fact is, that my new higher-level assembler is for simple code only, not touching interrupts, and hw access from code, in fact, primary goal of this new virtual assembler is to code the most compact and fast transformed RV32E interpreter itself in this new syntax, to be as much portable to all such processors as possible, and eventually, to write some native code for targe processors in it for native speed. main application code is considered to be written for virtual transformed RV32E cores

---

Understood. In that case, focusing on a compact and efficient syntax for the RV32E interpreter is a good strategy. Here are a few additional suggestions to help achieve this goal:

1. Use a minimalistic syntax that avoids unnecessary keywords and punctuation. For example, you could use single-letter mnemonics for instructions and operands, and eliminate unnecessary whitespace.
2. Use macro expansion or function calls to simplify commonly used code patterns. For example, you could define a macro that expands to a sequence of instructions that sets up a function call, or that implements a common algorithm.
3. Optimize the generated code for size and performance by using efficient instructions and avoiding unnecessary register spills and reloads.
4. Consider using an iterative development process, where you gradually build up the RV32E interpreter and test it on a range of target architectures. This can help you identify any issues or inefficiencies early on in the development process.
5. Document the syntax and usage of the new assembler thoroughly, and provide clear examples of how to use it to write code for the RV32E interpreter. This can help users quickly learn the syntax and start using it effectively.
