1 Given,

instructions	clocks cycles
ALU operations	1.0
Loads	3.5
stores	2.8
Branches	
Taken	4.0
not taken	2.0
Jumps	2.4

avg $\frac{4+2}{2} = 3.0$

need to find effective CPI for RISC-V.

→ obtain average instruction frequencies by considering Perlbench & sjeng.

instruction	Perlbench	sjeng	Avg. freq.
loads	25%	19%	22%
Stores	14%	7%	10.5%
Branches	15%	15%	15%
Jumps	7%	3%	5%
ALU	39%	56%	47.5%

for eff. CPI.

$$CPI = (22 * 3.5 + 10.5 * 2.8 + 15 * 3.0 + 5 * 2.4 + 47.5 * 1.0) \times \frac{1}{100}.$$

$= (0.77 + 0.294 + 0.45 + 0.12 + 0.475)$

$= 2.109$ //

$\Rightarrow$ Effective CPI for RISC V mentioned is 2.109 //

2. Given,

a) So we need to see the execution time. in case of without enhancement. ~~% of E.T.~~ also, the computer worked. % of E.T.

enhanced mode → 50%.

non-enhanced mode → 50%.

without enchancement, non-enhanced mode. takes 50%. but enhanced takes 10 times longer ie, 500%.

So, this means without enhancement, relative Execution time = 50% + 500% = 550%.

$$\text{Speed UP} = E.T_{(\text{non-enh})} / E.T_{(\text{enhan})}$$

$$= 550 / 100 = 5.5 //$$

b. % of original E.T has been converted to fast mode.

By Amdahl's law, (speed UP → S.U)

$$\text{fraction vector} = \frac{S.U_{\text{overall}} * S.U_{\text{fast}} - S.U_{\text{non-fast}}}{S.U_{\text{overall}} * S.U_{\text{fast}} - S.U_{\text{overall}}}$$

$$\Rightarrow \text{fraction Vertel} = \frac{5.5 * 10 - 10}{5.5 * 10 - 5.5}$$

$$= \frac{55 - 10}{55.5.5}$$

$$= \frac{45}{49.5}$$

$$= 0.9090$$

$$\Rightarrow 90.90\ \%.$$

4. Given,

Consider basic RISC-V, 5-stage Pipeline & full forwarding.

(i) How Pipelining can improve the performance of a given instruction mix.

→ Pipelining is a process used to improve performance of a given instruction mix, by providing parallelism to the set of instructions executed. Although they execute instructions in sequence, certain parts of execution overlaps and improves performance via higher throughput.

(ii)

	1	2	3	4	5	6	7	8	9	10	11
lw x10, 0(x11)	F	D	E	M	W						
add x9, x11, x11		F	D	E	M	W					
sub x8, x10, x9			F	D	E	M	W				
lw x7, 0(x8)				F	D	E	M	W			
sw x7, 4(x8)					F	D	E	M	W		

(iii) where sw gets data from?

Here data is present in 'lw' in the MEM/WB register. Here we can also have a feedback path from this register to one of the i/p's to the data memory. From there then a control signal could select the appropriate i/p to data memory to support this lw-sw i/p combination.

B.

```
beq x1,x2,X
lw x10,0(x11)
sub x14,x10,x10
X: add x4,x1,x2
lw x1,0(x4)
sub x1,x1,x1
add x1,x1,x1
```

	1	2	3	4	5	6	7	8	9	10	11	12	13
	F	D	E										
		F	D										
			F										
				F	D	E	M	W					
					F	D	E	M	W				
						F	D	E	M	W			
							F	D	E	M	W		
								F	D	E	M	W	

→ add depends on sub instruction before it & produces the data which was consumed by add instruction.

5. For cache organized in two ways mentioned,

(i) more data elements per block & fewer blocks,

Cons:

→ There is higher potential for conflict misses as there will be fewer blocks.

Pros:

→ Higher performance can be achived from spatial locality due to larger block size.

Example:

In a case where we have high degree of sequential data access.

(ii) Fewer elements per block but more blocks.

Pros :-

→ Here in this case we may see fewer conflits misses due to more unique mappings unlike other case.

Cons:-

→ we may be more subjected to compulsory misses due to smaller block-size.

Example

→ In a case where we have more random memory access.

(ii) Given,

Processor has direct mapped cache.

Data words → 8 bits

Physical addess → 20 bits.

tag → 11 bits.

As Physical addess is 20 bits out of which 11 bits are tag bits, so 9 bits are left for index & off set.

→ Each block holds 16 bytes of data. 16 words & so, 4 bits are needed for off set.

⇒ index has 5 bits (9-4)

cache = 2^n

n → index

$= 2^5$

= 32 blocks in the cache.

(iii) Given,

16-way set-associative cache.

→ Data words are 64-bits long.

→ words are addressed to the half-word.

→ cache holds 2 Mbytes of data

→ Each block holds 16 data words.

→ Physical address are 64-bits long.

word length = 64 bits = 8 Bytes.

cache size in words = 2MB/8 = 256kB → 18bits

Block size = 16 words → 4 bits

no of lines = CS/BS = $2^{(18-4)}$

$= 2^{14}$ → 14 bits.

but, as we have 16 way set-associative.

set bits = 10 bits.

block bits = 4 bits.

Data word contains half of data word.

→ 64/2 = 32 = 2^5

→ offset → 5 bit

total capacity of cache → 2MB → 2^{21} bytes

→ total no. of blocks in cache.

= 2^{21} bytes (1 block / 16 words) * (1 word / 64 bits) * (8 bits / 1 byte)

= 2^{14} blocks.

Now, we have 2^4 blocks / set.

So there are 2^{14} blocks (1 set / 2^4 blocks)

= 2^{10} sets

⇒ index → 10 bits

Now, as we know offset & index we can get tag bits.

64 - 10 - 5 = 49

⇒ tag → 49 bits

index → 10 bits

offset → 5 bits //

3. Given,
we run 2 applications on our dual core.

→ 1st Application needs 80% of resources, & other needs 20%. So assume when we parallelize a Portion of Program speed up for that Portion is 2.
we know that,

$$\text{Speed UP} = \frac{1}{\left(1 - \text{Fraction of enhanced program}\right) + \frac{\text{Frac. enh. Prog}}{\text{S.UP}_{\text{enh. Portion}}}}$$

(1) Given,
40% of 1st application is paralligable.

$$\text{Speedup} = \frac{1}{(1 - 0.4) + \frac{0.4}{2.}}$$

$$= \frac{1}{0.6 + 0.2} = \frac{1}{0.8} = 1.25 //$$

(ii) Given,
99% of 2nd application is parallelized,

$$\text{Speed up} = \frac{1}{(1-0.99) + \frac{0.99}{2}}$$

$$= \frac{1}{(0.01) + 0.495}$$

$$= \frac{1}{0.505}$$

$$= 1.98 \text{ //}$$

(iii) Given,
40% of 4th application parallizable,
overal speed up = ?

$$\text{overall speed up} = \frac{1}{(1-0.8) + \frac{0.8}{1.25}}$$

$$= \frac{1}{0.2 + 0.64}$$

overall speed up = 1.19 //