PDS1:

Deciding registers and their usage protocol

8 registers to store variables
8 registers to store temporary variables
ra - return address register
This is highlighted in the code.

PDS2:

Decide upon the size for instruction and data memory in VEDA.

Size for the instruction in VEDA: 32 bit wide and the memory can contain 32 instructions
Size for data memory in VEDA: 32 bit wide and the memory can contain 32 instructions

PDS3:

Instruction layout for R- type Instructions:

From left to right:
6 bits - op_code
5 bits - rs
5 bits - rt
5 bits - rd
5 bits - empty
6 bits - func_code

Instruction layout for I- type Instructions:

From left to right:
6 bits - op_code
5 bits - rs
5 bits - rt
16 bits - offset address or number

Instruction layout for J- type Instructions:

From left to right:
6 bits - op_code
5 bits - func_code
5 bits - empty
16 bits - address

PDS4:

Implementation of instruction fetch phase:

PC is implemented which keeps track of current address and a mode is maintained which is changed in always block. The instruction will be fetched when the mode is changed and will be decoded further in the same clock cycle.

PDS5:

Implementation of Instruction Decode:
decode.v file contains a decode module in which layouts for all the instructions are defined. The module takes an instruction as input and outputs op_code, func_code, rs, rt, rd, shamt, branch, and offset address depending upon instruction type.

PDS6:

Implementation of Arithmetic Logic Unit (ALU):
All the instructions and operations provided in ISA are executed using behavioral statements of verilog. We first check the layout of instruction using the obtained op_code, func_code, rs, rt, rd, shamt, branch, and offset address from decode module, perform the operations and store it in rd register.

PDS7:

Designing and implementation of the Branching operation along the ALU implementation:
Branch is already set in decode module. Whenever we want to fetch a new instruction or data from VEDA memory, we are going to change the mode in always block and it reflects in that clock cycle outside the always block.

PDS8:

Designing the FSM for the control signals:
There are 3 states designed: state_fetch, state_decode, state_execute. There are flags such as mode and decode_flag to instantiate modules outside always block. The states are incremented when the module is executed and next state modules are executed later.

PDS9:

Developing the MIPS Code for Bubble Sort:
bubble_sort.asm file contains the MIPS code for bubble sorting operation. And then by using the ISA, the instructions are converted into machine code.

PDS10:

Final execution of the CSE-BUBBLE
Finally, the machine code is stored in the instruction memory and on executing the instructions, the output which is the sorted data by bubble sort is stored in the data memory.

We have stored the numbers
78 456 1 89 13 56 267 102 3 51
in data memory.

Upon running “iverilog cse_bubble_tb.v” we get following output

```
Before Sorting:      78      456      1      89      13      56      267      102      3
51
After Sorting:      1      3      13      51      56      78      89      102      267      456
```