

Dot product

4/3/2022

Amuylar.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 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 ($EX/MEM.RegisterRd \neq 0$) and ($EX/MEM.RegisterRd = ID/EX.RegisterRs$)) ForwardA = 10

if ($EX/MEM.RegWrite$ and ($EX/MEM.RegisterRd \neq 0$) and ($EX/MEM.RegisterRd = ID/EX.RegisterRt$)) ForwardB = 10

2. *MEM hazard:*

if ($MEM/WB.RegWrite$ and ($MEM/WB.RegisterRd \neq 0$) and ($MEM/WB.RegisterRd = ID/EX.RegisterRs$)) ForwardA = 01

if ($MEM/WB.RegWrite$ and ($MEM/WB.RegisterRd \neq 0$) and ($MEM/WB.RegisterRd = ID/EX.RegisterRt$)) ForwardB = 01[1]

Code:

Without Forward Chaining

```
1 import sys
2 print('----- DOT product simulation-----')
3 vectorA_94 = [0,1,6,0,0,6,0,9,4] #R2 - Initializing list with student ID: 016006094
4 vectorB_94 = [2,3,8,2,2,8,2,1,6] #R4 - Adding 2 to each elememnt and wrap around
5
6 print("vectorA: ",vectorA_94)
7 print("vectorB: ",vectorB_94)
8 #Intializing R0,R3,R5 registers
9 R0_94 = 0 #Hardwired register R0
10 R3_94 = 0 #Register used to access vectorA
11 R5_94 = 0 #Register used to access vectorB
12 R7_94 = len(vectorA_94) #Storing length of vectorA
13 Stall_count_94 = 0
14 CPU_cycles_94 = 0
15 R2_94 = 0
16 def done():
17     global Stall_count_94
18     print("-----Results without forwarding-----")
19     print("Dot product stored in R1: ", R1_94)
20     print("Number of stalls " + str(Stall_count_94))
21     print("Number of cycles " + str(CPU_cycles_94))
22     sys.exit()
23 print("-----Start of Simulation-----")
24 R1_94 = R0_94 # addu $r1 $r0 $r0 #Resetting R1 register
25 CPU_cycles_94 += 1
26 if R7_94>=0: # Fetch beq $R7 $R0 done first time
```

```
25 CPU_cycles_94 += 1
26 if R7_94>=0: # Fetch beq $R7 $R0 done first time
27     CPU_cycles_94 += 1
28 while(1):
29     if(R0_94==R7_94):
30         done()
31     else:
32         if vectorA_94: # Fetch lw $r2 0($r3)
33             CPU_cycles_94 += 1
34             pass
35         if vectorA_94[R3_94]!= "": # Decode lw $r2 0($r3)
36             if vectorB_94: # Fetch lw $r4 0($r5)
37                 CPU_cycles_94 += 1
38                 pass
39             if len(vectorA_94) >= 0: # Execute lw $r2 0($r3)
40                 if vectorB_94[R5_94]!= "": # Decode lw $r2 0($r3)
41                     if R2_94 != None: # Fetch mul $R2 $R2 $R4
42                         CPU_cycles_94 += 1
43                         pass
44                     if vectorA_94[R3_94] >= 0 : # Memory lw $R2 0($R3)
45                         if len(vectorB_94) >= 0: # Execute lw $r4 0($r5)
46                             Stall_count_94 += 1 # Stall mul $r2 $r2 $r4
47                             CPU_cycles_94 += 1
48                             pass
49                     if vectorA_94[R3_94] >= 0:
50                         R2_94 = vectorA_94[R3_94] # Write lw $R2 0($R3)
```

```

50     R2_94 = vectorA_94[R3_94] # Write Lw $R2 0($R3)
51     if vectorB_94[R5_94] >= 0: # Memory Lw $R4 0($R5)
52         # Stall mul $r2 $r2 $r4
53         CPU_cycles_94 += 1
54         Stall_count_94 += 1
55         pass
56     if vectorB_94[R5_94] >= 0:
57         R4_94 = vectorB_94[R5_94] # Write Lw $R4 0($R5)
58         # Stall mul $r2 $r2 $r4
59         CPU_cycles_94 += 1
60         Stall_count_94 += 1
61         pass
62     if R2_94 != '': # Decode mul $R2 $R2 $R4
63         # Fetch addu $R1 $R1 $R2
64         if R1_94 >= 0:
65             CPU_cycles_94 += 1
66             pass
67
68     if len(vectorA_94) >= 0: # Execute mul $R2 $R2 $R4
69         # Stall addu $R1 $R1 $R2
70         CPU_cycles_94 += 1
71         Stall_count_94 += 1
72     if R2_94 >= 0: # Memory mul $R2 $R2 $R4
73         # Stall addu $R1 $R1 $R2
74         Stall_count_94 += 1
75         CPU_cycles_94 += 1
76         Stall_count_94 += 1
77         CPU_cycles_94 += 1
78     if R2_94 >= 0: # Write mul $R2 $R2 $R4
79         R2_94 = R2_94 * R4_94
80         # Stall addu $R1 $R1 $R2
81         Stall_count_94 += 1
82         CPU_cycles_94 += 1
83     if R1_94 != '': # Decode addu $R1 $R1 $R2
84         if R3_94 >= 0: # Fetch addu $R3 $R3 $R4
85             CPU_cycles_94 += 1
86             pass
87     if R1_94 >= 0: # Execute addu $R1 $R1 $R2
88         if R3_94 != '': # Decode addiu $R3 $R3 #4
89             if R5_94 >= 0: # Fetch addiu $R5 $R5 #4
90                 CPU_cycles_94 += 1
91                 pass
92     if R1_94 >= 0: # Memory addu $R1 $R1 $R2
93         if R3_94 >= 0: # Execute addiu $R3 $R3 #4
94             if R5_94 != '': # Decode addiu $R5 $R5 #4
95                 if R7_94 >= 0: # Fetch addiu $R7 $R7 #-1
96                     CPU_cycles_94 += 1
97                     pass
98     if R1_94 >= 0: # Write addu $R1 $R1 $R2
99         R1_94 = R1_94 + R2_94
100     if R3_94 >= 0: # Memory addiu $R3 $R3 #4

```

```

99 ~         if R3_94 >= 0: # Memory addiu $R3 $R3 #4
100 ~             if R5_94 >= 0: # Execute addiu $R5 $R5 #4
101 ~                 if R7_94 != '': # Decode addiu $R7 $R7 #-1
102 ~                     CPU_cycles_94 += 1
103 ~                     pass
104 ~         if R3_94 >= 0: # Write addiu $R3 $R3 #4
105 ~             R3_94 = R3_94 + 1
106 ~             if R5_94 >= 0: # Memory addiu $R5 $R5 #4
107 ~                 if R7_94 >= 0: # Execute addiu $R7 $R7 #-1
108 ~                     CPU_cycles_94 += 1
109 ~                     pass # Fetch beq $R7 $R0 done
110 ~         if R5_94 >= 0: # Write addiu $R5 $R5 #4
111 ~             R5_94 = R5_94 + 1
112 ~             if R7_94 >= 0: # Memory addiu $R7 $R7 #-1
113 ~                 # Stall beq $R7 $R0 done
114 ~                 CPU_cycles_94 += 1
115 ~                 Stall_count_94 += 1
116 ~         if R7_94 >= 0: # Write addiu $R7 $R7 #-1
117 ~             R7_94 = R7_94 - 1
118 ~             # Stall beq $R7 $R0 done
119 ~             CPU_cycles_94 += 1
120 ~             Stall_count_94 += 1
121 ~         if R7_94 != '': # Decode beq $R7 $R0 done ; done Looping?
122 ~             CPU_cycles_94 += 1
123 ~             pass
124 ~         if R7_94 >= 0: # Execute BEQ $R7 $R0 done ; done Looping?
125 ~             CPU_cycles_94 += 1
126 ~             pass
127 ~
128 ~         if R7_94: # Memory BEQ $R7 $R0 done ; done Looping?
129 ~             CPU_cycles_94 += 1
130 ~             pass
131 ~         CPU_cycles_94 += 1
132 ~         if R7_94 == R0_94: # Write BEQ $R7 $R0 done ; done Looping?
133 ~             pass
134 ~         print("-----Element----- " + str(R3_94))
135 ~         print("Register Values " + "\tR0 - " + str(R0_94) + "\tR1 - " + str(R1_94) + "\tR2 - " + str(R2_94)
136 ~             + "\tR3 - " + str(R3_94) + "\tR4 - " + str(R4_94) + "\tR5 - " + str(R5_94)
137 ~             + "\tR7 - " + str(R7_94))
138 ~         print("Stalls piled up: " + str(Stall_count_94))
139 ~         print("Clock cycles " + str(CPU_cycles_94))

```

With Forward Chaining

```
1 import sys
2
3 print('----- DOT product simulation-----')
4
5 vectorA_94 = [0,1,6,0,0,6,0,9,4] #R2 - Initializing list with student ID: 016006094
6 vectorB_94 = [2,3,8,2,2,8,2,1,6] #R4 - Adding 2 to each elememnt and wrap around
7 print("vectorA_94: ",vectorA_94)
8 print("vectorB_94: ",vectorB_94)
9 #Intializing R0,R3,R5 registers
10 R0_94 = 0 #Hardwired register R0
11 R3_94 = 0 #Register used to access vectorA_94
12 R5_94 = 0 #Register used to access vectorB_94
13 R7_94 = len(vectorA_94) #Storing length of vectorA_94
14 Stall_count_94 = 0
15 CPU_cycles_94 = 0
16 R2_94 = 0
17 def done():
18     global Stall_count_94
19     print("-----Results with forwarding-----")
20     print("Dot product stored in R1: ", R1_94)
21     print("Number of stalls " + str(Stall_count_94))
22     print("Number of cycles " + str(CPU_cycles_94))
23     sys.exit()
24 print("-----Start of Simulation-----")
25 R1_94 = R0_94 # addu $r1 $r0 $r0 #Resetting R1 register
26 CPU_cycles_94 += 1
```

```
29 while(1):
30     if(R0_94==R7_94):
31         done()
32     else:
33         if vectorA_94: # Fetch Lw $r2 0($r3)
34             CPU_cycles_94 += 1
35             pass
36         if vectorA_94[R3_94] != "": # Decode Lw $r2 0($r3)
37             if vectorB_94: # Fetch Lw $r4 0($r5)
38                 CPU_cycles_94 += 1
39                 pass
40             if len(vectorA_94) >= 0: # Execute Lw $r2 0($r3)
41                 if vectorB_94[R5_94] != "": # Decode Lw $r2 0($r3)
42                     if R2_94 != None: # Fetch mul $R2 $R2 $R4
43                         CPU_cycles_94 += 1
44                         pass
45                     if vectorA_94[R3_94] >= 0: # Memory Lw $R2 0($R3)
46                         if len(vectorB_94) >= 0: # Execute Lw $r4 0($r5)
47                             if R2_94 != '': # Decode mul $R2 $R2 $R4
48                                 if R1_94 >= 0: # Fetch addu $R1 $R1 $R2
49                                     CPU_cycles_94 += 1
50                                     pass
51                             if vectorA_94[R3_94] >= 0:
52                                 R2_94 = vectorA_94[R3_94] # Write Lw $R2 0($R3)
53                                 if vectorB_94[R5_94] >= 0: # Memory Lw $R4 0($R5)
54                                     # Decode addu $R1 $R1 $R2
55                                     if R1_94 != '':
56                                         if R3_94 >= 0: # Fetch addu $R3 $R3 #4
```

```

56~         if R1_94 != '':
57~             if R3_94 >= 0: # Fetch addu $R3 $R3 #4
58~                 # Stall mul $r2 $r2 $r4
59~                 Stall_count_94 += 1
60~                 CPU_cycles_94 += 1
61~                 pass
62~         if vectorB_94[R5_94] >= 0:
63~             R4_94 = vectorB_94[R5_94] # Write Lw $R4 0($R5)
64~             # Execute mul $R1 $R1 $R2
65~             if R2_94 >= 0:
66~                 # Stall addu $r1 $r1 4r2
67~                 Stall_count_94 += 1
68~             if R3_94 is not None: # Decode addu $R3 $R3 #4
69~                 if R5_94 >= 0: #Fetch addu $R5 $R5 #4
70~                     CPU_cycles_94 += 1
71~                     pass
72~         if R2_94 >= 0: # Memory mul $R1 $R1 $R2
73~             if R1_94 >= 0: # Execute addiu $r1 $r1 4r2
74~                 if R3_94 >= 0: # Execute addiu $R3 $R3 #4
75~                     if R5_94 is not None: # Decode addiu $R5 $R5 #4
76~                         if R7_94 >= 0: # Fetch addiu $R7 $R7 #-1
77~                             CPU_cycles_94 += 1
78~                             pass
79~
80~         if R2_94 >= 0: # Write mul $R1 $R1 $R2
81~             R2_94 = R2_94 * R4_94
82~             if R1_94 >= 0: # Memory addiu $r1 $r1 4r2
83~                 if R3_94 >= 0: # Memory addiu $R3 $R3 #4
84~                     if R5_94 >= 0: # Execute addiu $R5 $R5 #4

```

```

83~         if R3_94 >= 0: # Memory addiu $R3 $R3 #4
84~             if R5_94 >= 0: # Execute addiu $R5 $R5 #4
85~                 if R7_94 is not None: # Decode addiu $R7 $R7 #-1
86~                     CPU_cycles_94 += 1
87~                     pass
88~         if R1_94 >= 0: # Write addiu $r1 $r1 $r2
89~             R1_94 = R1_94 + R2_94
90~             if R3_94 >= 0: # Write addiu $R3 $R3 #4
91~                 R3_94 = R3_94 + 1
92~             if R5_94 >= 0: # Memory addiu $R5 $R5 #4
93~                 if R7_94 >= 0: # Execute addiu $R7 $R7 #-1
94~                     # Fetch beq $r7 $r0
95~                     CPU_cycles_94 += 1
96~                     pass
97~         if R5_94 >= 0: # Write addiu $R5 $R5 #4
98~             R5_94 = R5_94 + 1
99~             if R7_94 >= 0: # Memory addiu $R7 $R7 #-1
100~                 # Decode beq $r7 $r0
101~                 CPU_cycles_94 += 1
102~                 pass
103~         if R7_94 >= 0: # Write addiu $R7 $R7 #-1
104~             # Execute beq $r7 $r0
105~             R7_94 = R7_94 - 1
106~             CPU_cycles_94 += 1
107~             pass
108~         if R0_94 >= 0: # Memory beq $r7 $r0
109~             CPU_cycles_94 += 1
110~             pass
111~         if R0_94 >= 0: # Write beq $r7 $r0

```

```

111~         if R0_94 >= 0: # Write beq $r7 $r0
112~             CPU_cycles_94 += 1
113~             pass
114~         print("-----Element----- " + str(R3_94))
115~         print("Register Values " + "\tR0 - " + str(R0_94) + "\tR1 - " + str(R1_94) + "\tR2 - " + str(R2_94)
116~             + "\tR3 - " + str(R3_94) + "\tR4 - " + str(R4_94) + "\tR5 - " + str(R5_94)
117~             + "\tR7 - " + str(R7_94))
118~         print("Stalls piled up: " + str(Stall_count_94))
119~         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 piled up: 8
Clock cycles 23
-----Element----- 2
Register Values      R0 - 0  R1 - 3  R2 - 3  R3 - 2  R4 - 3  R5 - 2  R7 - 7
Stalls piled up: 16
Clock cycles 44
-----Element----- 3
Register Values      R0 - 0  R1 - 51 R2 - 48 R3 - 3  R4 - 8  R5 - 3  R7 - 6
Stalls piled up: 24
Clock cycles 65
-----Element----- 4
Register Values      R0 - 0  R1 - 51 R2 - 0  R3 - 4  R4 - 2  R5 - 4  R7 - 5
Stalls piled up: 32
Clock cycles 86
-----Element----- 5
Register Values      R0 - 0  R1 - 51 R2 - 0  R3 - 5  R4 - 2  R5 - 5  R7 - 4
Stalls piled up: 40
Clock cycles 107
-----Element----- 6
Register Values      R0 - 0  R1 - 99 R2 - 48 R3 - 6  R4 - 8  R5 - 6  R7 - 3
Stalls piled 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 piled up: 56
Clock cycles 149
-----Element----- 8
Register Values      R0 - 0  R1 - 108      R2 - 9  R3 - 8  R4 - 1  R5 - 8  R7 - 1
Stalls piled up: 64
Clock cycles 170
-----Element----- 9
Register Values      R0 - 0  R1 - 132      R2 - 24 R3 - 9  R4 - 6  R5 - 9  R7 - 0
Stalls piled 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 piled up: 2
Clock cycles 15
-----Element----- 2
Register Values      R0 - 0  R1 - 3  R2 - 3  R3 - 2  R4 - 3  R5 - 2  R7 - 7
Stalls piled up: 4
Clock cycles 28
-----Element----- 3
Register Values      R0 - 0  R1 - 51 R2 - 48 R3 - 3  R4 - 8  R5 - 3  R7 - 6
Stalls piled up: 6
Clock cycles 41
-----Element----- 4
Register Values      R0 - 0  R1 - 51 R2 - 0  R3 - 4  R4 - 2  R5 - 4  R7 - 5
Stalls piled up: 8
Clock cycles 54
-----Element----- 5
Register Values      R0 - 0  R1 - 51 R2 - 0  R3 - 5  R4 - 2  R5 - 5  R7 - 4
Stalls piled up: 10
Clock cycles 67
-----Element----- 6
Register Values      R0 - 0  R1 - 99 R2 - 48 R3 - 6  R4 - 8  R5 - 6  R7 - 3
Stalls piled up: 12
Clock cycles 80
-----Element----- 7
Register Values      R0 - 0  R1 - 99 R2 - 0  R3 - 7  R4 - 2  R5 - 7  R7 - 2
Stalls piled 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 piled up: 16
Clock cycles 106
-----Element----- 9
Register Values      R0 - 0  R1 - 132      R2 - 24 R3 - 9  R4 - 6  R5 - 9  R7 - 0
Stalls piled 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.