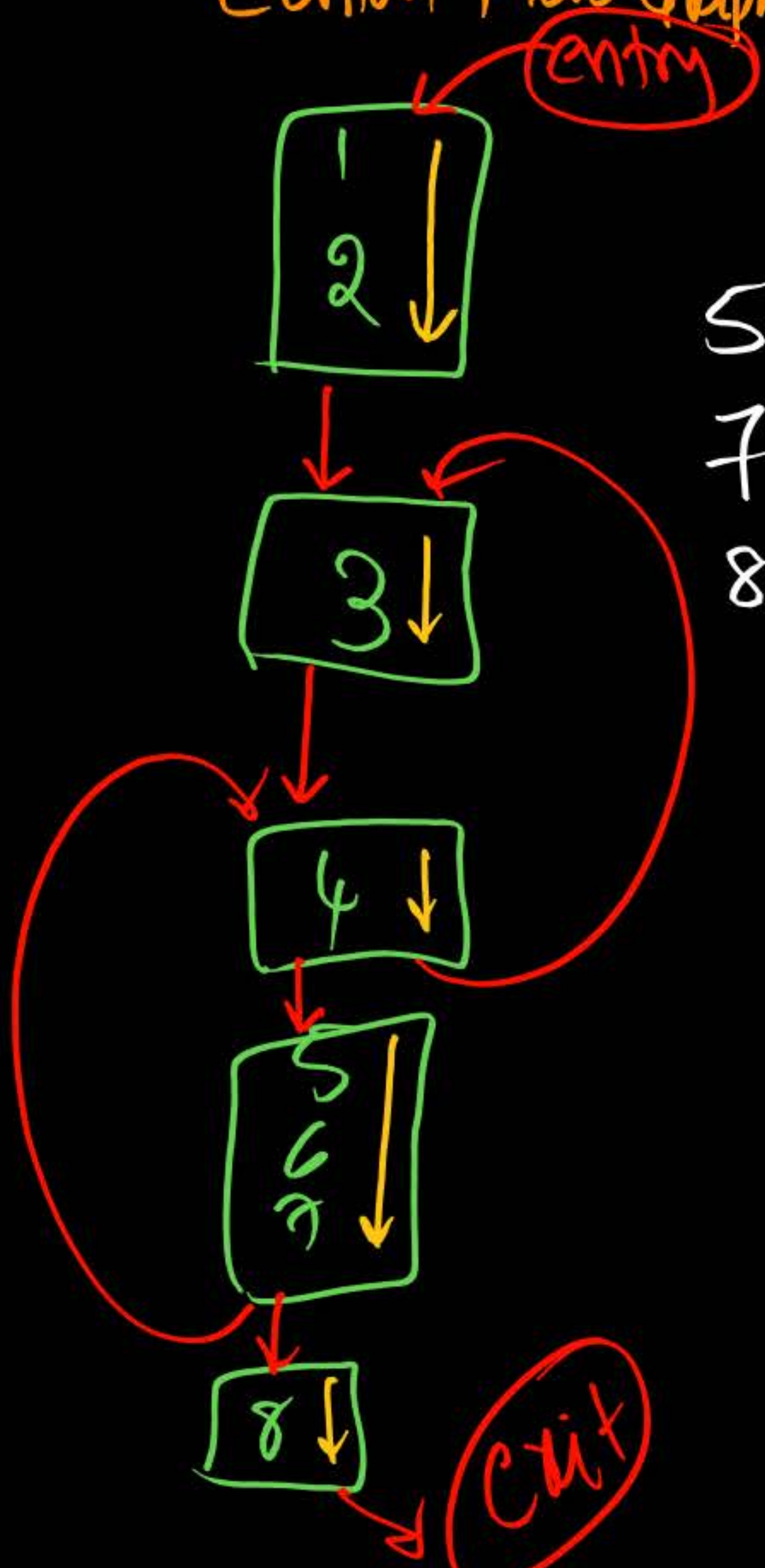
Control Flow Graph



5 BBs
7 nodes
8 edges



Control Flow Graph



5 BBs
7 nodes
8 edges

Control Flow Graph

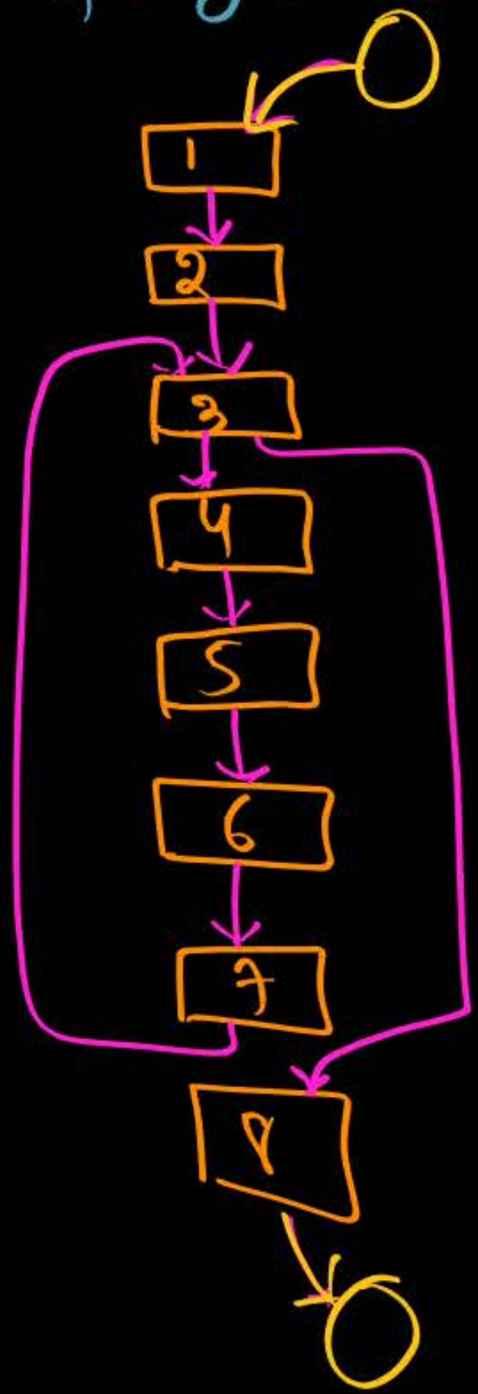
Q1) No. of BBs = 4

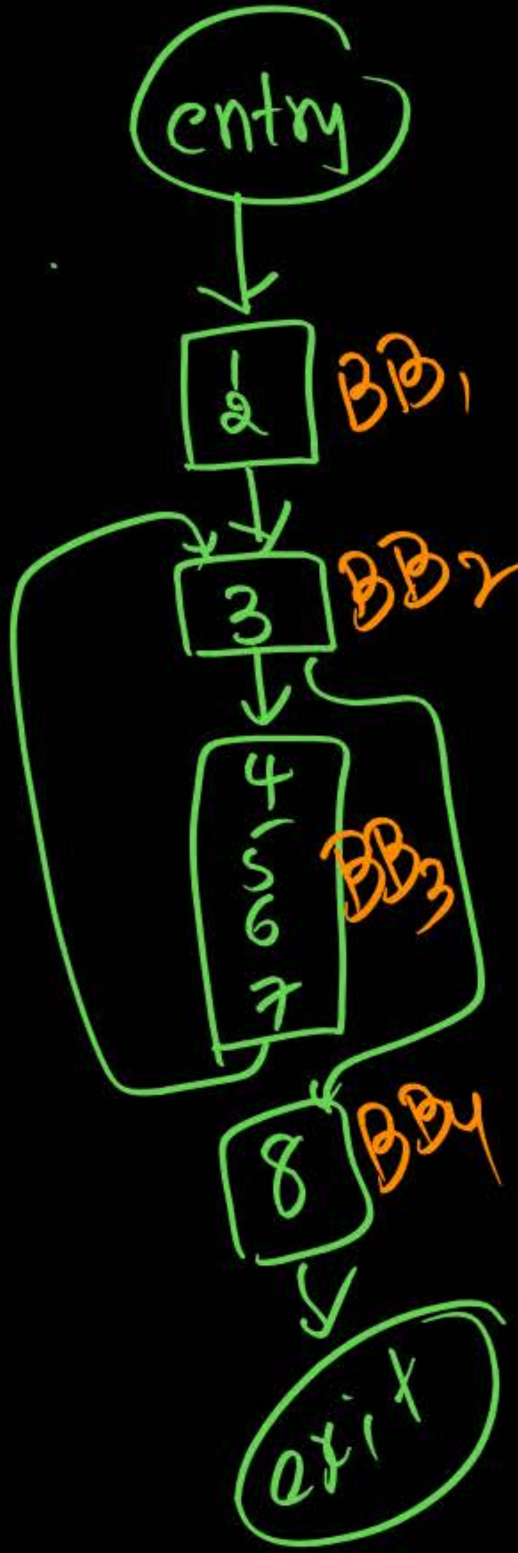
Q2) No. of nodes = 6

Q3) No. of edges = 6

```
1.      s := 0
2.      i := 1
3. L1:  if i > n goto L2
4.      t := i * i
5.      s := s + t
6.      i := i + 1
7.      goto L1
8. L2:  return s
```

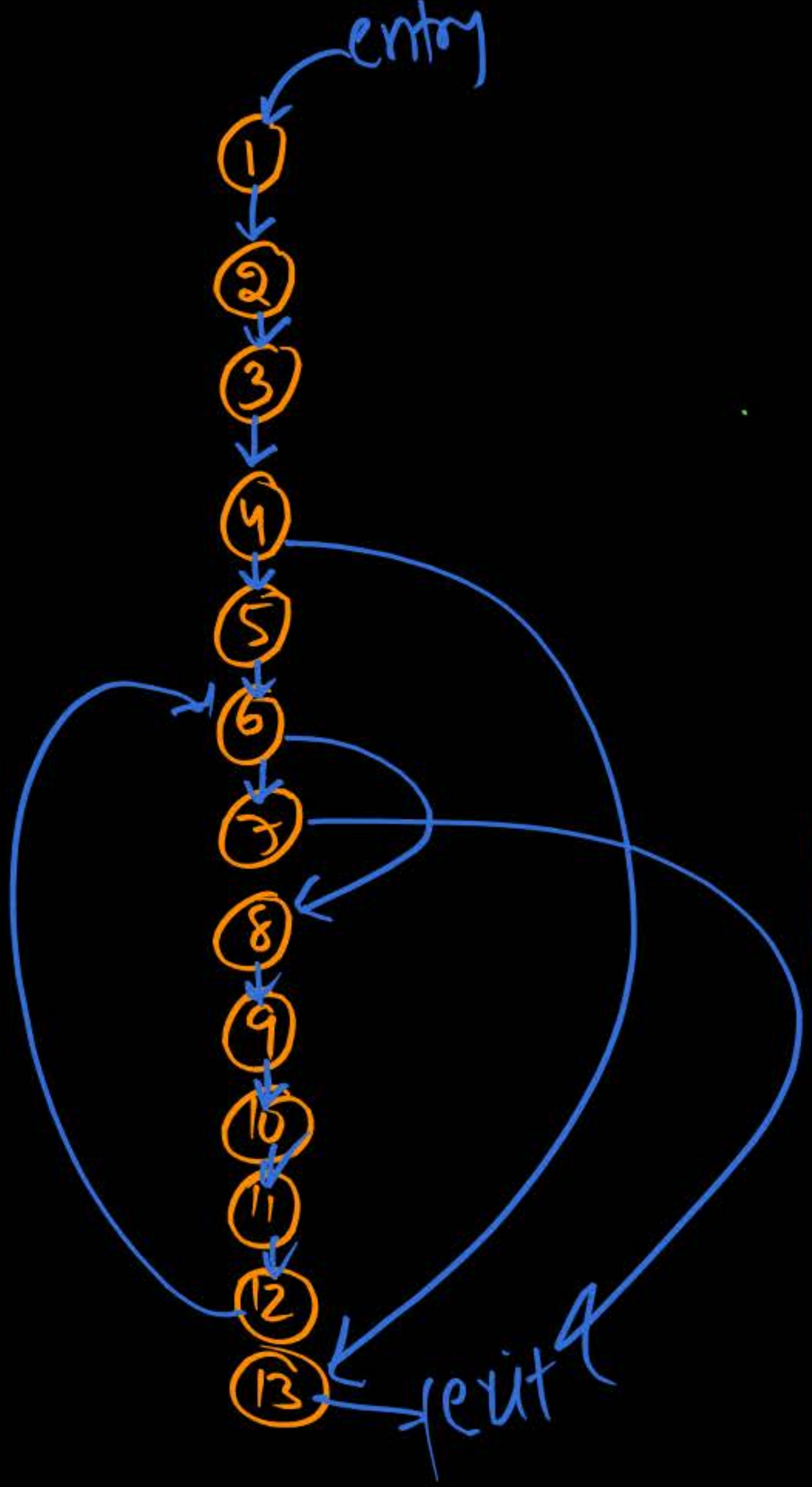
conditional jump

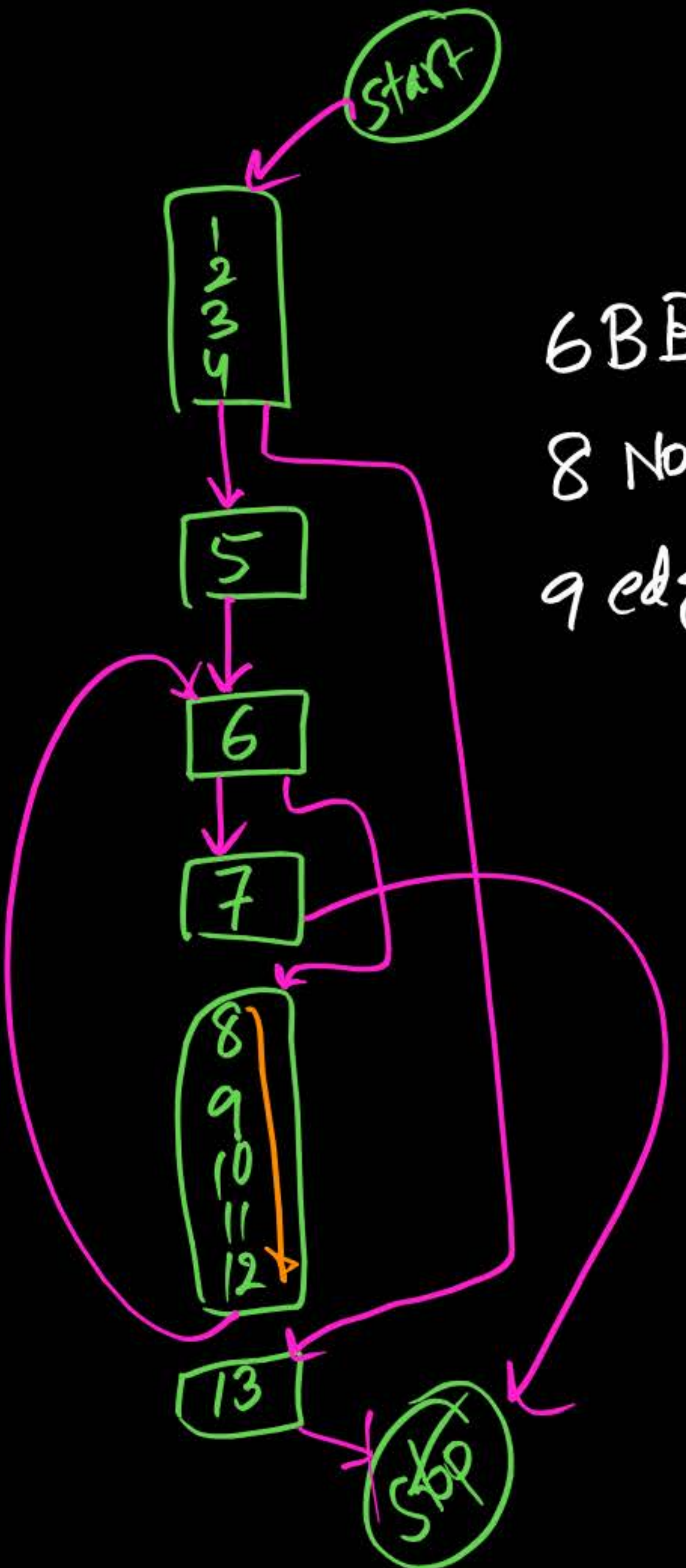




Control Flow Graph

```
1        receive m (val)
2        f0 ← 0
3        f1 ← 1
4        if m <= 1 goto L3
5        i ← 2
6    L1: if i <= m goto L2
7        return f2
8    L2: f2 ← f0 + f1
9        f0 ← f1
10       f1 ← f2
11       i ← i + 1
12       goto L1
13   L3: return m
```



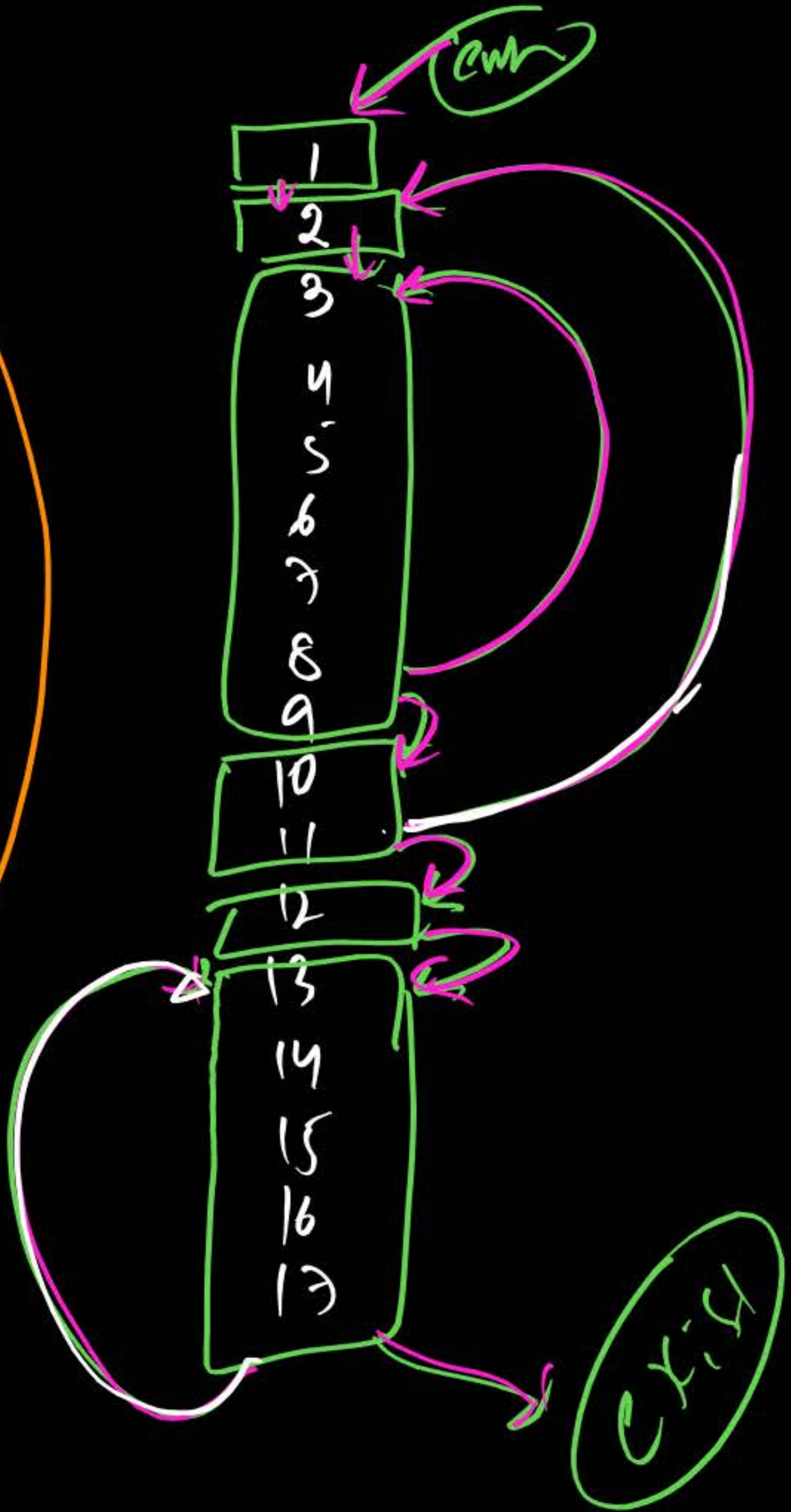


6 BBs

8 Nodes

9 edges

```
1)  i = 1
2)  j = 1
3)  t1 = 10 * i
4)  t2 = t1 + j
5)  t3 = 8 * t2
6)  t4 = t3 - 88
7)  a[t4] = 0.0
8)  j = j + 1
9)  if j <= 10 goto (3)
10) i = i + 1
11) if i <= 10 goto (2)
12) i = 1
13) t5 = i - 1
14) t6 = 88 * t5
15) a[t6] = 1.0
16) i = i + 1
17) if i <= 10 goto (13)
```

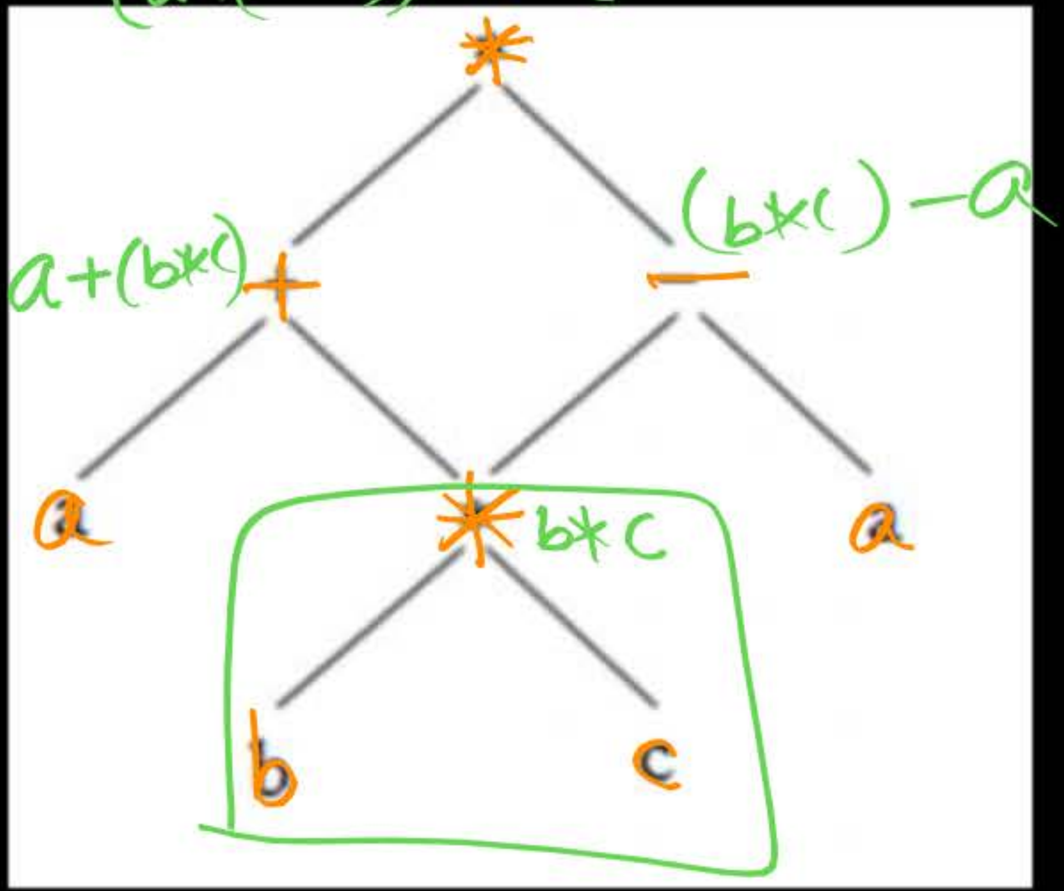


6 BBs
8 nodes
10 edges

Practice Questions

Q1 Find equivalent expression for the following syntax tree.

$(a+(b*c))*((b*c)-a) = (a+b*c)*(b*c-a)$



(a) $((a + b) * c) * (b * (c - a))$

(b) $a + (b * c - a)$

(c) $(a + (b * c)) * ((b * c) - a)$

(d) $a * (a + b * c) - a$

Q2 Find Equivalent Three Address code

X=((a*a)+(a*b))-(a*b

Q3

Find SSA Code

$$t_1 = b*c$$
$$t_2 = a+t_1$$
$$t_3 = b * c$$
$$t_4 = d * t_3$$
$$t_5 = t_2 + t_4$$

Q4 Finding minimum number of variables in 3AC and SSA

$$t_1 = a*b$$
$$t_2 = f/t_1$$
$$t_3 = a * t_2$$
$$t_4 = b * t_3$$
$$t_5 = t_4 + e$$

Q5 Finding minimum number of variables in 3AC and SSA

$$t_1 = (a + b)$$
$$t_2 = (c + d)$$
$$t_3 = t_1 + t_2$$
$$t_4 = t_1 + e$$
$$t_5 = t_4 + d$$

Q6 Finding minimum number of variables in 3AC and SSA

$$p = q + r$$
$$s = p + 1$$
$$t = q + r$$
$$u = s + t$$
$$v = t + u$$

Q7 Finding minimum number of nodes in DAG

$$p = q + r$$
$$s = p + 1$$
$$t = q + r$$
$$u = s + t$$
$$v = t + u$$

Q8

Three address codes can be implemented by

(a) indirect triples

(b) direct triples

(c) quadruples

(d) none of the above

Q9

An array a[n] is used in the following Pseudo code and each element of array is size of 8 bytes.

```
do
i=i+1;
while (a[i]>v);
```

Which of the following is equivalent 3-address code for above program.

(a) L: $t_1 = i+1$
$i = t_1$
$t_2 = i*8$
$t_3 = a[t_2]$
if $t_3 > v$ goto L

(b) L: $t_1 = i+1$
$t_2 = i*8$
$t_3 = a[t_2]$
if $t_3 < v$ goto L

(c) L: $t_1 = i+1$
$t_2 = i*8$
$t_3 = a[t_2]$
if $t_3 > v$ goto L

(d) L: $t_1 = i+1$
$i = t_1$
$t_2 = i*8$
$t_3 = a[t_2]$
if $t_3 < v$ goto L

Q10

Consider the following code segment.

```
x = u – t;
y = x * v;
x = y + w;
y = t – z;
y = x * y;
```

The minimum number of total variables required to convert the above code segment to static single assignment form is ______.

(GATE – 16 – SET1)

Q11

Consider the following intermediate program in three address code

$p = a - b$

$q = p * c$

$p = u * v$

$q = p + q$

Which one of the following corresponds to a static single assignment form of the above code? **(GATE – 17- SET1)**

(a)
$p_1 = a - b$
$q_1 = p_1 * c$
$p_1 = u * v$
$q_1 = p_1 + q_1$

(b)
$p_3 = a - b$
$q_4 = p_3 * c$
$p_4 = u * v$
$q_5 = p_4 + q_4$

(c)
$p_1 = a - b$
$q_1 = p_2 * c$
$p_3 = u * v$
$q_2 = p_4 + q_3$

(d)
$p_1 = a - b$
$q_1 = p * c$
$p_2 = u * v$
$q_2 = p + q$

Q12

For a C program accessing X[i] [j] [k], the following intermediate code is generated by a complier. Assume that the size of an integer is 32 bits and the size of character is 8 bits.

$t_0 = i * 1024$

$t_1 = j * 32$

$t_2 = k * 4$

$t_3 = t1 + t0$

$t_4 = t3 + t2$

$t_5 = X [t4]$

Which one of the following statements about the source code for the C program is **CORRECT**?

(GATE – 14 – SET2)

(a) X is declared as "int X[32] [32] [8]"

(b) X is declared as "int X [4] [1024] [32]"

(c) X is declared as "char X[4] [32] [8]"

(d) X is declared as "char X[32] [16] [2]"

Q13

The program below uses six temporary variables a, b, c, d, e, f.

```
a = 1
b = 10
c = 20
d = a + b
e = c + d
f = c + e
b = c + e
e = b + f
d = 5 + e
return d + f
```

Assuming that all operations take their operands from registers, what is the minimum number of registers needed to execute this program without spilling?

(GATE - 10)

(a) 2
(b) 3
(c) 4
(d) 6

Q14

Consider the basic block given below.

$a = b + c$

$c = a + d$

$d = b + c$

$e = d - b$

$a = e + b$

The minimum number of nodes and edges present in the DAG representation of the above basic block respectively are

(GATE – 14 – SET3)

(a) 6 and 6
(b) 8 and 10
(c) 9 and 12
(d) 4 and 4

Q15

Find number of nodes and edges in CFG.

```
(1) X := 20
(2) if X>=10 goto (8)
(3) X := X-1
(4) A[X] := 10
(5) if X<>4 goto (7)
(6) X := X-2
(7) goto (2)
(8) Y := X+5
```

Code optimization :

1) code optimization Techniques

2) Data Flow Analysis

Code optimization

→ To save space

→ To save Time

Code optimization

- High-Level
  - In IDEs
- Intermediate Level
  - 5^{th} phase
- Target Level (Assembly)
  - 7^{th} phase

Intermediate Level and Target Level: In compiler

Code optimization

- Statement Level
- Block Level (BasicBlock, peephole)
- Loop level
- Function Level [Intra-procedural]
- program Level [Inter-procedural]

Local optimizations

Global optimizations

Code Optimization Techniques:

1. Constant Folding
2. * Copy propogation
   - constant propogation
   - Variable propogation
3. * Common Sub expression elimination
4. * Strength Reduction
5. Algebraic Simplifications
6. * Dead code elimination
7. Loop optimizations
   - * code Motion
   - * Induction variables Elimination
   - loop Merging
   - loop unrolling

① Constant Folding:

$$x = \underbrace{2 * 3}_{6} + y \xRightarrow{\text{Constant Folding}} x = 6 + y$$

② Copy propogation :

I)

$$\boxed{\begin{array}{l} x = 20 \\ y = x + a \end{array}} \xrightarrow[\text{Replace } x \text{ with } 20]{\text{Constant propogation}} \boxed{\begin{array}{l} \underbrace{x = 20}_{\text{Dead code}} \\ y = 20 + a \end{array}} \xrightarrow{\text{Dead code elimination}} \boxed{y = 20 + a}$$

II)

$$\boxed{\begin{array}{l} x = b \\ y = x + a \end{array}} \xrightarrow[\text{Replace } x \text{ with 'b'}]{\text{Variable propogation}} \boxed{\begin{array}{l} \underbrace{x = b}_{\text{Dead code}} \\ y = b + a \end{array}} \Longrightarrow \boxed{y = b + a}$$

Example:

$$x = 10$$

$$y = x * 2 - b$$

Find possible optimizations.

A) Constant Folding

B) Copy propogation

C) Dead code Elimination

D) All of the above

$$\boxed{\begin{array}{l} x = 10 \\ y = x * 2 - b \end{array}}$$

$\Downarrow$ copy prop.

$$\boxed{\begin{array}{l} x = 10 \\ y = 10 * 2 - b \end{array}}$$

$\Downarrow$ Constant Folding

$$\boxed{\begin{array}{l} \cancel{x = 10} \text{ Deade} \\ y = 20 - b \end{array}}$$

$\Downarrow$ Dead code elimination

$$\boxed{y = 20 - b}$$

(3) Common Sub-expression Elimination:

↳ DAG can be used

$$x = \underbrace{(a+b)} * \underbrace{(a+b)} \Longrightarrow \begin{array}{l} t_1 = a+b \\ x = t_1 * t_1 \end{array}$$

④ Strength Reduction:

↳ It replaces costlier code with cheaper.

$$\underbrace{x = a * 2}_{\text{costlier}} \Longrightarrow x = a + a \quad \text{cheaper}$$

$$x = a * 2 \Longrightarrow x = a \ll 1 \quad \text{cheaper}$$

$$x = a * 8 \Longrightarrow x = a << 3$$

$$x = a / 8 \Longrightarrow x = a >> 3$$

5) Algebraic Simplifications :

Cancellation Law i) $x = a + \cancel{b} \cancel{-b} + c \Longrightarrow x = a + c$

Identity law ii) $x = a + \underbrace{b * 1}_{} \Longrightarrow x = a + b$

Identity law iii) $x = a + \underbrace{b + 0}_{} \Longrightarrow x = a + b$

Domination iv) $x = a + b + \underbrace{c * 0}_{0} \Longrightarrow x = a + b$

b^0

(6) Dead Code Elimination:

$$\boxed{x = a + b}$$ Dead code (useless)

$y = a * b$

$z = y + c$

print(z)

$\Longrightarrow$

$y = a * b$

$z = y + c$

print(z)

```
if (2>3)
        print("YOU")
else
        print("ME")
```

$\Rightarrow$

```
if (0)
    print("YOU")
else
    print("ME")
```

unreachable code (Dead code)

~~A)~~ Constant Folding

~~B)~~ Dead code Elimination

~~C)~~ Strength Reduction

Dead code elimination
Strength Reduction

```
print("ME")
```

7. Loop optimizations:

i) Code Motion:

```
for(i=0; i<n; i++)
{
    x = a+b
    y = x*i
}
```

loop invariant

Identify loop invariant code and Move outside loop

```
x = a+b
for(i=0;i<n;i++)
{
    y = x*i;
}
```

ii) Induction variables Elimination

```
for(i=0; i<n; i++)
{
  x=a+i
  y=b*i
  z=c+i
  p=x*y+z
}
```

```
k=0;
for(i=0, j=0; i<n; i++)
{
    x=a+i;
    y=b*j;
    z=c+k;
    p=x*y+z;
    j=j+1;
    k=k+1;
}
```

What is Induction Variable ?

↳ It depends on iterations (value changes in <u>some</u> iteration)

- i ✓
- j ✓
- k ✓
- ~~n~~
- ~~a~~
- ~~b~~
- ~~c~~
- x ✓
- y ✓
- z ✓
- p ✓

iii) Loop Merge / Loop Fusion

```
for(i=0; i<n; i++)
{
    A[i] = i+a;
}

for(j=0; j<n; j++)
{
    B[j] = j*b;
}
```

```
for(i=0; i<n; i++)
{
    A[i] = i+a;

    B[i] = i*b;

}
```

iv) Loop unrolling :

Half

```
for(i=0; i<4n; i++)
{ printf("GATE");
}
```

⟹ double

```
for(i=0; i<2n; i++)
{ printf("GATE");
  printf("GATE");
}
```

⟹

```
for(i=0; i<n; i++)
{ printf("GATE")
  printf("GATE")
  printf("GATE")
  printf("GATE")
}
```

GATEGATEGATE .... 4n times

GATEGATE ..... 4n times

Summary

- → CFG ✓
- → Code optimize Techniques ✓
- → Next: Live variable Analysis ***

Thank you
PW
Soldiers