Assignment 4

Q.1) Vector Architecture-

- Vector Architecture includes instruction set extension to an ISA to support vector operations, which are deeply pipelined.

- Vector architecture read sets of data elements into vector registers.

- Vector operations are on vector registers which are length of Bank of operations.

- Each vector operation takes vector registers and scalar value as input.

- It calculate the value using vector register and disperse the result back into memory.

- It canbe only effective on application that have significant Data level Parallelism.

Q.2) ① vadd v3, v1, v2 -

- It is used to add 2 vectors and add their result into third variable.

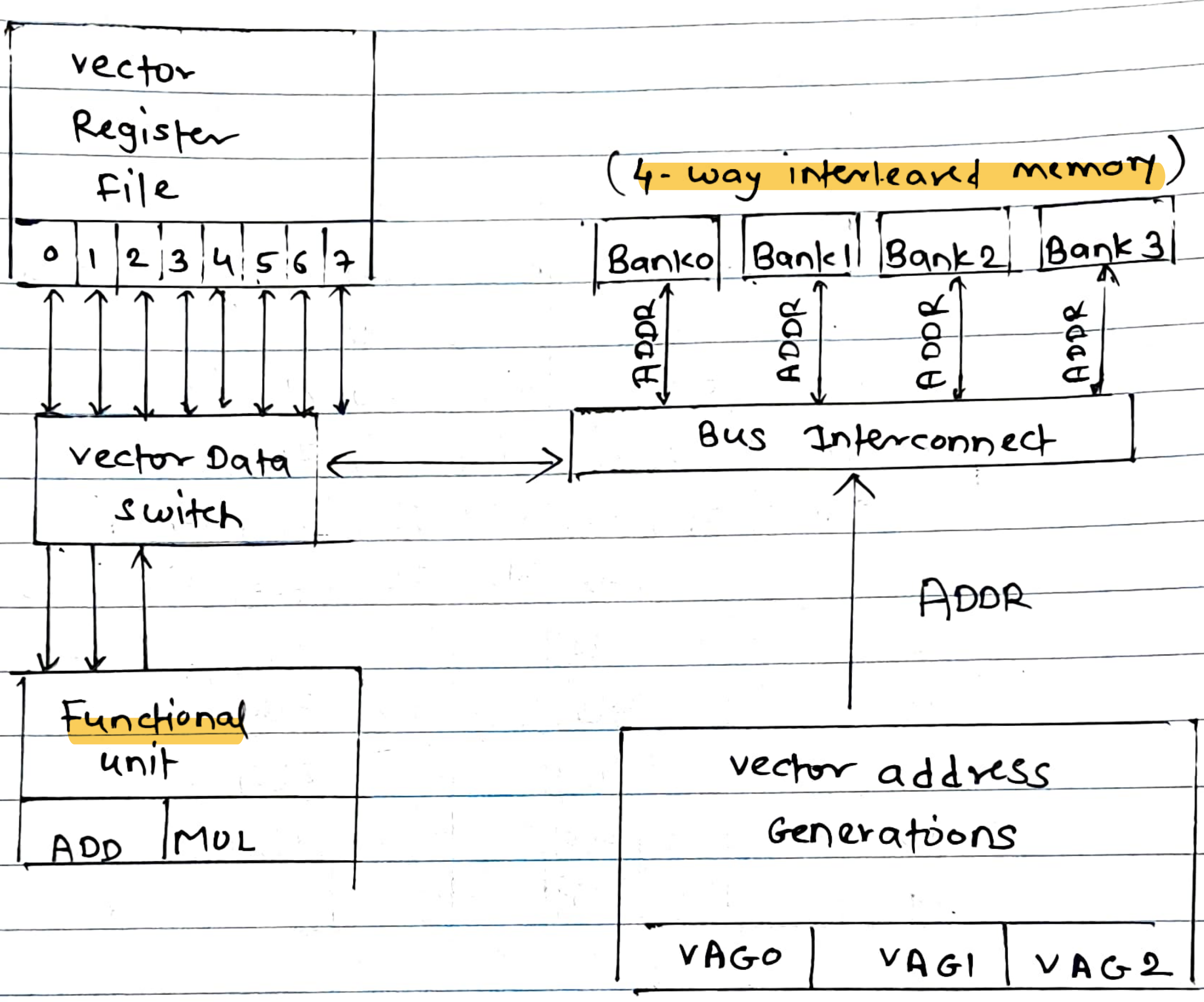
- The above instruction will add elements of v1 and v2, then put each result in third element v3.

② vld v1, r1 -

- It is used to load vector from memory address.

- The above instruction will load vector register v1 from memory starting at address r1.

Q.3)


vector Register File
0
1
2
3
4
5
6
7
(4-way interleaved memory)
Bank0
Bank1
Bank2
Bank3
ADDR
ADDR
ADDR
ADDR
vector Data switch
Bus Interconnect
ADDR
Functional unit
ADD
MUL
vector address Generations
VAG0
VAG1
VAG2

Best	VRF			Data switch			Addr 2 stages.		
1	x[0]	y[0]							
2	x[1]	y[1]		x[0]	y[0]				
3	x[2]	y[2]		x[1]	y[1]		z[0]		
4	x[3]	y[3]		x[2]	y[2]		z[1]	z[0]	
5	x[4]	y[4]		x[3]	y[3]		z[2]	z[1]	z[0]
6	x[5]	y[5]		x[4]	y[4]	z[0]	z[3]	z[2]	z[1]
7	x[6]	y[6]	z[0]	x[5]	y[5]	z[1]	z[4]	z[3]	z[2]
8	x[7]	y[7]	z[1]	x[6]	y[6]	z[2]	z[5]	z[4]	z[3]
9			z[2]	x[7]	y[7]	z[3]	z[6]	z[5]	z[4]
10			z[3]			z[4]	z[7]	z[6]	z[5]
11			z[4]			z[5]		z[7]	z[6]
12			z[5]			z[6]			z[7]
13			z[6]			z[7]			
14			z[7]						

8 elements vector add - (Data Pipelining)

- 7 bits to first result
- 1 bit to steady state result.

Q. 4)

Grid is a group of block. There is no synchroniezatio at all between the blocks. An entire grid is handled by a single GPU chip.

Thread Block -

It is a programming abstraction that represents a group of threads that can be executing serially or in parallely. for better process and data mapping, threads are grouped into thread blocks. The number of threads varies with available shared memory. The number of threads in a thread block is also limited by the architecture to a total of 512 threads per block. Threads in same thread block run on the same stream processor. Threads in same block can communicate with each other shared memory, barrier synchronization or other synchronization primitives such as atomic operation.

Q.5)

To map the data index to thread id on block id, we used -

(d) $i = \text{blockIdx.x} * \text{blockDim.x} + \text{threadId.x}$

$\uparrow$ Size of block

$\uparrow$ Built in varfable For each block

$\uparrow$ thread within a block

The above statement provides the best mapping among all the other choices.

This is the linearised global thread block index for the whole grid.

Q.6) In this S1 and S2, these are true dependencies, S2 cannot execute until S1 is completed, S2 depends on the result of S1.

S1 and S3 are output dependency, S3 complete before S1, then the result of S1 will be last.

S3 and S4 are anti-dependency, S3 should read c[i], before S4 changes c[i].

S2 and S3 are anti dependency, S2 should read A [i] before S3 changes A [i].

Due to all dependencies all the instruction have to be executed in pipeline-parallel execution is not possible.

Q.7) $S1 \Rightarrow A[i] = A[i] + B[i]$
$S2 \Rightarrow B[i+1] = C[i] + D[i]$

we can see that there is no dependencies between S1 and S2 as no elements A [i] or B [i] or C[i] or D[i] of S1 dependent on S2 or element of S2 on S1.

Loops are not parallel as we are running 2 lines of code inside for loop on same processo core to make it parallel.

we call loops parallel when we can run Statements by separate process in parallel like in difference CPU cores.

```
For (i=0 ; i<100 ; i++) {
    A[i] = A[i] + B[i] ;
}
For (i=0 ; i<100 ; i++) {
    B[i+1] = c[i] + D[i] ;
}
```

So, above loops will run parrallel in different CPU cores.

Q.8) The single peak precision flaating point throughput this CPU in

$= 1.5 * 16 * 16$

$= 1.5 * 256$

$= 384$ ~~GHz~~ .GFlop/s

Assuming each single precision operation require 4 byte in 2 operands and output one ~~[illegible]~~. 4byte result sustaining would require the memory bandwidth

$$= \frac{12 \text{ byte}}{\text{flop}} * 384 \text{ Gflop/s}$$

$= 4.216\ TB/s.$

Throughput is not sustainable because
$4.216\ TB/s \geq 100\ GB/s$

Therefore it can still be achieved in short burst when we using the chip cache.