Dot product

4/3/2022

Amuyla .R

016006094

Introduction:

We have to implement subset of the mips ISA that can execute the dot product benchmark with forward chaining using Python programming language.

Description of the design:

There are 5 stages in this RISC architecture and are stated one by one below –

1. Instruction Fetch (IF):

In this stage PC is sent to instruction memory and the current instruction is fetched from memory as well update the PC to next in sequence by adding 4 to the PC (PC = PC+4).

2. Instruction Decode (ID):

Instruction Decode is the second stage of MIPS pipeline. In this stage, the instruction gets decoded and the registers are read as specified in instruction. If there is a branch 8 instruction, the registers are compared as they are read. Moreover, Sign extends the offset field if it is needed. Compute the possible branch target address. Decoding can be done in parallel with reading the registers since the register specifiers at a fixed location; this is called as ‘fixed field decoding’

3. Execute (EX):

In this stage, ALU operations based on the instruction type. In terms of memory instructions, it adds base address and offset to acquire effective address. For register –register operations, as per the ALU – opcode it performs addition, subtraction as it is needed. It also performs operation for register –immediate ALU instructions.

4. Memory access (MEM):

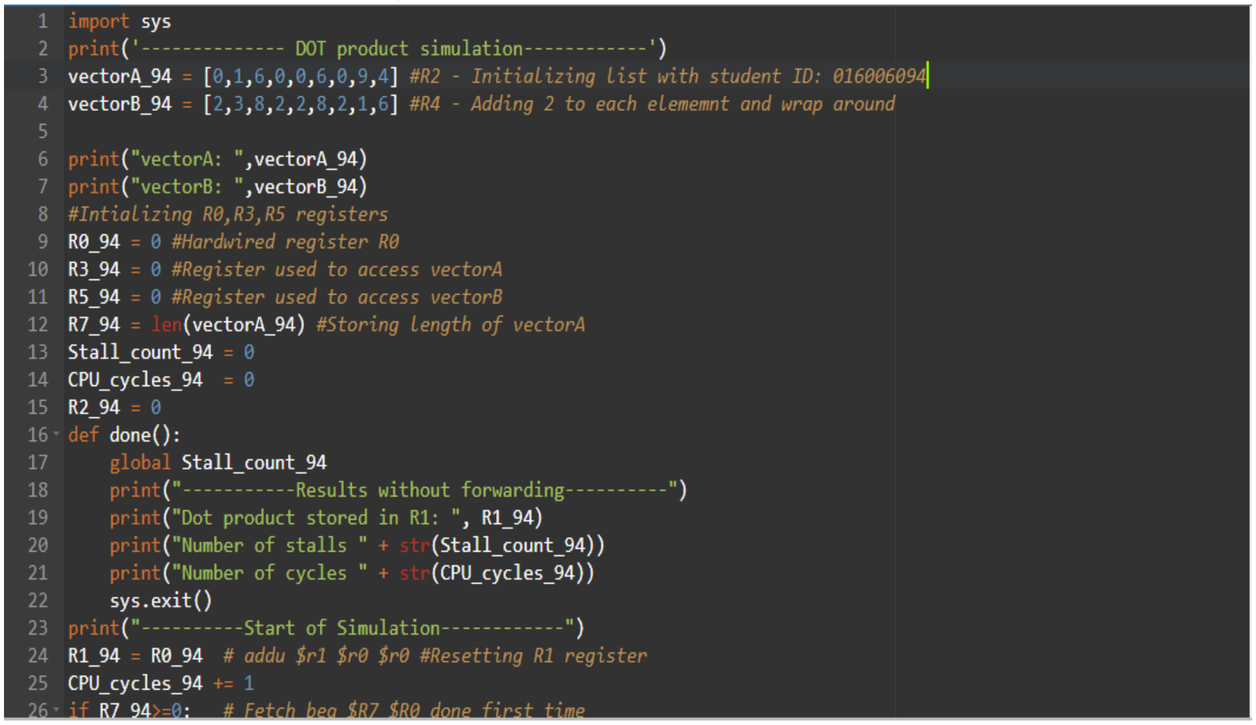
In this stage, load and store instructions are being performed. If it is a load instruction, then it reads an effective address from the memory and in the case of store instruction it writes the data from register in to memory.

5. Write Back (WB):
This is the last stage and it performs register – register ALU instruction or LOAD instruction to write the result in to register file (at ID stage), to check whether it comes through load instruction or from ALU when it is a case of ALU instruction

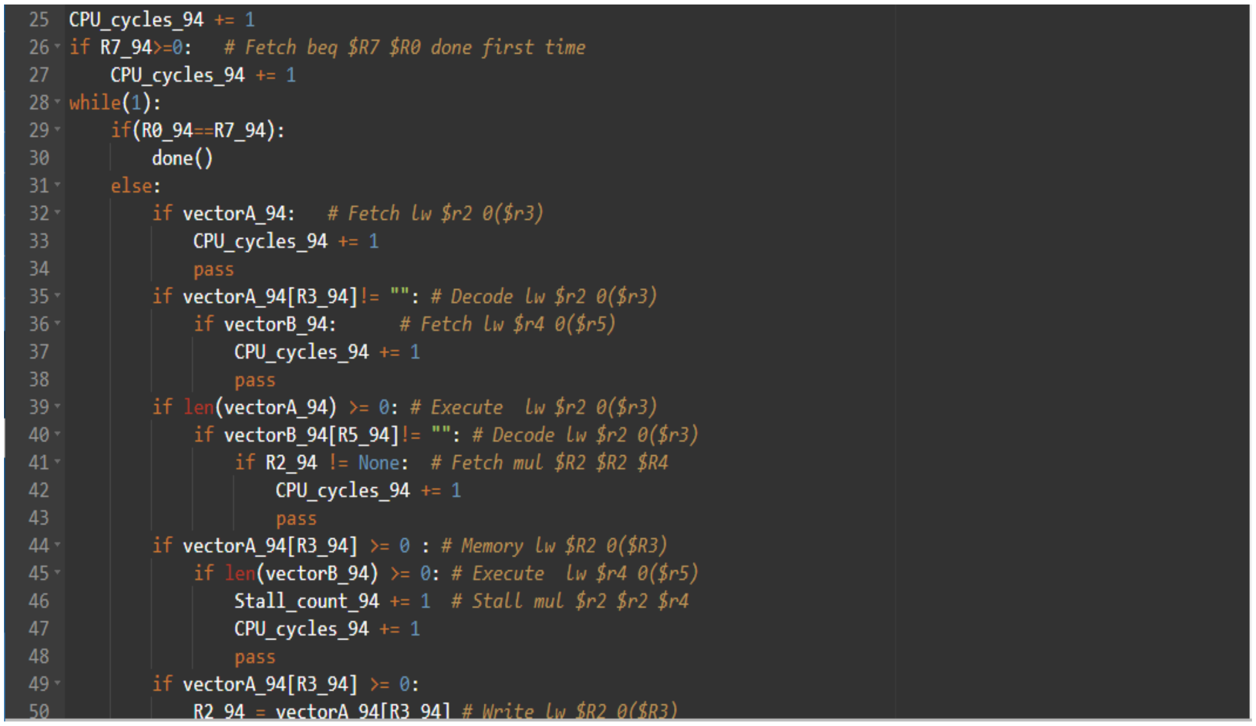
In case of Forwarding, the data hazard is detected in the following cases –
The data hazard is detected when-
1a.) EX/MEM.RegisterRd = ID/EX.RegisterRs
1b.) EX/MEM.RegisterRd = ID/EX.RegisterRt
2a.) MEM/WB.RegisterRd = ID/EX.RegisterRs
2b.) MEM/WB.RegisterRd = ID/EX.RegisterRt
; where Rs and Rd are Source, destination registers.
The conditions for detecting hazards and the control signals to resolve them:
1. *EX hazard:*
if (EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0) and (EX/MEM.RegisterRd = ID/EX.RegisterRs)) ForwardA = 10
if (EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0) and (EX/MEM.RegisterRd = ID/EX.RegisterRt)) ForwardB = 10
2. *MEM hazard:*
if (MEM/WB.RegWrite and (MEM/WB.RegisterRd ≠ 0) and (MEM/WB.RegisterRd = ID/EX.RegisterRs)) ForwardA = 01
if (MEM/WB.RegWrite and (MEM/WB.RegisterRd ≠ 0) and (MEM/WB.RegisterRd = ID/EX.RegisterRt)) ForwardB = 01[1]

Code:

Without Forward Chaining

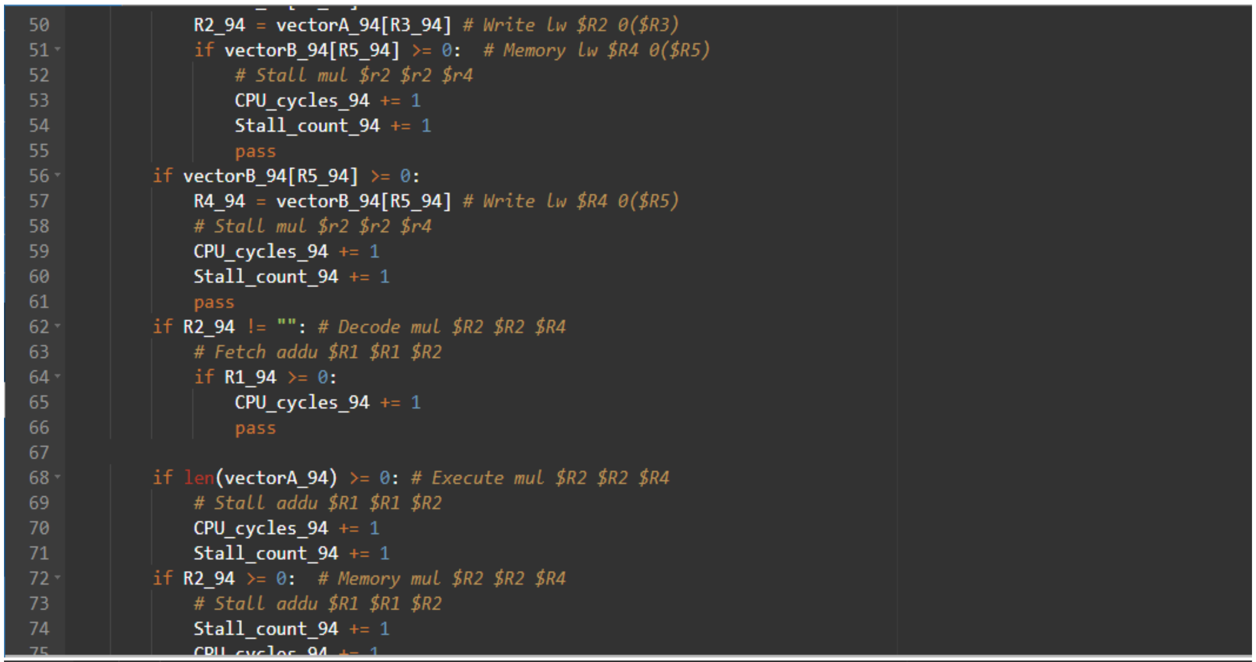


```python
import sys
print('------------- DOT product simulation-----------')
vectorA_94 = [0,1,6,0,0,6,0,9,4] #R2 - Initializing list with student ID: 016006094
vectorB_94 = [2,3,8,2,2,8,2,1,6] #R4 - Adding 2 to each elememnt and wrap around

print("vectorA: ",vectorA_94)
print("vectorB: ",vectorB_94)
#Intializing R0,R3,R5 registers
R0_94 = 0 #Hardwired register R0
R3_94 = 0 #Register used to access vectorA
R5_94 = 0 #Register used to access vectorB
R7_94 = len(vectorA_94) #Storing length of vectorA
Stall_count_94 = 0
CPU_cycles_94  = 0
R2_94 = 0
def done():
    global Stall_count_94
    print("----------Results without forwarding---------")
    print("Dot product stored in R1: ", R1_94)
    print("Number of stalls " + str(Stall_count_94))
    print("Number of cycles " + str(CPU_cycles_94))
    sys.exit()
print("---------Start of Simulation-----------")
R1_94 = R0_94  # addu $r1 $r0 $r0 #Resetting R1 register
CPU_cycles_94 += 1
if R7_94>=0:   # Fetch beq $R7 $R0 done first time
```

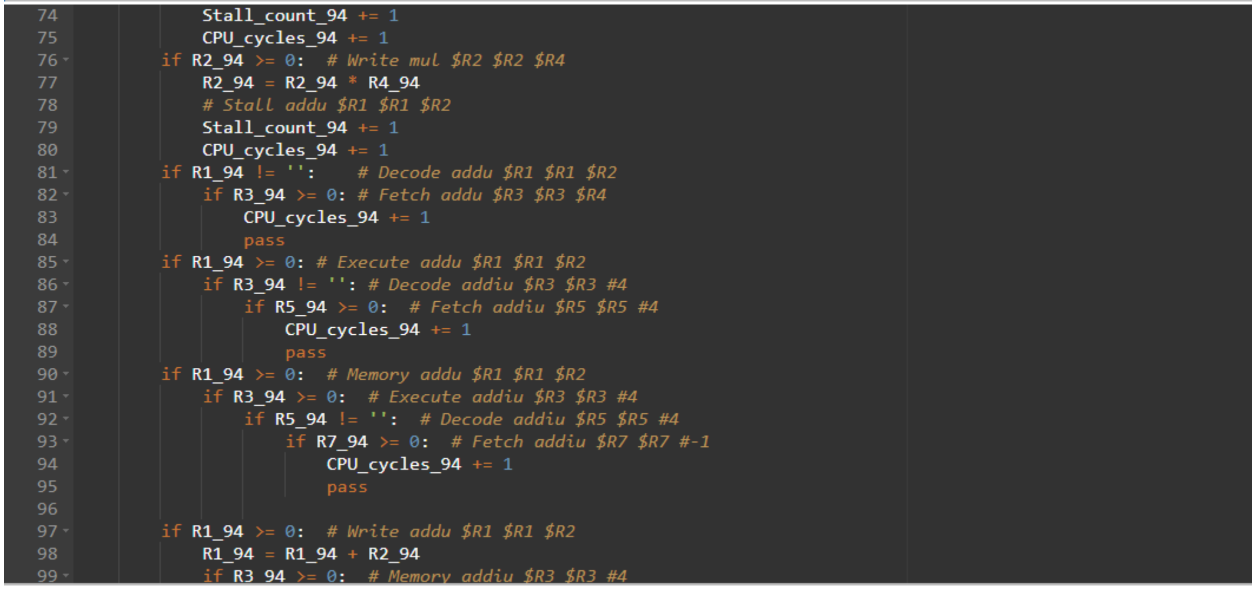


```python
CPU_cycles_94 += 1
if R7_94>=0:   # Fetch beq $R7 $R0 done first time
```

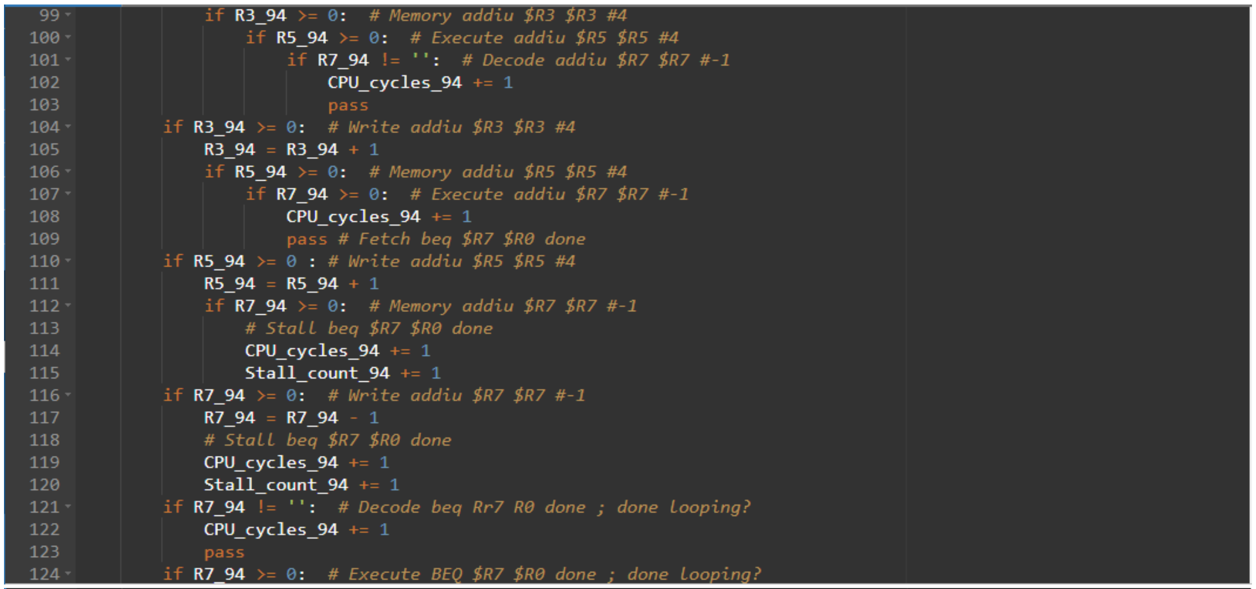
```python
    CPU_cycles_94 += 1
while(1):
    if(R0_94==R7_94):
        done()
    else:
        if vectorA_94:   # Fetch lw $r2 0($r3)
            CPU_cycles_94 += 1
            pass
        if vectorA_94[R3_94]!= "": # Decode lw $r2 0($r3)
            if vectorB_94:      # Fetch lw $r4 0($r5)
                CPU_cycles_94 += 1
                pass
        if len(vectorA_94) >= 0: # Execute  lw $r2 0($r3)
            if vectorB_94[R5_94]!= "": # Decode lw $r2 0($r3)
                if R2_94 != None:  # Fetch mul $R2 $R2 $R4
                    CPU_cycles_94 += 1
                    pass
        if vectorA_94[R3_94] >= 0 : # Memory lw $R2 0($R3)
            if len(vectorB_94) >= 0: # Execute  lw $r4 0($r5)
                Stall_count_94 += 1  # Stall mul $r2 $r2 $r4
                CPU_cycles_94 += 1
                pass
        if vectorA_94[R3_94] >= 0:
            R2_94 = vectorA_94[R3_94] # Write lw $R2 0($R3)
```



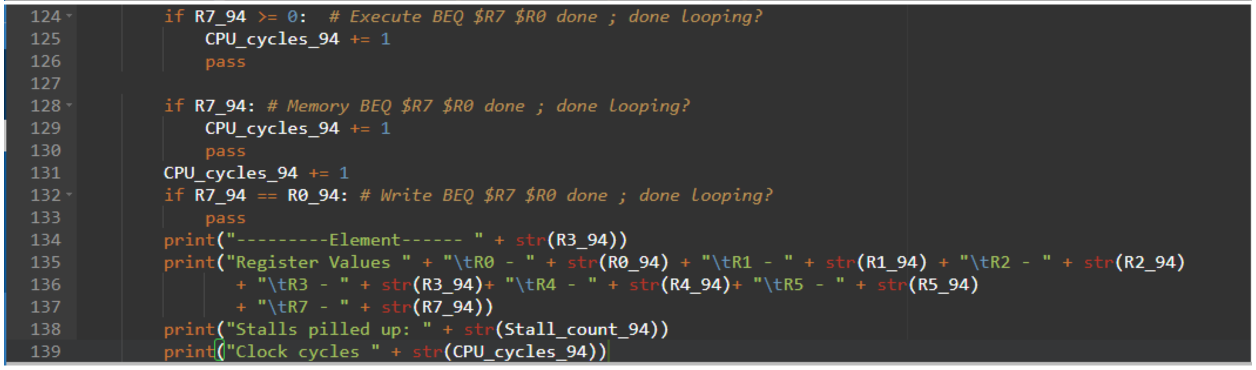
```
            R2_94 = vectorA_94[R3_94] # Write lw $R2 0($R3)
            if vectorB_94[R5_94] >= 0:  # Memory lw $R4 0($R5)
                # Stall mul $r2 $r2 $r4
                CPU_cycles_94 += 1
                Stall_count_94 += 1
                pass
        if vectorB_94[R5_94] >= 0:
            R4_94 = vectorB_94[R5_94] # Write lw $R4 0($R5)
            # Stall mul $r2 $r2 $r4
            CPU_cycles_94 += 1
            Stall_count_94 += 1
            pass
        if R2_94 != "": # Decode mul $R2 $R2 $R4
            # Fetch addu $R1 $R1 $R2
            if R1_94 >= 0:
                CPU_cycles_94 += 1
                pass

        if len(vectorA_94) >= 0: # Execute mul $R2 $R2 $R4
            # Stall addu $R1 $R1 $R2
            CPU_cycles_94 += 1
            Stall_count_94 += 1
        if R2_94 >= 0:  # Memory mul $R2 $R2 $R4
            # Stall addu $R1 $R1 $R2
            Stall_count_94 += 1
```



```
            Stall_count_94 += 1
            CPU_cycles_94 += 1
        if R2_94 >= 0:  # Write mul $R2 $R2 $R4
            R2_94 = R2_94 * R4_94
            # Stall addu $R1 $R1 $R2
            Stall_count_94 += 1
            CPU_cycles_94 += 1
        if R1_94 != '':    # Decode addu $R1 $R1 $R2
            if R3_94 >= 0: # Fetch addu $R3 $R3 $R4
                CPU_cycles_94 += 1
                pass
        if R1_94 >= 0: # Execute addu $R1 $R1 $R2
            if R3_94 != '': # Decode addiu $R3 $R3 #4
                if R5_94 >= 0:  # Fetch addiu $R5 $R5 #4
                    CPU_cycles_94 += 1
                    pass
        if R1_94 >= 0:  # Memory addu $R1 $R1 $R2
            if R3_94 >= 0:  # Execute addiu $R3 $R3 #4
                if R5_94 != '':  # Decode addiu $R5 $R5 #4
                    if R7_94 >= 0:  # Fetch addiu $R7 $R7 #-1
                        CPU_cycles_94 += 1
                        pass

        if R1_94 >= 0:  # Write addu $R1 $R1 $R2
            R1_94 = R1_94 + R2_94
            if R3_94 >= 0:  # Memory addiu $R3 $R3 #4
```



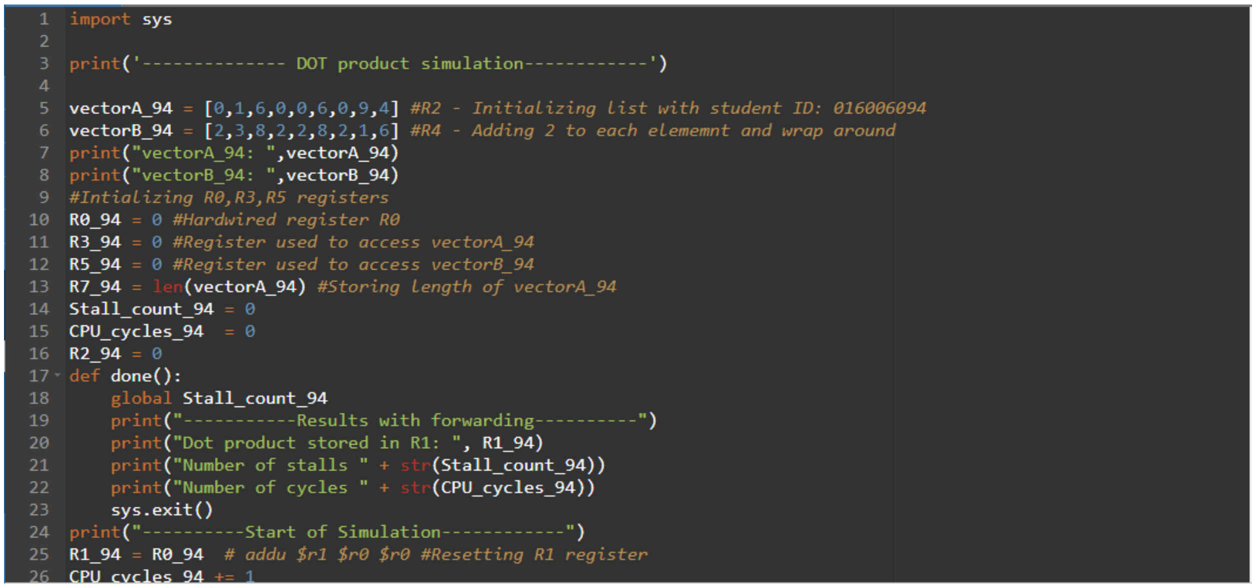
```
            if R3_94 >= 0:  # Memory addiu $R3 $R3 #4
                if R5_94 >= 0:  # Execute addiu $R5 $R5 #4
                    if R7_94 != '':  # Decode addiu $R7 $R7 #-1
                        CPU_cycles_94 += 1
                        pass
        if R3_94 >= 0:  # Write addiu $R3 $R3 #4
            R3_94 = R3_94 + 1
            if R5_94 >= 0:  # Memory addiu $R5 $R5 #4
                if R7_94 >= 0:  # Execute addiu $R7 $R7 #-1
                    CPU_cycles_94 += 1
                    pass # Fetch beq $R7 $R0 done
        if R5_94 >= 0 : # Write addiu $R5 $R5 #4
            R5_94 = R5_94 + 1
            if R7_94 >= 0:  # Memory addiu $R7 $R7 #-1
                # Stall beq $R7 $R0 done
                CPU_cycles_94 += 1
                Stall_count_94 += 1
        if R7_94 >= 0:  # Write addiu $R7 $R7 #-1
            R7_94 = R7_94 - 1
            # Stall beq $R7 $R0 done
            CPU_cycles_94 += 1
            Stall_count_94 += 1
        if R7_94 != '':  # Decode beq Rr7 R0 done ; done looping?
            CPU_cycles_94 += 1
            pass
        if R7_94 >= 0:  # Execute BEQ $R7 $R0 done ; done looping?
```



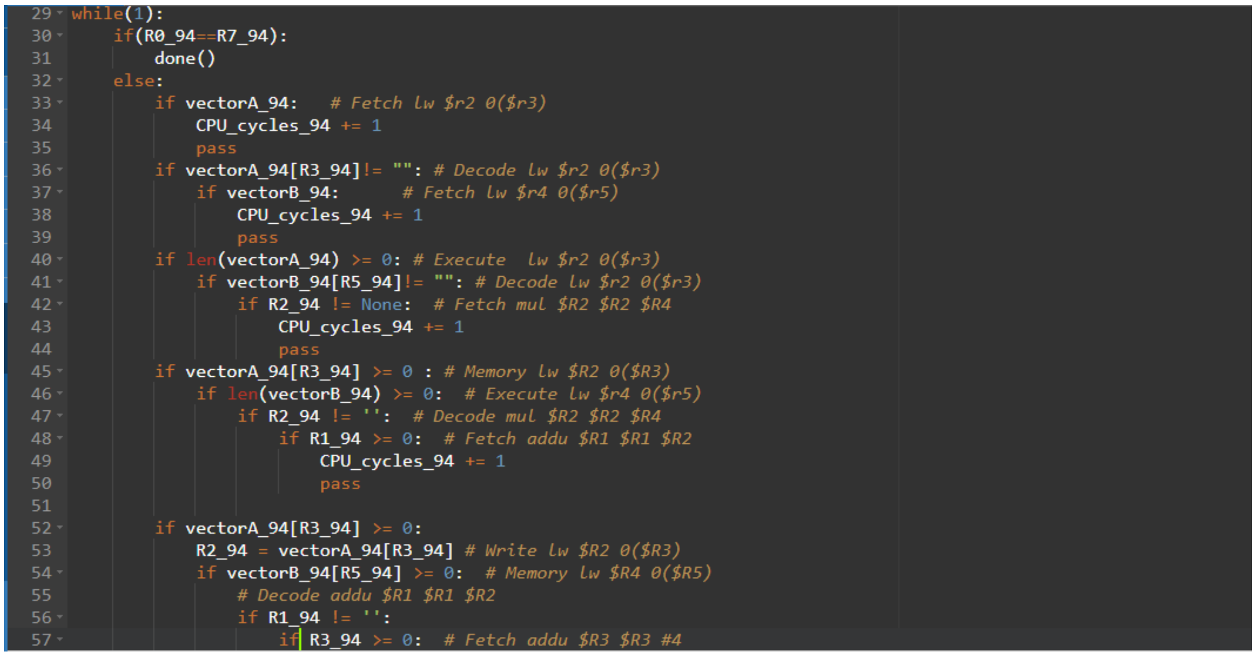
```
        if R7_94 >= 0:  # Execute BEQ $R7 $R0 done ; done looping?
```

```
            CPU_cycles_94 += 1
            pass

        if R7_94: # Memory BEQ $R7 $R0 done ; done looping?
            CPU_cycles_94 += 1
            pass
        CPU_cycles_94 += 1
        if R7_94 == R0_94: # Write BEQ $R7 $R0 done ; done looping?
            pass
        print("---------Element------ " + str(R3_94))
        print("Register Values " + "\tR0 - " + str(R0_94) + "\tR1 - " + str(R1_94) + "\tR2 - " + str(R2_94)
               + "\tR3 - " + str(R3_94)+ "\tR4 - " + str(R4_94)+ "\tR5 - " + str(R5_94)
               + "\tR7 - " + str(R7_94))
        print("Stalls pilled up: " + str(Stall_count_94))
        print("Clock cycles " + str(CPU_cycles_94))
```

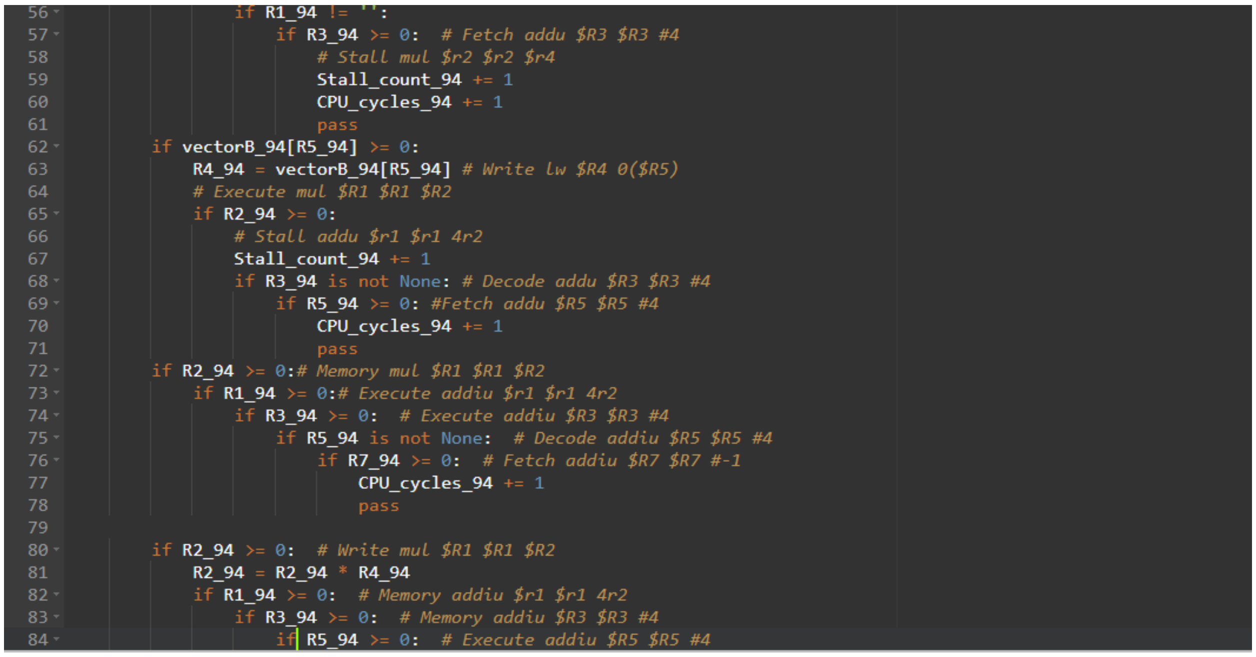
With Forward Chaining



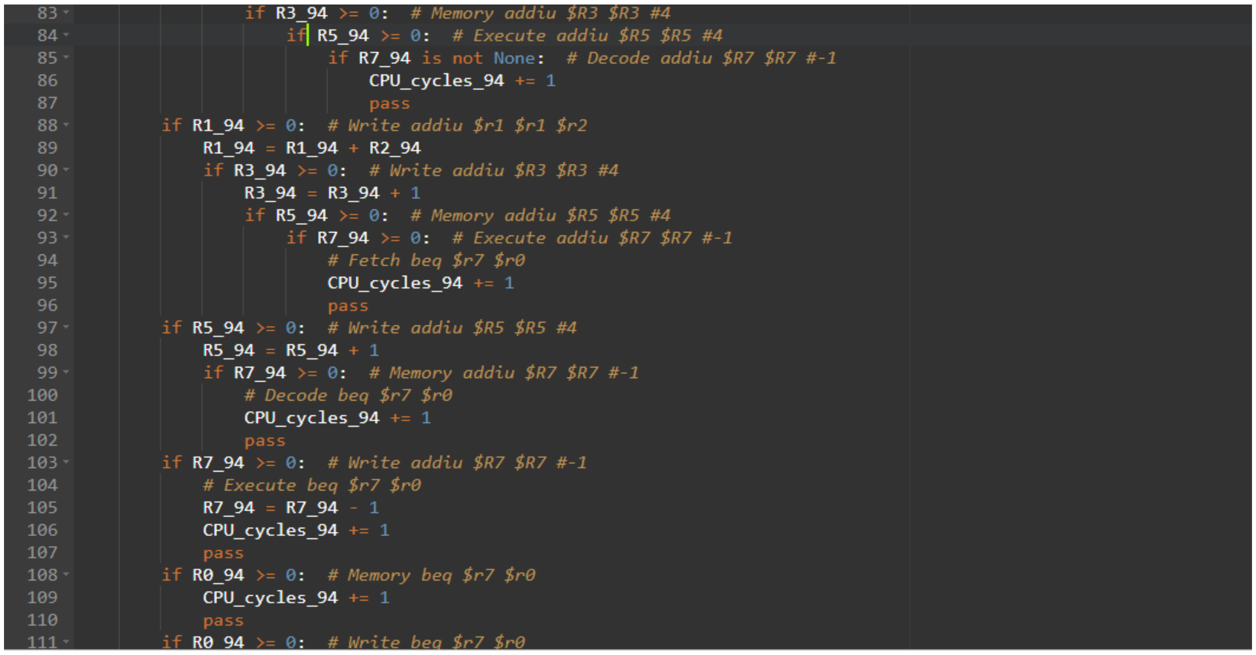
```python
import sys

print('-------------- DOT product simulation------------')

vectorA_94 = [0,1,6,0,0,6,0,9,4] #R2 - Initializing list with student ID: 016006094
vectorB_94 = [2,3,8,2,2,8,2,1,6] #R4 - Adding 2 to each elememnt and wrap around
print("vectorA_94: ",vectorA_94)
print("vectorB_94: ",vectorB_94)
#Intializing R0,R3,R5 registers
R0_94 = 0 #Hardwired register R0
R3_94 = 0 #Register used to access vectorA_94
R5_94 = 0 #Register used to access vectorB_94
R7_94 = len(vectorA_94) #Storing length of vectorA_94
Stall_count_94 = 0
CPU_cycles_94  = 0
R2_94 = 0
def done():
    global Stall_count_94
    print("-----------Results with forwarding----------")
    print("Dot product stored in R1: ", R1_94)
    print("Number of stalls " + str(Stall_count_94))
    print("Number of cycles " + str(CPU_cycles_94))
    sys.exit()
print("----------Start of Simulation------------")
R1_94 = R0_94  # addu $r1 $r0 $r0 #Resetting R1 register
CPU_cycles_94 += 1
```



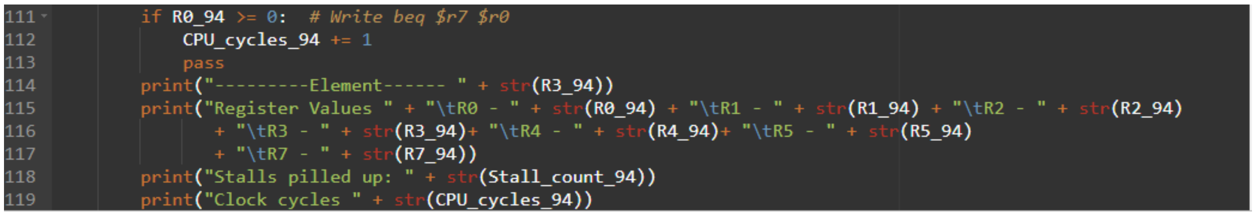
```python
while(1):
    if(R0_94==R7_94):
        done()
    else:
        if vectorA_94:   # Fetch lw $r2 0($r3)
            CPU_cycles_94 += 1
            pass
        if vectorA_94[R3_94]!= "": # Decode lw $r2 0($r3)
            if vectorB_94:      # Fetch lw $r4 0($r5)
                CPU_cycles_94 += 1
                pass
        if len(vectorA_94) >= 0: # Execute  lw $r2 0($r3)
            if vectorB_94[R5_94]!= "": # Decode lw $r2 0($r3)
                if R2_94 != None:  # Fetch mul $R2 $R2 $R4
                    CPU_cycles_94 += 1
                    pass
        if vectorA_94[R3_94] >= 0 : # Memory lw $R2 0($R3)
            if len(vectorB_94) >= 0:  # Execute lw $r4 0($r5)
                if R2_94 != '':  # Decode mul $R2 $R2 $R4
                    if R1_94 >= 0:  # Fetch addu $R1 $R1 $R2
                        CPU_cycles_94 += 1
                        pass

        if vectorA_94[R3_94] >= 0:
            R2_94 = vectorA_94[R3_94] # Write lw $R2 0($R3)
            if vectorB_94[R5_94] >= 0:  # Memory lw $R4 0($R5)
                # Decode addu $R1 $R1 $R2
                if R1_94 != '':
                    if R3_94 >= 0:  # Fetch addu $R3 $R3 #4
```



```
                if R1_94 != '':
                    if R3_94 >= 0:  # Fetch addu $R3 $R3 #4
                        # Stall mul $r2 $r2 $r4
                        Stall_count_94 += 1
                        CPU_cycles_94 += 1
                        pass
        if vectorB_94[R5_94] >= 0:
            R4_94 = vectorB_94[R5_94] # Write lw $R4 0($R5)
            # Execute mul $R1 $R1 $R2
            if R2_94 >= 0:
                # Stall addu $r1 $r1 4r2
                Stall_count_94 += 1
                if R3_94 is not None: # Decode addu $R3 $R3 #4
                    if R5_94 >= 0: #Fetch addu $R5 $R5 #4
                        CPU_cycles_94 += 1
                        pass
        if R2_94 >= 0:# Memory mul $R1 $R1 $R2
            if R1_94 >= 0:# Execute addiu $r1 $r1 4r2
                if R3_94 >= 0:  # Execute addiu $R3 $R3 #4
                    if R5_94 is not None:  # Decode addiu $R5 $R5 #4
                        if R7_94 >= 0:  # Fetch addiu $R7 $R7 #-1
                            CPU_cycles_94 += 1
                            pass

        if R2_94 >= 0:  # Write mul $R1 $R1 $R2
            R2_94 = R2_94 * R4_94
            if R1_94 >= 0:  # Memory addiu $r1 $r1 4r2
                if R3_94 >= 0:  # Memory addiu $R3 $R3 #4
                    if R5_94 >= 0:  # Execute addiu $R5 $R5 #4
```



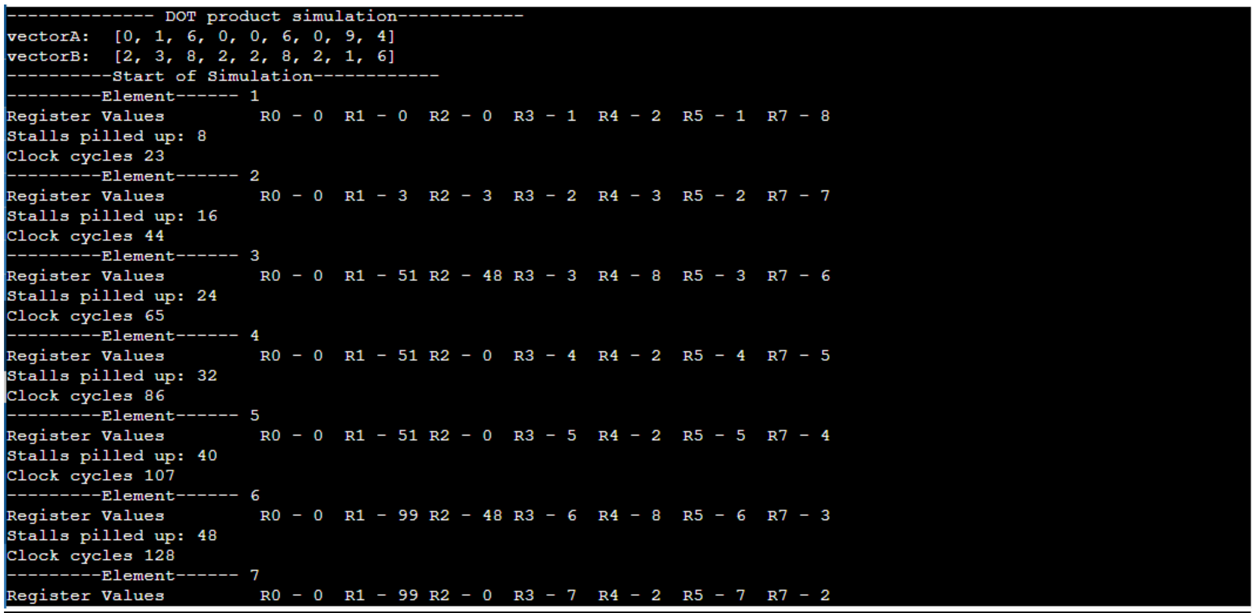
```
                if R3_94 >= 0:  # Memory addiu $R3 $R3 #4
                    if R5_94 >= 0:  # Execute addiu $R5 $R5 #4
                        if R7_94 is not None:  # Decode addiu $R7 $R7 #-1
                            CPU_cycles_94 += 1
                            pass
        if R1_94 >= 0:  # Write addiu $r1 $r1 $r2
            R1_94 = R1_94 + R2_94
            if R3_94 >= 0:  # Write addiu $R3 $R3 #4
                R3_94 = R3_94 + 1
                if R5_94 >= 0:  # Memory addiu $R5 $R5 #4
                    if R7_94 >= 0:  # Execute addiu $R7 $R7 #-1
                        # Fetch beq $r7 $r0
                        CPU_cycles_94 += 1
                        pass
        if R5_94 >= 0:  # Write addiu $R5 $R5 #4
            R5_94 = R5_94 + 1
            if R7_94 >= 0:  # Memory addiu $R7 $R7 #-1
                # Decode beq $r7 $r0
                CPU_cycles_94 += 1
                pass
        if R7_94 >= 0:  # Write addiu $R7 $R7 #-1
            # Execute beq $r7 $r0
            R7_94 = R7_94 - 1
            CPU_cycles_94 += 1
            pass
        if R0_94 >= 0:  # Memory beq $r7 $r0
            CPU_cycles_94 += 1
            pass
        if R0_94 >= 0:  # Write beq $r7 $r0
```



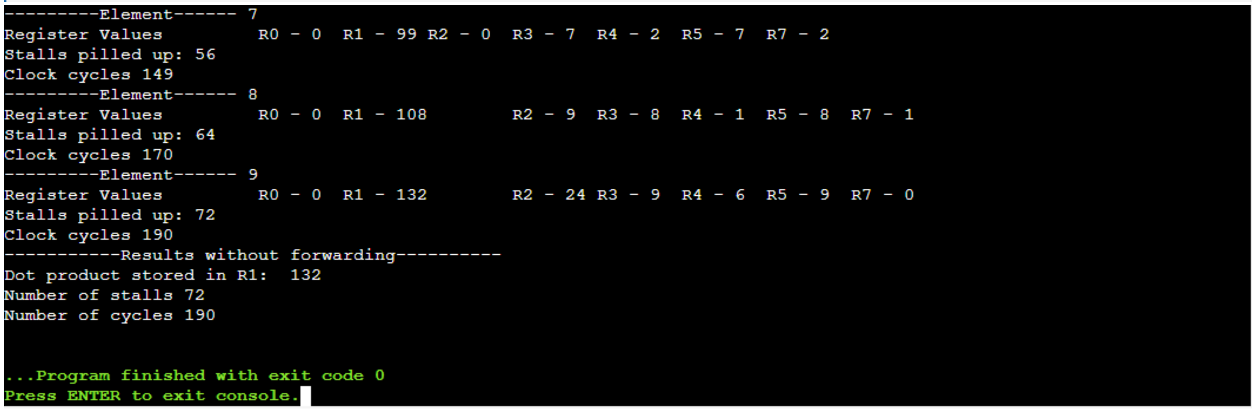
```
        if R0_94 >= 0:  # Write beq $r7 $r0
            CPU_cycles_94 += 1
            pass
        print("---------Element------ " + str(R3_94))
        print("Register Values " + "\tR0 - " + str(R0_94) + "\tR1 - " + str(R1_94) + "\tR2 - " + str(R2_94)
               + "\tR3 - " + str(R3_94)+ "\tR4 - " + str(R4_94)+ "\tR5 - " + str(R5_94)
               + "\tR7 - " + str(R7_94))
        print("Stalls pilled up: " + str(Stall_count_94))
        print("Clock cycles " + str(CPU_cycles_94))
```

Result:

Without Forward chaining result:



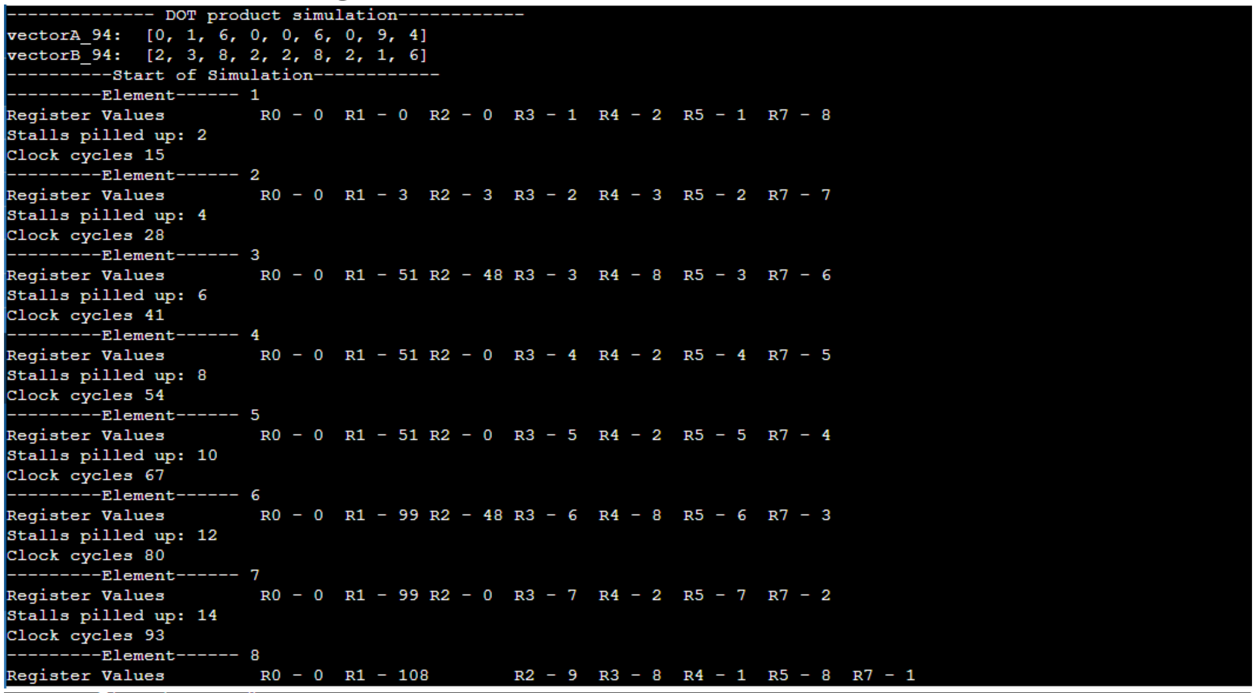
```
-------------- DOT product simulation------------
vectorA:  [0, 1, 6, 0, 0, 6, 0, 9, 4]
vectorB:  [2, 3, 8, 2, 2, 8, 2, 1, 6]
----------Start of Simulation------------
---------Element------ 1
Register Values         R0 - 0  R1 - 0  R2 - 0  R3 - 1  R4 - 2  R5 - 1  R7 - 8
Stalls pilled up: 8
Clock cycles 23
---------Element------ 2
Register Values         R0 - 0  R1 - 3  R2 - 3  R3 - 2  R4 - 3  R5 - 2  R7 - 7
Stalls pilled up: 16
Clock cycles 44
---------Element------ 3
Register Values         R0 - 0  R1 - 51 R2 - 48 R3 - 3  R4 - 8  R5 - 3  R7 - 6
Stalls pilled up: 24
Clock cycles 65
---------Element------ 4
Register Values         R0 - 0  R1 - 51 R2 - 0  R3 - 4  R4 - 2  R5 - 4  R7 - 5
Stalls pilled up: 32
Clock cycles 86
---------Element------ 5
Register Values         R0 - 0  R1 - 51 R2 - 0  R3 - 5  R4 - 2  R5 - 5  R7 - 4
Stalls pilled up: 40
Clock cycles 107
---------Element------ 6
Register Values         R0 - 0  R1 - 99 R2 - 48 R3 - 6  R4 - 8  R5 - 6  R7 - 3
Stalls pilled up: 48
Clock cycles 128
---------Element------ 7
Register Values         R0 - 0  R1 - 99 R2 - 0  R3 - 7  R4 - 2  R5 - 7  R7 - 2
```



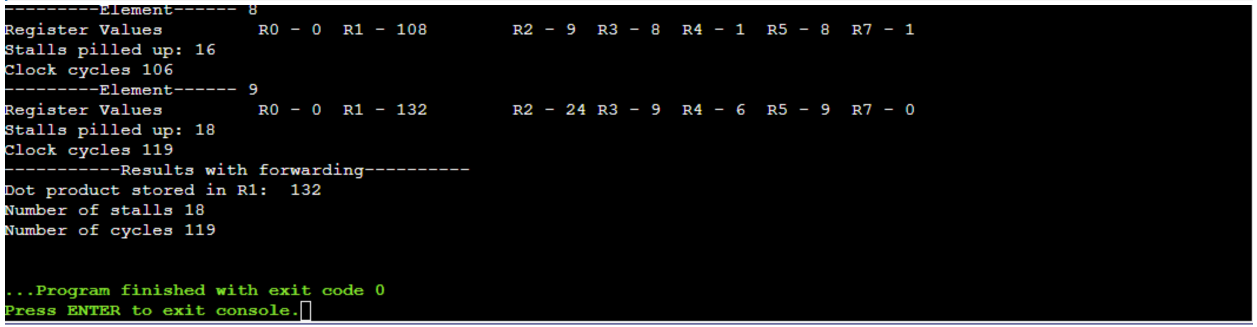
```
---------Element------ 7
Register Values         R0 - 0  R1 - 99 R2 - 0  R3 - 7  R4 - 2  R5 - 7  R7 - 2
```

```
Stalls pilled up: 56
Clock cycles 149
---------Element------ 8
Register Values         R0 - 0  R1 - 108        R2 - 9  R3 - 8  R4 - 1  R5 - 8  R7 - 1
Stalls pilled up: 64
Clock cycles 170
---------Element------ 9
Register Values         R0 - 0  R1 - 132        R2 - 24 R3 - 9  R4 - 6  R5 - 9  R7 - 0
Stalls pilled up: 72
Clock cycles 190
-----------Results without forwarding----------
Dot product stored in R1:  132
Number of stalls 72
Number of cycles 190


...Program finished with exit code 0
Press ENTER to exit console.
```

With Forward chaining result:



```
------------- DOT product simulation-----------
vectorA_94:  [0, 1, 6, 0, 0, 6, 0, 9, 4]
vectorB_94:  [2, 3, 8, 2, 2, 8, 2, 1, 6]
----------Start of Simulation-----------
---------Element------ 1
Register Values         R0 - 0  R1 - 0  R2 - 0  R3 - 1  R4 - 2  R5 - 1  R7 - 8
Stalls pilled up: 2
Clock cycles 15
---------Element------ 2
Register Values         R0 - 0  R1 - 3  R2 - 3  R3 - 2  R4 - 3  R5 - 2  R7 - 7
Stalls pilled up: 4
Clock cycles 28
---------Element------ 3
Register Values         R0 - 0  R1 - 51 R2 - 48 R3 - 3  R4 - 8  R5 - 3  R7 - 6
Stalls pilled up: 6
Clock cycles 41
---------Element------ 4
Register Values         R0 - 0  R1 - 51 R2 - 0  R3 - 4  R4 - 2  R5 - 4  R7 - 5
Stalls pilled up: 8
Clock cycles 54
---------Element------ 5
Register Values         R0 - 0  R1 - 51 R2 - 0  R3 - 5  R4 - 2  R5 - 5  R7 - 4
Stalls pilled up: 10
Clock cycles 67
---------Element------ 6
Register Values         R0 - 0  R1 - 99 R2 - 48 R3 - 6  R4 - 8  R5 - 6  R7 - 3
Stalls pilled up: 12
Clock cycles 80
---------Element------ 7
Register Values         R0 - 0  R1 - 99 R2 - 0  R3 - 7  R4 - 2  R5 - 7  R7 - 2
Stalls pilled up: 14
Clock cycles 93
---------Element------ 8
Register Values         R0 - 0  R1 - 108        R2 - 9  R3 - 8  R4 - 1  R5 - 8  R7 - 1
```



```
---------Element------ 8
Register Values         R0 - 0  R1 - 108        R2 - 9  R3 - 8  R4 - 1  R5 - 8  R7 - 1
```

```
Stalls pilled up: 16
Clock cycles 106
---------Element------ 9
Register Values         R0 - 0  R1 - 132        R2 - 24 R3 - 9  R4 - 6  R5 - 9  R7 - 0
Stalls pilled up: 18
Clock cycles 119
----------Results with forwarding---------
Dot product stored in R1:  132
Number of stalls 18
Number of cycles 119


...Program finished with exit code 0
Press ENTER to exit console.
```

Conclusion:

Dot product with forward chaining reduces around 71 cycles. So that can improve the performance of the CPU using forward chaining which helps to reduce number of CPU cycles.