8 Vector Processing [80 points]

Assume a vector processor that implements the following ISA:

Opcode	Operands	Latency (cycles)	Description
SET	V_{st}, #n	1	$V_{st} \leftarrow$ n (V_{st} = Vector Stride Register)
SET	V_{ln}, #n	1	$V_{ln} \leftarrow$ n (V_{ln} = Vector Length Register)
LDM	V_i	1	$V_{MSK} \leftarrow LSB(V_i)$ (V_{MSK} = Vector Mask Register)
VLD	V_i, #A	50 row hit, 100 row miss, pipelined	$V_i \leftarrow Mem[Address]$
VST	V_i, #A	50 row hit, 100 row miss, pipelined	$Mem[Address] \leftarrow V_i$
VMUL	V_i, V_j, V_k	10, pipelined	$V_i \leftarrow V_j * V_k$
VADD	V_i, V_j, V_k	5, pipelined	$V_i \leftarrow V_j + V_k$
VSHFR	V_i, V_j	10, pipelined	$V_i \leftarrow V_j >> 1$
VNOT	V_i	4, pipelined	$V_i \leftarrow BitwiseNOT(V_i)$
VCMPZ	V_i, V_j	4, pipelined	if($V_j == 0$) $V_i \leftarrow$ 0xFFFF; else $V_i \leftarrow$ 0x0000

Assume the following:

- The processor has an in-order pipeline and issues one instruction per cycle.
- There are 8 vector registers (V_0,V_1,V_2,V_3,V_4,V_5,V_6,V_7), and the size of a vector element is 4 bytes.
- V_{st} and V_{ln} are 10-bit registers.
- The processor *does not* support chaining between vector functional units.
- LDM moves the least-significant bit (LSB) of each vector element in a vector register V_i into the corresponding position in V_{MSK}. This instruction is executed in one single cycle.
- The main memory is composed of N banks, and each bank has a row buffer of size 64 bits.
- All rows in main memory are initially closed (i.e., all banks are precharged).
- The memory is byte addressable, and the address space is represented using 32 bits.
- Vector elements are stored in memory in a 4-byte-aligned manner. The first element of a vector always starts at the beginning of a memory row.
- Vector elements stored in consecutive memory addresses are interleaved between the memory banks. E.g., if a vector element at address A maps to bank B, a vector element at $A + 4$ maps to bank $(B + 1)\%N$, where % is the modulo operator and N is the number of banks. N is not necessarily a power of two.
- The latency of accessing memory is 100 cycles when the memory request misses in the row buffer, and 50 cycles when the memory request hits in the row buffer.
- Each memory bank has a single read and a single write port so that a load and a store operation can be performed simultaneously.
- There is one functional unit for executing VLD instructions and a separate functional unit for executing VST instructions. This means the load and store operations for *different vectors* cannot be overlapped.
- The operations on a vector do not affect the vector elements corresponding to the locations in the Vector Mask Register (V_{MSK}) that are set to 0.

(a) [20 points] What should the minimum number of banks (N) be to avoid stalls while executing a VLD or VST instruction? Calculate the minimum number of banks for every stride from 1 to 10. Explain.

101 banks for even strides, 100 banks for odd strides

Explanation.

To calculate the minimum value, we have to assume the worst case, which is when all memory accesses are row buffer misses (latency = 100). To avoid stalls, we need to ensure that consecutive vector elements access 100 different banks.

We illustrate the solution for even strides (2 and 4) and odd strides (1 and 3).

101 banks are enough to avoid stalls with even numbers. For example, with a vector stride of 2, consecutive elements of a vector will map to banks 0, 2, 4 ... 96, 98, 100, 1, 3 ... 97, 99. With a vector stride of 4, consecutive elements of a vector will map to banks 0, 4, 8 ... 96, 100, 3, 7 ... 95, 99, 2, etc.

100 banks are enough to avoid stalls with odd numbers. For example, with a vector stride of 1, consecutive elements of a vector will map to banks 0, 1, 2, 3 ... 98, 99. With a vector of stride 3, consecutive elements of a vector will map to banks 0, 3, 6 ... 96, 99, 2, 5 ... 95, 98

So, the minimum number of banks is 100 for odd strides, and 101 for even strides.

(b) [30 points] Translate the following loop into assembly code that can be executed in the least possible number of cycles in the previously described vector machine:

```
for i= 0 to 45:
    if(a[i] == 0):
        c[i] = b[i]
    else:
        c[i] = a[i] * b[i] + a[i]/2
```

Assume:

- The same machine as in part (a).
- In the for loop, 45 is inclusive, i.e., [0, 45]
- The size of the elements of vectors a, b, and c is 4 bytes
- Vectors a, b, and c do not share parts of the same DRAM row

```
SET V_st, 1        # Load Vector Stride Register
SET V_ln, 46       # Load Vector Length Register
VLD V_1, a         # Read from array a
VLD V_2, b         # Read from array b
VCMPZ V_3, V_1     # Compare V1 to 0
LDM V3             # Load Vector Mask Register
VST c, V2          # Write to array c
VNOT V3            # BitwiseNOT
LDM V3             # Load Vector Mask Register
VSHFR V_4, V_1     # Shift to divide
VMUL V_5, V_1, V_2 # Multiply
VADD V_6, V_5, V_4 # Add
VST c, V_6         # Write to array c
```

(c) [30 points] What is the number of cycles the previous code takes to execute in the vector processor described in this question? Assume:

- Vectors a and b are in different rows
- A machine that has a memory with 8 banks.
- The rest of the machine is the same as in part (a).

1822 cycles
Explanation.

The memory accesses look like:

```
bank0 --MISS-|--HIT--|--MISS--|--HIT--|--MISS--|--HIT--|
bank1  --MISS--|--HIT--|--MISS--|--HIT--|--MISS--|--HIT--|
bank2   --MISS--|--HIT--|--MISS--|--HIT--|--MISS--|--HIT--|
bank3    --MISS--|--HIT--|--MISS--|--HIT--|--MISS--|--HIT--|
bank4     --MISS--|--HIT--|--MISS--|--HIT--|--MISS--|--HIT--|
bank5      --MISS--|--HIT--|--MISS--|--HIT--|--MISS--|--HIT--|
bank6       --MISS--|--HIT--|--MISS--|--HIT--|--MISS--|
bank7        --MISS--|--HIT--|--MISS--|--HIT--|--MISS--|
```

Therefore, the latency of the load corresponds to the latency of the bank with the larger latency. In this case, bank 5 (300+150+5 = 455 cycles). The latency of a store is also 455 cycles.

The general picture is:

```
SET: |-S-|
SET:     |-S-|
VLD:         |-----VLD-----|
VLD:                         |-----VLD-----|
VCMPZ:                        |VCMPZ|
LDM:                                |L|
VST:                                        |-----VST-----|
VNOT:                                        |VNOT|
LDM:                                              |L|
VSHFR:                                              |VSHFR|
VMUL:                                                |VMUL|
VADD:                                                      |VADD|
VST:                                                             |-----VST-----|
```

S = 1
VLD_cycles = VST_cycles = 455
VMUL_cycles = 10 + 45 = 55
VCMPZ_cycles = 4 + 45 = 49
VNOT_cycles = 4 + 45 = 49
L = 1
VSHFR = 10 + 45 = 55
VADD = 5 + 45 = 50

Considering how the latency of some instructions is hidden by the other instructions, the total cycles can be calculated as:
$total\_cycles = S + S + VLD\_cycles + VLD\_cycles + VST\_cycles + VST\_cycles =$ 1822 *cycles*