A screenshot of a cell phone

Description automatically generated

I1: add $t1, $t2, $t3  
I2: sw $t1, 12($t0)  
I3: add $t4, $t1, $t2  
I4: sw $t4, 16($t0)  
I5: add $t2, $t3, $t5

**Data Hazards**  
I1 -> I2 ($t1) Read After Write Data Hazard  
I1 -> I3 ($t1) Read After Write Data Hazard  
I3 -> I4 ($t4) Read After Write Data Hazard

**Re-arranging the Assembly code**  
I1: add $t1, $t2, $t3  
I2: add $t4, $t1, $t2  
I3: add $t2, $t3, $t5  
I4: sw $t1, 12($t0)  
I5: sw $t4, 16($t0)

**Data Hazards**  
I1 -> I2 ($t1) Read After Write Data Hazard

**Removing Data Hazard using NOP (add $1, $1, $0)**  
I1: add $t1, $t2, $t3  
I2: add $1, $1, $0   (NOP)  
I3: add $1, $1, $0   (NOP)  
I4: add $t4, $t1, $t2  
I5: add $t2, $t3, $t5  
I6: sw $t1, 12($t0)  
I7: sw $t4, 16($t0)

A screenshot of a cell phone

Description automatically generated

**RAW: Instructions lw and sub. Register $t1**

**RAW: Instructions add and sub. Register $t1**

The pipeline chart:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| CPU Cycles | | | | | | | | |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| IF | ID | EX | MEM | WB |  |  |  |  |
|  | IF | ID | +|- (i) | MEM | WB |  |  |  |
|  |  | IF | ID | S | S | +|- (i) | MEM | WB |

Applying it forward will take less time and won’t stall

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| IF | ID | EX | MEM | WB |  |  |
|  | IF | ID | +|- (i) | MEM | WB |  |
|  |  | IF | ID | +|- (i) | MEM | WB |

A screenshot of a cell phone

Description automatically generated

Initial state of predictor = Taken  
$t1 = 3

**first execution of instruction beq $t1, $zero, out**  
prediction = Branch taken  
Here $t1 not equal to 0, Hence Branch Not taken.  
Next state of predictor = Not Taken  
value of $t1 = 2

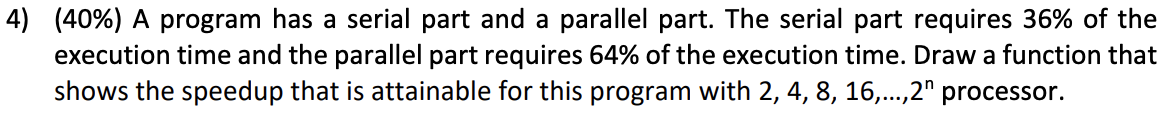
**first execution of instruction beq $t1, $t1, Loop**  
prediction = Branch Not taken  
Here $t1 is equal to $t1, Hence Branch is taken.  
Next state of predictor = Taken

**second execution of instruction beq $t1, $zero, out**  
prediction = Branch taken  
Here $t1 not equal to 0, Hence Branch Not taken.  
Next state of predictor = Not Taken  
value of $t1 = 1

**second execution of instruction beq $t1, $t1, Loop**  
prediction = Branch Not taken  
Here $t1 is equal to $t1, Hence Branch is taken.  
Next state of predictor = Taken

**Third execution of instruction beq $t1, $zero, out**  
prediction = Branch taken  
Here $t1 not equal to 0, Hence Branch Not taken.  
Next state of predictor = Not Taken  
value of $t1 = 0  
**Third execution of instruction beq $t1, $t1, Loop**  
prediction = Branch Not taken  
Here $t1 is equal to $t1, Hence Branch is taken.  
Next state of predictor = Taken

**Fourth execution of instruction beq $t1, $zero, out**  
prediction = Branch taken  
Here $t1 is equal to 0, Hence Branch is taken.  
Next state of predictor = Taken



Amdahl's law can be used to speed up the program

SpeedUp = \frac{1}{(1-FE)+ \frac{FE}{SE}}

FE is the parallel fraction of the code

SE is the number of processors available

In our case FE = 0.64

If we substituting the value of FE in the formula gives :

SpeedUp = \frac{1}{(0.36)+ \frac{0.64}{SE}}

Now in our case SE = 2^{n}

So the function for Speedup is:

SpeedUp = \frac{1}{(0.36)+ \frac{0.64}{2^{n}}}

By changing the value of n, we can get the speedup for the different number of processors