Assignment 3

4.10.1)

The total execution time for the program would have to be:

There are 5 commands

1. sw r16,12(r6)
2. lw r16,8(r6)
3. beq r5,r4,Label # Assume r5!=r4
4. add r5,r1,r4
5. slt r5,r15,r4

each stage takes 1 cycle so 1 command takes 5 cycles:   
(1 for IF, 1 for ID, 1 for EX, 1 for MEM, and 1 for Write).

If we load the first command in cycle 1 and then the 2nd command in cycle 2, the third in stage 3 (etc…) , it would optimally take 9 cycles. It cannot however because stage 4 and stage 5 are accessing the memory with IF at the same time as stage 1 and 2 with MEM, so the machine must stall these stages. Making the total execution time 11 cycles, or 2010 ps.

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| OPERATION | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| Sw\_r16,12(r6) | IF | ID | EXE | MEM | WB |  |  |  |  |  |  |
| Lw\_r16,8,(r6) |  | IF | ID | EXE | MEM | WB |  |  |  |  |  |
| BEQ r5,r4,lbl |  |  | IF | ID | EXE | MEM | WB |  |  |  |  |
| Add r5,r1,r4 |  |  |  | # | # | IF | ID | EXE | MEM | WB |  |
| Slt r5, r15, r4 |  |  |  |  |  |  | IF | ID | EXE | MEM | WB |
| Latency (ps) | 200 | 200 | 200 | 190 | 190 | 200 | 200 | 150 | 190 | 190 | 100 |

It has to do this because as displayed in this chart here beq doesn’t need data access so the memory is free for command 4 and 5 to start.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| LW | 200 | 120 | 150 | 190 | 100 |
| SW | 200 | 120 | 150 | 190 |  |
| ARITHMETIC | 200 | 120 | 150 |  | 100 |
| BRANCH | 200 | 120 | 150 |  |  |

You cannot eliminate this structure hazard with NOPs like you could a data hazard because NOPs are still fetch functions, you still have to get the machine to go to the memory to have it return the NOP. You would need a failsafe on the hardware side to be able to stop this structural hazard.

4.10.2)

The code changed to accommodate the 4 stage pipeline would be:

sw r16,12(r6)

lw r16,8(r6)

beq r5,r4,Label # Assume r5!=r4

add r5,r1,r4

slt r5,r15,r4

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| LW | 200 | 120 |  | 190 | 100 |
| SW | 200 | 120 |  | 190 |  |
| ARITHMETIC | 200 | 120 | 150 |  | 100 |
| BRANCH | 200 | 120 | 150 |  |  |

The change keeping the code the same only saves 1 cycle so instead of 5 stages there are 4 so instead of

this: (2010 execution time in ps)

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| OPERATION | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| Sw\_r16,12(r6) | IF | ID | EXE | MEM | WB |  |  |  |  |  |  |
| Lw\_r16,8,(r6) |  | IF | ID | EXE | MEM | WB |  |  |  |  |  |
| BEQ r5,r4,lbl |  |  | IF | ID | EXE | MEM | WB |  |  |  |  |
| Add r5,r1,r4 |  |  |  | # | # | IF | ID | EXE | MEM | WB |  |
| Slt r5, r15, r4 |  |  |  |  |  |  | IF | ID | EXE | MEM | WB |
| Latency (ps) | 200 | 200 | 200 | 190 | 190 | 200 | 200 | 150 | 190 | 190 | 100 |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| OPERATION | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| Sw\_r16,12(r6) | IF | ID | MEM | WB |  |  |  |  |  |  |
| Lw\_r16,8,(r6) |  | IF | ID | MEM | WB |  |  |  |  |  |
| BEQ r5,r4,lbl |  |  | # | # | IF | ID | EXE | WB |  |  |
| Add r5,r1,r4 |  |  |  |  |  | IF | ID | EXE | WB |  |
| Slt r5, r15, r4 |  |  |  |  |  |  | IF | ID | EXE | WB |
| Latency (ps) | 200 | 200 | 190 | 190 | 200 | 200 | 200 | 150 | 150 | 100 |

You get this: (1780 execution time in ps)

In this case there are 5 instructions executed(a).  
cycles with n-1 stages == (n-1) + a = x(old) == (5-1) + 5 = 9  
cycles with n-1 stages == (n-1) + a = y(new) == (4-1) + 5 = 8  
x(old)/y(new) = z (speedup)  
z = 1.125  
old/new = 1.129

The speedup by this change would be about 1.13 ps

4.10.3

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| OPERATION | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| Sw\_r16,12(r6) | IF | ID | EXE | MEM | WB |  |  |  |  |  |
| Lw\_r16,8,(r6) |  | IF | ID | EXE | MEM | WB |  |  |  |  |
| BEQ r5,r4,lbl |  |  | IF | ID | EXE | MEM | WB |  |  |  |
| Add r5,r1,r4 |  |  |  | # | IF | ID | EXE | MEM | WB |  |
| Slt r5, r15, r4 |  |  |  |  |  | IF | ID | EXE | MEM | WB |
| Latency (ps) | 200 | 200 | 200 | 190 | 200 | 200 | 150 | 190 | 190 | 100 |

If branch prediction is changed to occur in the ID stage it would allow all following operations to move forward by 1 cycle. Instead of having to stall 2 times it would only stall 1 (unless we have perfect branch prediction in which case we would have 0 stalls – However the question mentions stall on branch so we are operating on 1 stall if executed in the id stage).

The formula for this would be 4 + instructions + (branches executed\*stalls) = z(execution time)

4 + 5 +(1\*2) = 11(old)  
4 + 5 +(1\*1) = 10(new)

Old/new = 11/10 = 1.10

The speedup would be about 1.10 (10%)

4.10.4

(using info from question 4.10.2)

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| OPERATION | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| Sw\_r16,12(r6) | IF | ID | EXE | MEM | WB |  |  |  |  |  |  |
| Lw\_r16,8,(r6) |  | IF | ID | EXE | MEM | WB |  |  |  |  |  |
| BEQ r5,r4,lbl |  |  | IF | ID | EXE | MEM | WB |  |  |  |  |
| Add r5,r1,r4 |  |  |  | # | # | IF | ID | EXE | MEM | WB |  |
| Slt r5, r15, r4 |  |  |  |  |  |  | IF | ID | EXE | MEM | WB |
| Latency (ps) | 200 | 200 | 200 | 190 | 190 | 200 | 200 | 150 | 190 | 190 | 100 |

cycle time with 5 stages longest = IF (200)  
cycle time with 4 stages longest = MEM (190 || 150) + 20(for work that cannot be done in parallel)  
 = MEM (210)

we know from question 2 that:  
cycles with n-1 stages == (n-1) + a = x(old) == (5-1) + 5 = 9  
cycles with n-1 stages == (n-1) + a = y(new) == (4-1) + 5 = 8  
x(old)/y(new) = z (speedup)

cycles \* longest time = (x)old  
cycles \* longest time = (y)new  
(9\*200)/(8\*210) = z (speedup)  
1800ps/1680ps = z = 1.07

the speedup would be 120ps which is a speedup of about 1.07%

4.10.5

(Using info from question 4.10.3)

A 50% increase in ID and a 10ps reduction in exe would result in this:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Longest | 1 | 3 | 4 | 2 | 5 |
| LW | 200 | 120\*1.5=180 |  | 190 | 100 |
| SW | 200 | 120\*1.5=180 |  | 190 |  |
| ARITHMETIC | 200 | 120\*1.5=180 | 140 |  | 100 |
| BRANCH | 200 | 120\*1.5=180 | 140 |  |  |

(Copied from 4.10.3)

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| OPERATION | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| Sw\_r16,12(r6) | IF | ID | EXE | MEM | WB |  |  |  |  |  |
| Lw\_r16,8,(r6) |  | IF | ID | EXE | MEM | WB |  |  |  |  |
| BEQ r5,r4,lbl |  |  | IF | ID | EXE | MEM | WB |  |  |  |
| Add r5,r1,r4 |  |  |  | # | IF | ID | EXE | MEM | WB |  |
| Slt r5, r15, r4 |  |  |  |  |  | IF | ID | EXE | MEM | WB |
| Latency (ps) | 200 | 200 | 200 | 190 | 200 | 200 | 180 | 190 | 190 | 100 |

In the cycle times the precedence is given to the longest operation being in this order (IF (200), MEM(190), ID(180), EXE(140), and WB(100)) so the change in EXE and ID only matter is only relevant if it is higher than 200, with the change neither is greater than 200.

The formula for this would be 4 + instructions + (branches executed\*stalls) = z(execution time)

4 + 5 +(1\*2) = 11(old)  
4 + 5 +(1\*1) = 10(new)

Execution time \* longest cycle = z(old)  
Execution time \* longest cycle = y(new)

(11\*200)ps/(10\*200)ps = 1.10%

The speed would be 2200ps/2000ps = there is a 1.10% speedup if the branch resolution is moved from exe to id

4.10.6

(using information from 4.10.3)

A 50% increase in ID and a 10ps reduction in exe would result in this:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Longest | 1 | 3 | 4 | 2 | 5 |
| LW | 200 | 120 |  | 190 | 100 |
| SW | 200 | 120 |  | 190 |  |
| ARITHMETIC | 200 | 120 | 150 – 20 =130 |  | 100 |
| BRANCH | 200 | 120 | 150 – 20 = 130 |  |  |

Assuming there is 1 stall on branch(as there was in 4.10.3)

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| OPERATION | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| Sw\_r16,12(r6) | IF | ID | EXE | MEM | WB |  |  |  |  |  |  |  |
| Lw\_r16,8,(r6) |  | IF | ID | EXE | MEM | WB |  |  |  |  |  |  |
| BEQ r5,r4,lbl |  |  | # | # | IF | ID | EXE | MEM | WB |  |  |  |
| Add r5,r1,r4 |  |  |  |  |  | # | IF | ID | EXE | MEM | WB |  |
| Slt r5, r15, r4 |  |  |  |  |  |  |  | IF | ID | EXE | MEM | WB |
| Latency (ps) | 200 | 200 | 130 | 190 | 200 | 120 | 200 | 200 | 130 | 190 | 190 | 100 |

Using the same formula from (4.10.2)

If branch prediction is changed it would not change the cycle time because the longest cycle is the IF stage (200ps) regardless of the times of the other cycles it will always take 200ps. to occur in the MEM stage it would force all following operations to move backwards by 1 cycle. Instead of having to stall 2 times it would stall 3 (unless we have perfect branch prediction in which case we would have 0 stalls – However the question mentions stall on branch so we are operating on 3 stall if executed in the MEM stage).

The formula for this would be 4 + instructions + (branches executed\*stalls) = z (execution time)

4 + 5 +(1\*2) = 11(old)  
4 + 5 +(1\*3) = 12(new)

Old/new = 11/12 = 0.916667

11\*200ps = 2200ps (execution time in EX stage)  
12\*200ps = 2400ps (execution time in MEM stage)

2400/2200 = 0.916667

The speedup would be about 0.92%  
which would mean that it is slowing down as a result of the extra stall cycle it has to put in to accommodate the branches being executed in the MEM stage.