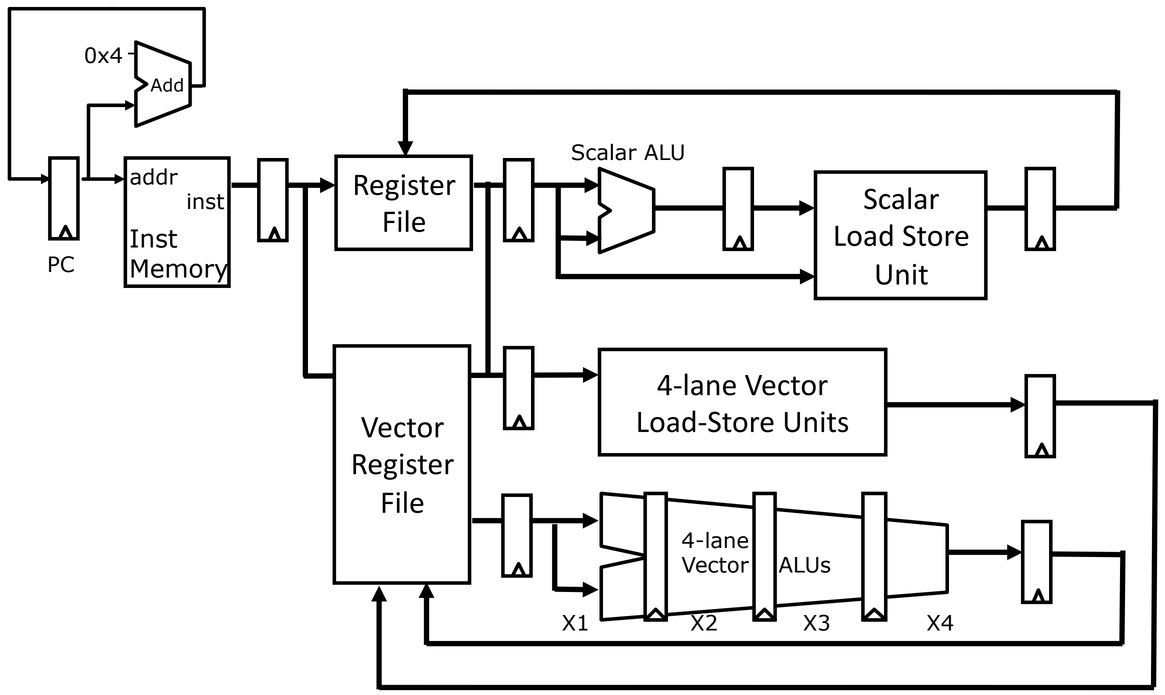
Quiz 4 Handout

Consider a vector processor with the following features:

- Single-issue, in-order execution
- Scalar instructions execute on a 5-stage, fully-bypassed pipeline
- 32 vector registers, **16 elements per vector register**
- **Four** vector lanes, with one ALU and one load-store unit per lane. Both take four cycles, are fully pipelined, and can process vector elements from independent instructions at the same time (the vector register file has enough ports per lane to feed both functional units)
- No support for vector chaining



The processor can issue a single (scalar or vector) instruction per cycle. Once it issues, a vector instruction uses the lanes' ALUs or load-store units for as many consecutive cycles as needed to produce all of its results. Vector instructions are maskable, but each lane always processes all its vector elements and turns off writeback for the masked ones. A vector instruction stalls if either its functional unit is unavailable, or if it depends on the result of a prior instruction, in which case it stalls until the prior instruction finishes writing back **all** of its elements. The vector register file has enough ports to keep the vector ALUs and load-store units fully utilized. The processor implements the MIPS ISA plus the following vector instructions:

Instruction	**Meaning**
`setvlr Rs`	Set vector length register (VLR) to the value in Rs
`lv Vt, Rs`	Load vector register Vt starting at address in Rs
`sv Vt, Rs`	Store vector register Vt starting at address in Rs
`add.vv Vd, Vs, Vt`	Add elements in Vs, Vt, and store result in Vd
`mul.vv Vd, Vs, Vt`	Multiply elements in Vs, Vt, and store result in Vd
`add.vs Vd, Vs, Rt`	Add Rt to each element in Vs, and store result in Vd
`mul.vs Vd, Vs, Rt`	Multiply each element in Vs by Rt, and store result in Vd
`s--.vs Vd, Rs`	Compare the elements (eq, ne, gt, lt, ge, le) in Vd and Rs. *For each element, if the condition is true, set the corresponding bit of the vector mask register to 1. If the condition is false, set the corresponding bit of the vector mask register to 0.*
`cvm`	Set all elements in vector mask register to 1.
`lvi Vt, Rs, Vs`	Gather load of vector register Vt starting at address in Rs and offsets from Vs. The i-th element of vector register Vt is loaded from an address that is the sum of Rs and the i-th element of Vs.
`svi Vt, Rs, Vs`	Scatter store of vector register Vt starting at address in Rs and offsets from Vs. The i-th element of vector register Vi is stored at an address that is the sum of Rs and the i-th element of Vs.