Shri Ramdeobaba College of Engineering and Management, Nagpur

Department of CSE – Cyber Security
Session: 2023-2024
Compiler Design Lab

PRACTICAL No. 7

Name: Saloni Vishwakarma(C1-13)

Aim:

a. Write TAC to identify whether the number is prime or not.
b. Write a program to find leader statement, basic blocks, program flow graph & dominators.
c. Identify the Generate and Kill function for each block to be used in Elimination of loop invariant computation

Hint:
GEN (B): Set of all definitions generated in Block B.
KILL (B): Set of all definitions outside Block B that define the same variables which are defined in Block B.

Input: Three address code statements.

Output: 1) Leader Statements
2) Basic blocks
3) Program flow graph indicating the successor & predecessor.
4) Dominators of all the basic blocks
5) Natural loop detection

Sample input: 3AC

1. count = 0
2. Result = 0
3. If count > 20 GOTO 8
4. count=count + 1
5. increment = 2 * count
6. result = result +increment
7. GOTO 3
8. end

Sample Output:

A.

The leader statements are:
1) count=0
3) If count > 20 GOTO 8
4) count=count + 1
8) end

The Basic blocks are:
B1: contains: 1 & 2
B2 : contains 3
B3 : contains 4 5 6 7
B4 : contains 8

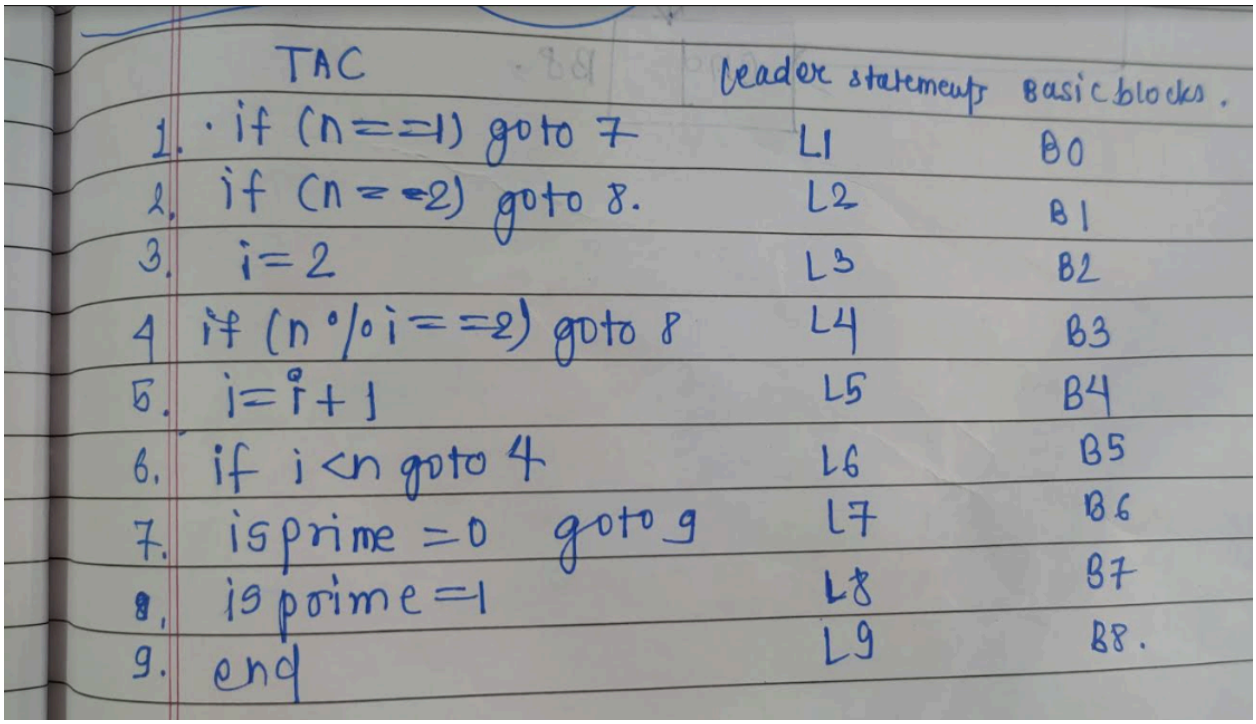
The PFG is
B1->B2
B2->B3
B2->B4
B3->B2

The dominators of all basic block are:
B1 →

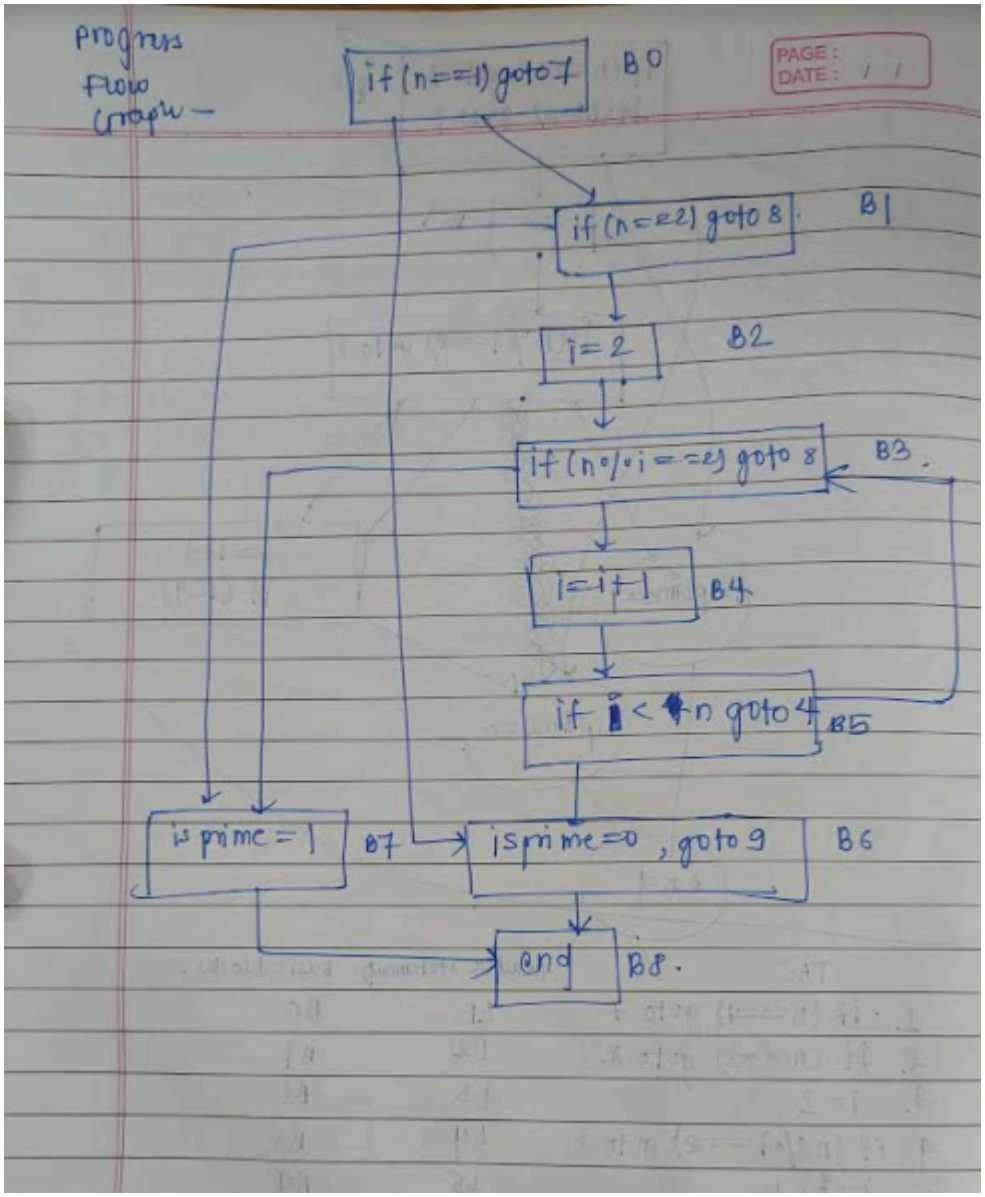
B.

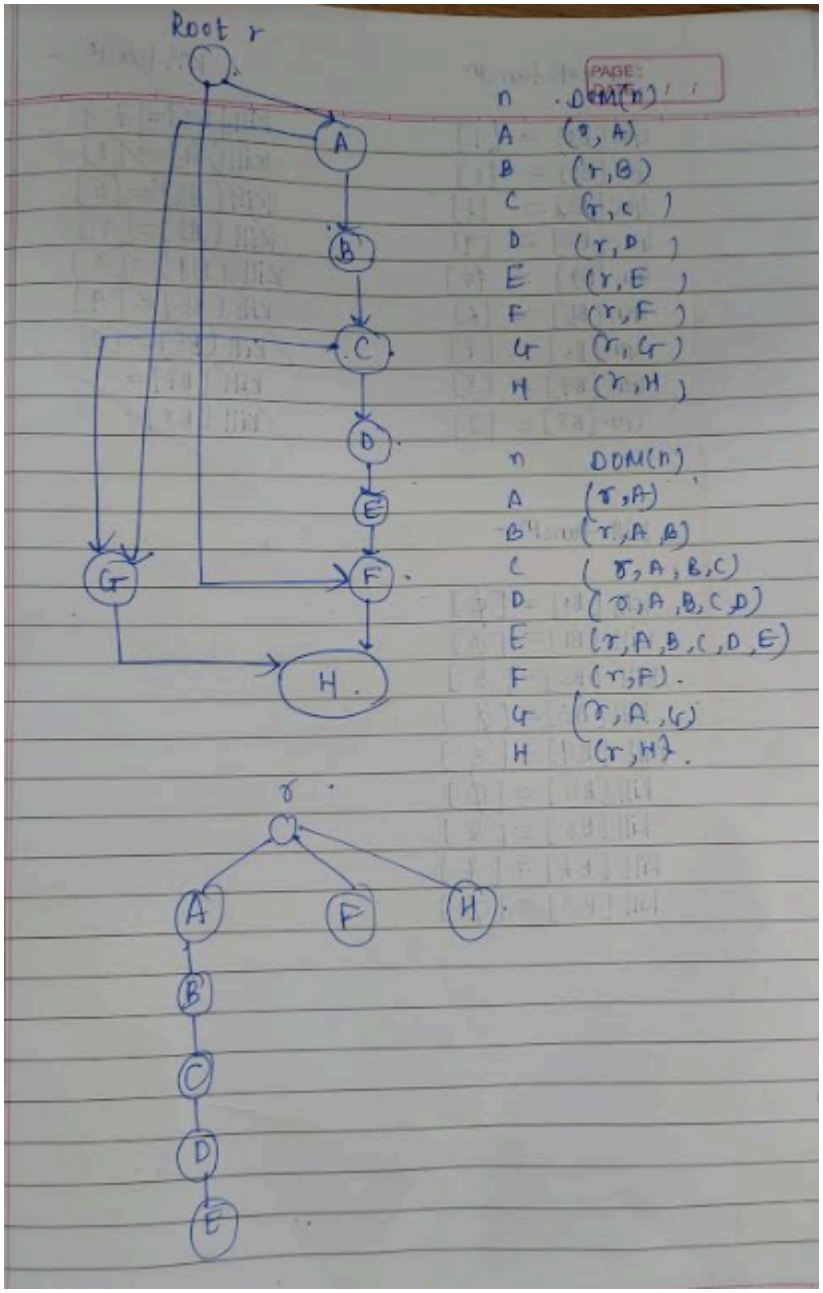
GEN(B1) = [1,2]
GEN(B2) = [3]
GEN(B3) = [4,5,6,7]
GEN(B4) = [8]

KILL(B1) = [4,6]
KILL(B2) = [Φ]
KILL(B3) = [1,2]
KILL(B4) = [Φ]



	TAC	leader statements	basic blocks.
1.	if (n==1) goto 7	L1	B0
2.	if (n==2) goto 8.	L2	B1
3.	i=2	L3	B2
4	if (n%i==2) goto 8	L4	B3
5.	i=i+1	L5	B4
6.	if i<n goto 4	L6	B5
7.	isprime =0 goto 9	L7	B6
8.	isprime=1	L8	B7
9.	end	L9	B8.


Program
flow
Graph –
if (n==1) goto 7
B0
if (n==2) goto 8
B1
i=2
B2
if (n%i==0) goto 8
B3
i=i+1
B4
if i<n goto 4
B5
is prime = 1
B7
isprime=0, goto 9
B6
end
B8.

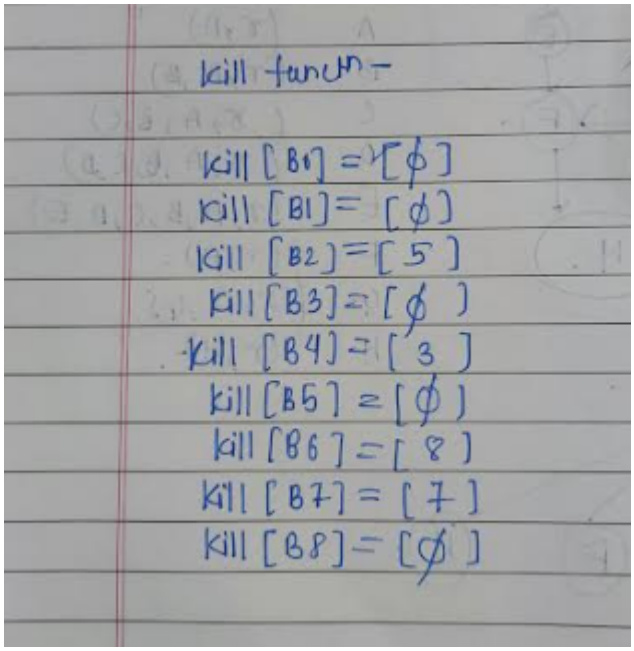


n	DOM(n)
A	(r, A)
B	(r, B)
C	(r, C)
D	(r, D)
E	(r, E)
F	(r, F)
G	(r, G)
H	(r, H)

n	DOM(n)
A	(r, A)
B'	(r, A, B)
C	(r, A, B, C)
D	(r, A, B, C, D)
E	(r, A, B, C, D, E)
F	(r, F).
G	(r, A, G)
H	(r, H}.

Generate function

Gen (B0) = [1]
Gen [B1] = [2]
Gen [B2] = [3]
Gen [B3] = [4]
Gen [B4] = [5]
Gen [B5] = [6]
Gen [B6] = [7]
Gen [B7] = [8]
Gen [B8] = [9]



Program code and Output:

```
#include <iostream>
#include <vector>
#include <unordered_set>
#include <unordered_map>
#include <string>
#include <sstream>

using namespace std;

class BasicBlock {
public:
    int start;
    int end;
    vector<int> leaders;

    BasicBlock(int start, int end) : start(start), end(end) {}
};

class ProgramFlowGraph {
public:
    int vertices;
    vector<vector<int>> adjacencyList;

    ProgramFlowGraph(int n) : vertices(n), adjacencyList(n) {}

    void addEdge(int u, int v) {
        adjacencyList[u].push_back(v);
    }
};
```

```
vector<string> split(const string& s, char delimiter) {
    vector<string> tokens;
    stringstream ss(s);
    string token;
    while (getline(ss, token, delimiter)) {
        tokens.push_back(token);
    }
    return tokens;
}

vector<int> findLeaders(const vector<string>& code) {
    vector<int> leaders;
    for (int i = 0; i < code.size(); ++i) {
        leaders.push_back(i);
    }
    return leaders;
}

vector<BasicBlock> findBasicBlocks(const vector<string>& code, const vector<int>&
leaders) {
    vector<BasicBlock> basicBlocks;
    for (int i = 0; i < leaders.size(); ++i) {
        basicBlocks.emplace_back(i, i);
    }
    return basicBlocks;
}

ProgramFlowGraph buildProgramFlowGraph(const vector<string>& code, const
vector<BasicBlock>& basicBlocks) {
    ProgramFlowGraph graph(basicBlocks.size());
    for (int i = 0; i < basicBlocks.size(); ++i) {
        if (i < basicBlocks.size() - 1) {
            graph.addEdge(i, i + 1);
        }
    }
    return graph;
}

vector<unordered_set<int>> computeDominators(const ProgramFlowGraph& graph) {
    vector<unordered_set<int>> dominators(graph.vertices);
    for (int i = 0; i < graph.vertices; ++i) {
        for (int j = 0; j < graph.vertices; ++j) {
            dominators[i].insert(j);
        }
        dominators[i].erase(i);
        dominators[i].insert(i);
```

```
    }
    return dominators;
}

int main() {
    vector<string> code = {
        "if n==1 goto 7",
        "if n==2 goto 8",
        "i=2",
        "if(n%i==0) goto 8",
        "i=i+1",
        "if i<n goto 4",
        "isprime=0 goto 9",
        "isprime=1",
        "end"
    };

    vector<int> leaders = findLeaders(code);
    vector<BasicBlock> basicBlocks = findBasicBlocks(code, leaders);
    cout << "Leader Statements:" << endl;
    for (int i = 0; i < leaders.size(); ++i) {
        cout << "L" << i + 1 << ": " << code[i] << endl;
    }
    cout << "\nBasic Blocks:" << endl;
    for (int i = 0; i < basicBlocks.size(); ++i) {
        cout << "B" << i << ": " << code[i] << endl;
    }
    ProgramFlowGraph graph = buildProgramFlowGraph(code, basicBlocks);
    cout << "\nProgram Flow Graph (PFG):" << endl;
    for (int i = 0; i < graph.vertices; ++i) {
        cout << "B" << i << " -> ";
        for (int vertex : graph.adjacencyList[i]) {
            cout << "B" << vertex << " ";
        }
        cout << endl;
    }
    vector<unordered_set<int>> dominators = computeDominators(graph);
    cout << "\nDominators of all Basic Blocks:" << endl;
    for (int i = 0; i < dominators.size(); ++i) {
        cout << "B" << i << " → ";
        for (int dom : dominators[i]) {
            cout << "B" << dom << " ";
        }
        cout << endl;
    }

    return 0;
```

}

```
Leader Statements:
L1: if n==1 goto 7
L2: if n==2 goto 8
L3: i=2
L4: if(n%i==0) goto 8
L5: i=i+1
L6: if i<n goto 4
L7: isprime=0 goto 9
L8: isprime=1
L9: end

Basic Blocks:
B0: if n==1 goto 7
B1: if n==2 goto 8
B2: i=2
B3: if(n%i==0) goto 8
B4: i=i+1
B5: if i<n goto 4
B6: isprime=0 goto 9
B7: isprime=1
B8: end

Program Flow Graph (PFG):
B0 -> B1
B1 -> B2
B2 -> B3
B3 -> B4
B4 -> B5
B5 -> B6
B6 -> B7
B7 -> B8
B8 ->

Dominators of all Basic Blocks:
B0 → B0 B8 B7 B6 B5 B4 B3 B2 B1
B1 → B1 B8 B7 B6 B5 B4 B3 B2 B0
B2 → B2 B8 B7 B6 B5 B4 B3 B1 B0
B3 → B3 B8 B7 B6 B5 B4 B2 B1 B0
B4 → B4 B8 B7 B6 B5 B3 B2 B1 B0
B5 → B5 B8 B7 B6 B4 B3 B2 B1 B0
B6 → B6 B8 B7 B5 B4 B3 B2 B1 B0
B7 → B7 B8 B6 B5 B4 B3 B2 B1 B0
B8 → B8 B7 B6 B5 B4 B3 B2 B1 B0
```