Q1. When it comes to optimizing a processor, it's a common scenario where improving one aspect of performance can inadvertently affect another. For instance, introducing a sophisticated, fast floating-point unit may boost floating-point operations, but that takes space, and something might have to be moved farther away from the middle to accommodate it, adding an extra cycle in the delay to reach that unit. The basic Amdahl's law equation does not take into account this trade-off.

(a) If the new fast floating-point unit speeds up floating-point operations by, on average, 2×, and floating-point operations take 30% of the original program's execution time, what is the overall speedup (ignoring the penalty to any other instructions)? **[2.5 Marks]**

(b) Now assume that speeding up the floating-point unit slowed down data cache accesses, resulting in a 1.5× slowdown (or 2/3 speedup). Data cache accesses consume 10% of the execution time. What is the overall speedup now? **[2.5 Marks]**

(c) After implementing the new floating-point operations, what percentage of execution time is spent on floating-point operations? What percentage is spent on data cache accesses? **[5 Marks]**

__Ans:__

a) Speedup $=\frac{1}{(1-F)+\frac{F}{S}}$ where F = Fraction enhanced; S= Speedup enhanced

$=\frac{1}{(1-0.3)+\frac{0.3}{2}}=\frac{1}{0.7+0.15}$ =1.17

b) Overall Speedup= $\frac{1}{(1-F_1-F_1)+\frac{F_1}{S_1}+\frac{F_2}{S_2}}=\frac{1}{(1-0.3-0.1)+\frac{0.3}{2}+\frac{0.1}{2/3}}$= 1/0.9= 1.11

c) Floating point operation= $\frac{0.3/2}{(1-0.3-0.1)+\frac{0.3}{2}+\frac{0.1}{2/3}}$ =0.167=16.7 %

Data cache =$\frac{(0.1*3)/2}{(1-0.3-0.1)+\frac{0.3}{2}+\frac{0.1}{2/3}}$ = 0.167 = 16.7%

Q2. Consider a variable pipelined processor where arithmetic (addition, subtraction etc.) and memory based (load, store) instructions take different numbers of execution stages. Every stage takes one clock cycle. For the first category, arithmetic instructions, the execution stage has just one stage: **Y** while the memory based instructions have two execution stages: **X0** and **X1.** The selection of the number of pipeline stages is done at the decode stage based on the type of instruction CPU has to serve. So, the final pipeline stages for arithmetic instructions comprise of: **F, D, Y, W**. Pipeline stages for memory-type instructions contain: **F, D, X0, X1, W**. Where F, D and W correspond to the usual Fetch, Decode and Writeback stages, respectively. Full bypassing is allowed in this processor. The value to be bypassed (forwarded) to any proceeding instruction in case of any dependency, will only be available at the last **execution** stage of the instruction computing this value, till then stalling in the proceeding instruction will occur.

(a) Draw a pipeline diagram for the following set of instructions and report the total number of clock cycles required: **[7 Marks]**

```
1. LD   R1,  0(R2)
2. ADD  R3,  R1,  R2
3. SW   R3,  0(R4)
4. LD   R5,  0(R4)
5. SUB  R6,  R5,  R7.
```

(b) Calculate the CPI (cycles per instruction) for the program above. **[3 Marks]**

Ans.

(a)

	1	2	3	4	5	6	7	8	9	10	11
`LD R1, 0(R2)`	F	D	X0	X1	W						
`ADD R3, R1, R2`		F	D	D	Y	W					
`SW R3, 0(R4)`			F	F	D	X0	X1	W			
`LD R5, 0(R4)`					F	D	D	X0	X1	W	
`SUB R6, R5, R7`							F	D	D	Y	W

Marks Distribution: 0.5 + 1 + 2 + 2 + 1 ⇒ 6.5 Marks

- Total Clock cycles = 11 ⇒ **0.5 Marks**

(b) Lower value of CPI is desirable.
CPI ⇒ 11/5 = 2.2

Marks Distribution: 1.5 (showing calculation) + 1.5 (correct answer) ⇒ 3 Marks

Q3. Consider a 32-bit machine with a set-associative cache. The total cache size is 64 KB, and it is an 8-way set-associative cache. The length of the memory location is 4 bytes, and the cache line size is 64 Bytes.

(a) Calculate the total number of sets present in the cache. **[2 Marks]**

(b) Draw the 32-bit memory address showing the tag, index, and offset bits. **[2 Marks]**

(c) From what memory locations are the data fetched if the 9A6C7D90 memory location is accessed? Determine the set number where it is stored. **[6 Marks]**

Ans.

A) Number of sets = 64KB / 8*2^6 = 2^7 = 128 sets **[2 Marks]**

B)

Number of processor words = Cache line size/ Processor word size = 64/4 (in bytes) = 16 **[0.5 Mark]**

This implies 4 bits for the offset and 7 bits for the index.

Tag (21 bits)	Index (7 bits)	Offset(4 bits)
31 11	10 4	3 0

[0.5 *3 = 1.5 Marks]

C) Each cache line stores 16 words. When the address is fetched, it fetches 16 words of data from memory. **[1 Mark]**

- One memory location is 4 bytes and the word size of the processor is 4 bytes. This implies that a memory location is accessed for every processor word. The last byte of the address is 0 (0000), which means that the byte is stored at position 00 (binary)/ 0 (decimal). Thus, to fill the cache lines, memory locations from 9A6C7D90 to 9A6C7D9F are fetched. **[3 Marks]**
- The bits [10:4] represent the index bits and, hence, the set number. The data is stored at set number 59 (hex)/ 89 (decimal). **[2 Marks]**

Explanation on Offset Bits:
There can be two approaches to solve this problem. **Look into second approach only if you are very clear on how the processor- main memory interaction occurs.**

Approach 1:
The method to determine offset size ($log_2(Cache\ Line\ Size)$) assumes a byte-addressable memory (that is one byte is accessed at a time). However, in this question, the memory location size is 4 bytes. Hence, the formula is given as:

$$log_2(\frac{Cache\ Line\ Size}{Memory\ length}) = log_2(\frac{64}{4}) = log_2(16) = 4$$

In case of byte addressable memory, the denominator is always one. This is not valid here.

Approach 2:

The term offset can effectively be understood as a combination of block offset and byte offset. First, the block offset term. The processor provides 32-bit address to the memory and expects 32-bit data in return. In effect, it means that the processor effectively access processor word. Here, the processor aims to access 4 bytes out of 64 bytes. That leaves 64/4 = 16 possible combination. Hence, block offset = 4.

Byte offset comes into account when we have a byte addressable memory. When memory size is one byte and processor word size is 4 bytes, four addresses are fetched together to form the memory word. So, if the first access is at address 0 (0000), next will be at 4 (100), 8(1000), 12 (1100) and so on. Note that the last 2 bits are never accessed and hence not relevant. They are excluded as byte offset. In this question, however, as the memory location size and processor word size is same, access is at 0,1,2,3. Hence, the block offset is 0. Hence,only the block offset exists, which leaves the offset as 4.