Question 1

Part A

Using Amdahls law we can calculate the speedup.

$$\begin{aligned}\frac{1}{(1-f(e))+\dfrac{f(e)}{speedup(e)}} &= \frac{1}{(1-0.4)+\dfrac{0.4}{8}}\\ &= \frac{1}{0.6+0.05}\\ &= \frac{1}{0.65} = 1.538\end{aligned}$$

The overall speedup on the target workload is 1.538

Part B

$$\begin{aligned}\frac{1}{1-f(e)} &= \frac{1}{1-0.4}\\ &= \frac{1}{0.6} = 1.67\end{aligned}$$

The max speedup with the coprocessor is 1.67

Part C

First lets calculate the speedup of option-A

$$\begin{aligned}\frac{1}{(1-f(e))+\dfrac{f(e)}{speedup(e)}} &= \frac{1}{(1-0.6)+\dfrac{0.6}{1.05}}\\ &= \frac{1}{0.4+0.571} = \frac{1}{0.971} = 1.03\end{aligned}$$

Then we can calculate the speedup of option-B.
Current speedup $= 8 \rightarrow$ New speedup $= 8 \times 1.2 = 9.6$

$$\begin{aligned}\frac{1}{(1-f(e))+\dfrac{f(e)}{speedup(e)}} &= \frac{1}{(1-0.4)+\dfrac{0.4}{9.6}}\\ &= \frac{1}{0.6+0.042}\\ &= \frac{1}{0.642} = 1.558\end{aligned}$$

I think based on the numbers option-B would have to be a better choice as the overall speedup is much better than compared to option-A

Question 2

Part A

$$\begin{aligned}\text{avg CPI} &= 0.2 \cdot 1 + 0.25 \cdot 4 + 0.3 \cdot 7 + 0.25 \cdot 5 \\ &= 0.2 + 1 + 2.1 + 1.25 \\ &= 4.55\end{aligned}$$

Part B

First we reduce the CPI of branches by 1

$$\begin{aligned}\text{avg CPI} &= 0.2 \cdot 1 + 0.25 \cdot 3 + 0.3 \cdot 7 + 0.25 \cdot 5 \\ &= 0.2 + 0.75 + 2.1 + 1.25 \\ &= 4.3\end{aligned}$$

Next we reduce the CPI of load instructions

$$\begin{aligned}\text{avg CPI} &= 0.2 \cdot 1 + 0.25 \cdot 4 + 0.3 \cdot 6 + 0.25 \cdot 5 \\ &= 0.2 + 1 + 1.8 + 1.25 \\ &= 4.25\end{aligned}$$

Last we can reduce the CPI of store instructions

$$\begin{aligned}\text{avg CPI} &= 0.2 \cdot 1 + 0.25 \cdot 4 + 0.3 \cdot 7 + 0.25 \cdot 4 \\ &= 0.2 + 1 + 2.1 + 1 \\ &= 4.3\end{aligned}$$

We can see that if we reduce the CPI of load instructions that it makes the most difference with an average CPI of 4.25

Part C

Let x be the CPI of branches. We then have the following CPI after the load and store instruction speedups

$$\begin{aligned}\text{avg CPI} &= 0.2 \cdot 1 + 0.25 \cdot x + 0.3 \cdot 6 + 0.25 \cdot 4 \\ &= 0.2 + 0.25x + 1.8 + 1 \\ &= 3 + 0.25x\end{aligned}$$

Since the original average CPI was a 4.55, then the max CPI that branch

x	4	5	6	7
avg CPI	4	4.25	4.5	4.75

instructions can have whilst still having a net speedup is 6

Part D

Old average CPI = 4.55

$$\begin{aligned}\text{avg CPI} &= 0.25 \cdot 1 + 0.25 \cdot 4 + 0.25 \cdot 7 + 0.25 \cdot 5 \\ &= 0.25 + 1 + 1.75 + 1.25 \\ &= 4.25\end{aligned}$$

Although the average CPI is lower with the new computer. Due to the 1.1x increase in total instructions, the overall execution time of the new computer would overall be much longer, since $4.25 \cdot 1.1 = 4.675$

Question 3

Part A

i) y = t0, a = t1, b = t2, c = t3

```
ADD t0, t1, t2
SUB t0, t0, t3
```

ii) x = t0, y = t1

```
BNE t0, 2, else
ADD t1, 0, 3
JAL t2, end
else:
ADD t1, t0, 0
end:
```

iii) a = s0, N = t0, x = t1, i = t3

```
findX:
loop:

LW t2, 0(s0)
BEQ t2, t1, return

ADD, t3, t3, 1
ADDI, s0, s0, 4
BLT t3, t0, loop
return:

JAL ra, *something*
```

Part B

```
li t4, 0 #long total = 0
whileLoop: #assume t0 is the adress of the head
beq t0, x0, end #if ptr == 0 then end
lw t1, 0(t0) #loading ptr->data
add t4, t4, t1 #adding to total
lw t2, 4(t0) #loading ptr->next
add t0, t0, t2 #moving pointer
jal ra, whileLoop
end:
```

Part C

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
beq t0, x0, end	F	D	X	M	W										
lw t1, 0(t0)		F	D	X	M	W									
add t4, t4, t1			F				D	X	M	W					
lw t2, 4(t0)							F	D	X	M	W				
add t0, t0, t2								F				D	X	M	W

There should be two stalls due to dependency issues, which are both simply waiting for the load instructions to write back into the registers

Part D

The simplest way to improve my current implementation is to reorder the load instructions so that they are next to each other. This way, the add instructions still have the dependency, but only the first add instruction will have to stall. When the stall is finished, the second load instruction should be finished, so the second add instruction should be able to run without stalls.

Question 4

Part A

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
F	D	X	M	W															
	F				D	X	M	W											
					F				D	X	M	W							
									F	D	X	M	W						
										F				D	X	M	W		

Part B

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
F	D	X	M	W															
	F	D		X	M	W													
			F	D	X	M	W												
				F	D	X	M	W											
					F		D	X	M	W									

Part C

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
F	D	X	M	M	W														
	F	D		X	M	M	W												
			F	D	X	M	M	W											
				F	D	X	M	M	W										
					F		D	X	M	M	W								

I believe that it will reduce the IPC slightly since it simply creates more cycles per loop iteration. The best way to solve this issue is to move the `addi a1, a1, 8` instruction to right after the `ld t0, 0(a1)`. With this change, it gets rid of the necessary stalls due to dependency issues.

Part D

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
F	F	D	X	M	W														
	F	F	D	X	M	W													
			F	F	D	X	M	W											
				F	F	D	X	M	W										
					F	F	D	X	M	W									

It does not worsen the IPC The extra "fetch" step in the cycle and essentially gets rid of all the stalls that were needed before. Some instructions are finished in later cycles but the design change does not affect the overall number of cycles needed per loop iteration.