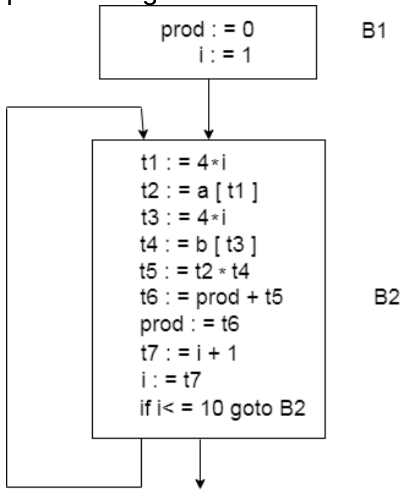
BHARATIYA VIDYA BHAVAN’S
SARDAR PATEL INSTITUTE OF TECHNOLOGY
(Empowered Autonomous Institute Affiliated to University of Mumbai)
[Knowledge is Nectar]

Department of Computer Engineering

Course - System Programming and Compiler Construction (SPCC)

UID	2021300108
Name	Hatim Sawai
Class and Batch	TE Computer Engineering Class B – Batch C
Date	1/4/2024
Lab #	6
Aim	Write a program to find Basic blocks and generate flow graph for the given three address code.
Objective	Input: Three Address Code (Assembly Code Snippet) Output: Print generated Control Flow graph from identified basic blocks
Theory	**Control Flow Graph:** The flow graph is a directed graph. It contains the flow of control information for the set of basic blocks. A control flow graph is used to depict that how the program control is being parsed among the blocks. It is useful in the loop optimization. Flow graph for the vector dot product is given as follows:  prod : = 0 i : = 1 B1 t1 : = 4*i t2 : = a [t1] t3 : = 4*i t4 : = b [t3] t5 : = t2 * t4 t6 : = prod + t5 prod : = t6 t7 : = i + 1 i : = t7 if i< = 10 goto B2 B2 Block B1 is the initial node. Block B2 immediately follows B1, so from B2 to B1 there is an edge. The target of jump from last statement of B1 is the first statement B2, so from B1 to B2 there is an edge. B2 is a successor of B1 and B1 is the predecessor of B2.

Basic Block

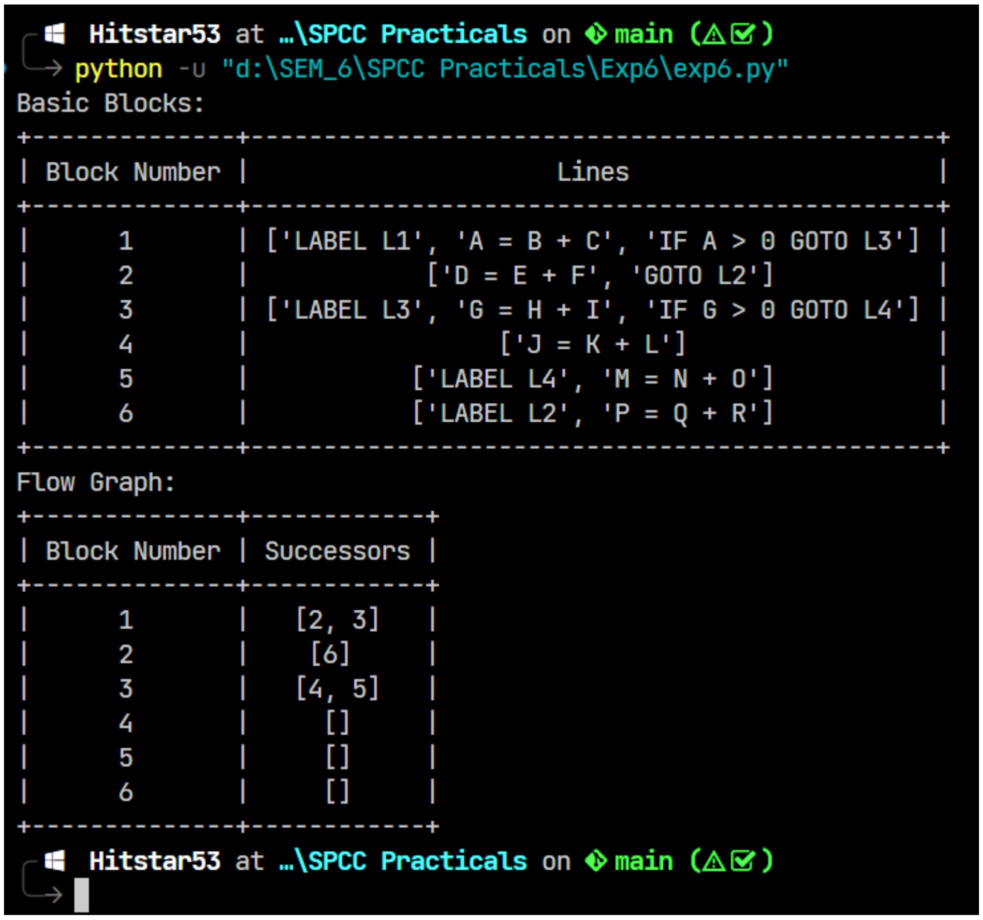
Basic block contains a sequence of statement. The flow of control enters at the beginning of the statement and leave at the end without any halt (except may be the last instruction of the block).

The following sequence of three address statements forms a basic block:

```
t1:= x * x
t2:= x * y
t3:= 2 * t2
t4:= t1 + t3
t5:= y * y
t6:= t4 + t5
```

Implementation / Code

```python
from prettytable import PrettyTable
def find_basic_blocks(code):
    basic_blocks = []
    current_block = []

    for line in range(len(code)):
        if code[line].startswith('LABEL'):
            if current_block:
                basic_blocks.append(current_block)
                current_block = []
            current_block.append(code[line])
        elif code[line-1].startswith('IF'):
            basic_blocks.append(current_block)
            current_block = [code[line]]
        else:
            current_block.append(code[line])

    if current_block:
        basic_blocks.append(current_block)

    return basic_blocks


def generate_flow_graph(basic_blocks):
    flow_graph = {}
    block_number = 1
    for block_num, block in enumerate(basic_blocks):
        successors = []
        for line_num in range(len(block)):
            if 'GOTO' in block[line_num]:
                goto_block = block[line_num].split()[-1]
                goto_block_num = None
                for i, blk in enumerate(basic_blocks):
                    if blk[0].split()[1] == goto_block:
                        goto_block_num = i + 1
                        break
                if goto_block_num is not None:
                    successors.append(goto_block_num)
            elif block[-1].startswith('IF'):
                if block_num +2 not in successors:
                    successors.append(block_num + 2)
            elif block[line_num].startswith('IF') or (line_num > 0 and
block[line_num-1].startswith('IF')):
                conditions = block[line_num].split()[2:]
                for condition in conditions:
                    goto_block = condition.split(':')[1]
                    goto_block_num = None
                    for i, blk in enumerate(basic_blocks):
                        if blk[0].split()[1] == goto_block:
                            goto_block_num = i + 1
                            break
                    if goto_block_num is not None:
                        successors.append(goto_block_num)

        flow_graph[block_number] = successors
        block_number += 1

    return flow_graph

def main():
    code = [
        'LABEL L1',
        'A = B + C',
        'IF A > 0 GOTO L3',
        'D = E + F',
        'GOTO L2',
        'LABEL L3',
        'G = H + I',
        'IF G > 0 GOTO L4',
        'J = K + L',
        'LABEL L4',
        'M = N + O',
        'LABEL L2',
        'P = Q + R'
    ]

    basic_blocks = find_basic_blocks(code)
    table = PrettyTable()
    print("Basic Blocks:")
    table.field_names = ["Block Number", "Lines"]
    for i in range(len(basic_blocks)):
        table.add_row([i+1, basic_blocks[i]])
    print(table)
    flow_graph = generate_flow_graph(basic_blocks)
    table = PrettyTable()
    table.field_names = ["Block Number", "Successors"]
    for block_num, successors in flow_graph.items():
        table.add_row([block_num, successors])
    print("Flow Graph:")
    print(table)

if __name__ == "__main__":
    main()
```

Output	
Conclusion	In this experiment, we learned how to construct and identify basic blocks from given three address code in assembly and then generate a control flow graph for it.
References	[1] Javatpoint: Flow Graph https://www.javatpoint.com/flow-graph [2] Javatpoint: Basic Block https://www.javatpoint.com/basic-block

```
 Hitstar53 at …\SPCC Practicals on  main (⚠☑)
╰→ python -u "d:\SEM_6\SPCC Practicals\Exp6\exp6.py"
Basic Blocks:
+--------------+------------------------------------------------+
| Block Number |                     Lines                      |
+--------------+------------------------------------------------+
|      1       | ['LABEL L1', 'A = B + C', 'IF A > 0 GOTO L3'] |
|      2       |            ['D = E + F', 'GOTO L2']            |
|      3       | ['LABEL L3', 'G = H + I', 'IF G > 0 GOTO L4'] |
|      4       |                 ['J = K + L']                  |
|      5       |            ['LABEL L4', 'M = N + O']           |
|      6       |            ['LABEL L2', 'P = Q + R']           |
+--------------+------------------------------------------------+
Flow Graph:
+--------------+------------+
| Block Number | Successors |
+--------------+------------+
|      1       |   [2, 3]   |
|      2       |    [6]     |
|      3       |   [4, 5]   |
|      4       |     []     |
|      5       |     []     |
|      6       |     []     |
+--------------+------------+
 Hitstar53 at …\SPCC Practicals on  main (⚠☑)
╰→ █
```