Our paper makes the following contributions :

- We presented a geneic Hoare-style progarm logic for SPARC code and our logic can support the main features of SPARC code. The main differences between SPARC code and other assembly codes are : (1) The control flow of SPARC program will not change to the target point immediately after executing the jump instruction, but after the instruction following the jump instruction. We call this feature "Delayed control transfer"; (2) SPARC has a series of special registers, like y register. Writing special registers will cause 0 ~ 3 cycles delayed. This feature is called "Delayed-write". CAP system views jump instruction as the end of a basic code block. However, in SPARC program, we can't regared jump instruction as a basic code block tail because of "Delayed control transfer". And the "Delayed-write" feature makes the executing of SPARC program different with our regular imperative language whose executing will change the program state immediately. So, it will cause some challenges for our logic rules designing. In our logic, we regard the instruction following jump as the end of a basic code block and define a new assertion [n]p which means the condition p will hold after n cycles.

- Our logic support module and local verification. Modularity and Locality are important techniques in program proof. Our verification framework supports module and local verification. We can check each basic code block independently. And the frame rule presented in our verfication framework supports a local way of reasoing about SPARC programs.

- We gave a semantics based soundness definition in our work. The previous works about assembly code reasoning like CAP guarantee that a program that can be checked by their logic rules will never get stuck. This syntax based soundness definition only make sure the program safety, however, can not describe the function of a program. The structureless assembly code makes us difficult to use the traditional semantics based soundness definition which means when a program execution starts from a state satisfying a precondition p, then the postcodition q will hold in the final state. Because a function can be split into several basic code blocks, not an integrated structure. In this paper, we present a semantics based soundness definition and prove that our logic is sound by Coq proof assistance.

- We designed an assertion languange in our work. Our assertion language provides a form to describe general registers, special registers, memory values in program state and the way to link separated parts together. The assertion language we presented has enough expression and straight-forward meanings. So we can use it write specifications of SPARC programs easily.

- Context switch is an important component in operation system. Most of context switch programs are written in assemly code because of the requirement to access registers and stack. We proved the correctness of the context switch algorithm in SpaceOS2 and presented that our logic is useful and can check realistic SPARC code easily and conviently.