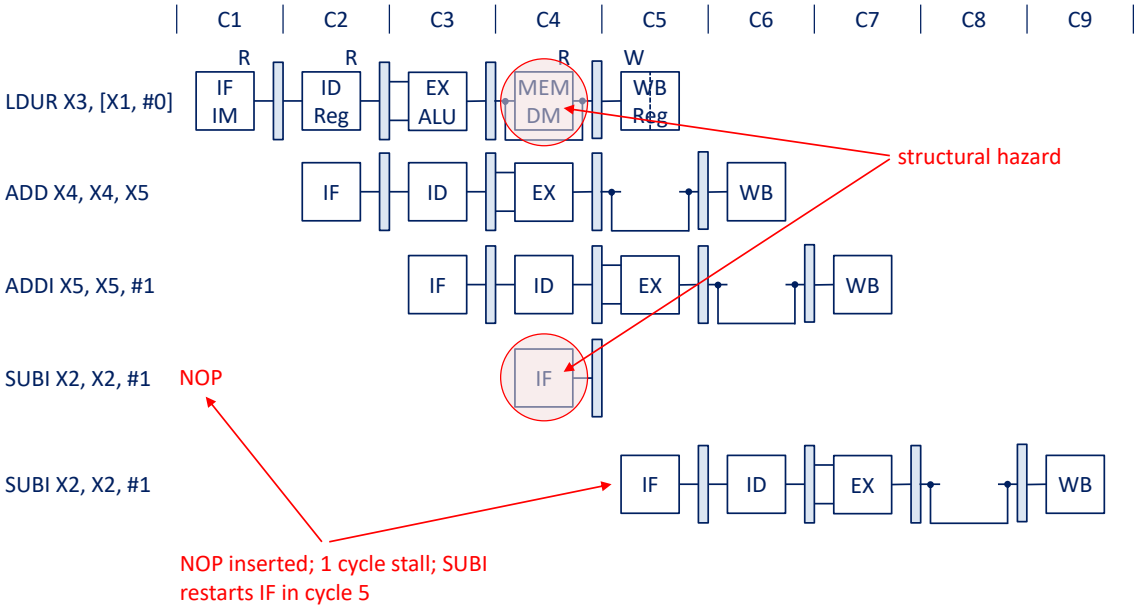
Question 1

(a) (i)

```
LDUR X3, [X1, #0]
ADD  X4, X4, X5
ADDI X5, X5, #1
SUBI X2, X2, #1
```

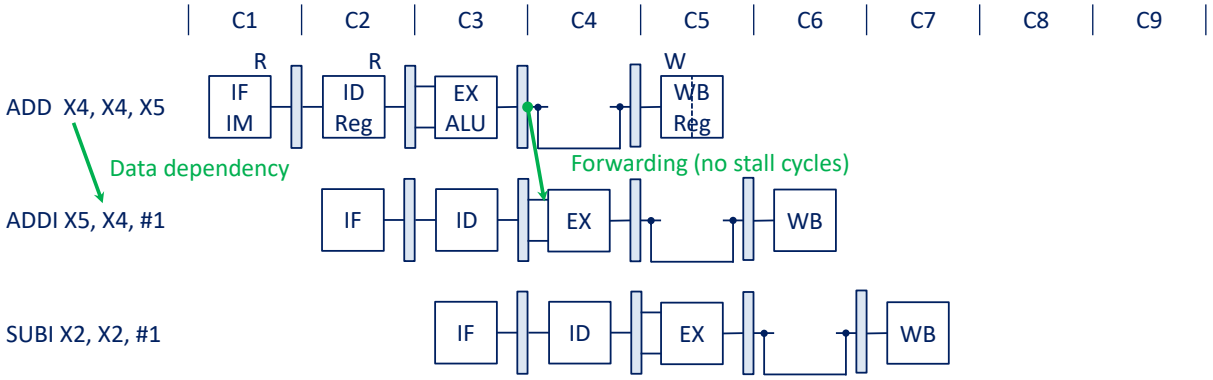


Answer Notes:

- *There are many equally correct answers*
- *Clearly show pipeline cycles*
- *Show the name of each pipeline unit*
- *Show the resource used by each pipeline unit for the first instruction*
- *Annotate the units (in cycle 4) where the hazard occurs*
- *Annotate the diagram to show the action of the interlock logic and the impact (stalled cycles) of the hazard.*

(a) (ii)

```
ADD  X4, X4, X5
ADDI X5, X4, #1
SUBI X2, X2, #1
```



Answer Notes:

- *There are many equally correct answers*
- *Show data dependency*
- *Clearly show pipeline cycles*
- *Show the name of each pipeline unit*
- *Show the resource used by each pipeline unit for the first instruction*
- *Annotate the diagram to show the action of the interlock and forwarding logic.*

(b) (i)

Definition of arithmetic operations

```
ADD   Ra, Rb    ; Ra = Ra + Rb

DIV   Ra, Rb    ; Ra = Ra / Rb
```

Program Z = (A + B + Z) / (A + Z)

```
LOAD  R0, A     ; R0 = [A]

LOAD  R1, B     ; R1 = [B]

LOAD  R2, Z     ; R2 = [Z]

ADD   R2, R0    ; R2 = A + Z

ADD   R1, R2    ; R1 = A + B + Z

DIV   R1, R2    ; R1 = (A + B + Z) / (A + Z)

STORE R1, Z     ; [Z] = (A + B + Z) / (A + Z)
```

Answer Notes:

- *There are many equally correct answers*
- *Define all arithmetic operations*
- *Define register usage in comment for each instruction*
- *Marks will be deducted for needless operations, but not looking for highly optimised program*
- *Must not overwrite values stored in variables A and B*

(b) (ii)

A 128 Ki word address space will be require Log_2 (128 x 1024) = Log_2 (2^7 x 2^{10}) = 17 address bits.

Load and Store instructions require:

- 6 bits to encode the operation
- 2 bits to encode the register
- 17 bits to encode the memory address

Minimum number of bits for Load/Store instructions = 25.

Arithmetic instructions require:

- 6 bits to encode the operation
- 2 bits to encode the source register
- 2 bits to encode the destination/source register

Minimum number of bits for arithmetic instructions = 10.

Question 2

(a) (i)

For two sequences of numbers X_i and Y_i, the geometric mean (GM) has the following property:

$$GM\left(\frac{X_i}{Y_i}\right) = \frac{GM(X_i)}{GM(Y_i)}$$

The geometric mean is the only mean that returns correct results when averaging normalised values as is the case when comparing the performance of a given computer system to a benchmark reference system.

GM for executions of the 5 benchmark programs executing on computer A is given by $(1000 \times 150 \times 250 \times 1500 \times 1500)^{1/5} = 610$

Performance ratio = 1200 / 610 = **1.967**

GM for executions of the 5 benchmark programs executing on computer B is given by $(1500 \times 200 \times 400 \times 1000 \times 200)^{1/5} = 474$

Performance ratio = 1200 / 474 = **2.531**

(b)

(i) Number of blocks in cache = (64 x 1024) / (512 x 2) = **64 blocks**

(ii) Number of sets = number of blocks / number of ways = 64 / 8 = **8 sets**

(iii) The 32-bit processor address comprises 3 fields (from MSB to LSB): tag bits, Set ID and Block Offset.

- The Set ID requires $Log_2(8) = 3$ bits to represent any of the 8 sets
- The Block Offset requires $Log_2(512) = 9$ bits to select the word in the block

There will therefore be 32 – 3 – 9 = **20 tag bits**.

(iv) 0xFC1C033F = 1111 1100 0001 1100 0000 0011 0011 1111

Separating into the three bit fields:

[1111 1100 0001 1100 0000] [001] [1 0011 1111]

Set number = **0x1**

Tag bits = **0xFC1C0**

(v) A 64-way set associative cache memory would require a tag bit comparator capable of simultaneously comparing 64 20-bit patterns from the cache memory with a 20-bit pattern extracted from the processor address. This is not feasible to achieve in combinatorial logic with a propagation delay that would not severely impact the cache access time.