**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:  Flow Graph  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: |
| **Implementation / Code** | 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> |