sounds very good, thanks; this means, that I can use first byte as quite comfortable opcode field together with third and fourth bytes for 16bit immediate values. As far as I design the embedded virtual processor implementable even on hardware 8bit MCUs, the 16MB address space seems to be okay for single core. Goal is to have as compact as possible interpreter code to fit in 8kB flash of 8bit MCU. Do you expect this as possible?

Yes, it is possible to design an embedded virtual processor with a compact interpreter that fits within the 8kB flash memory of an 8-bit MCU. By using a 20-bit address space, you can reduce the complexity of your instruction encoding, which will help minimize the interpreter's code size.

To achieve this goal, consider the following design principles:

1. Simplified encoding: As you mentioned, use the first byte as the opcode field, while the third and fourth bytes can be used for 16-bit immediate values. This will simplify the decoding process and reduce the size of the interpreter code.
2. Efficient instruction grouping: Group instructions based on register and immediate field usage, as discussed earlier. This will help to optimize the decoding process and reduce the interpreter's code size.
3. Compact instruction set: Limit the number of instructions in your ISA to only those that are essential for your use case. Reducing the number of instructions will simplify the decoding process and minimize the interpreter's code size.
4. Code optimization: Apply optimization techniques such as loop unrolling, function inlining, and dead code elimination to reduce the interpreter's code size further.
5. Data structure optimization: Use efficient data structures and memory management techniques to minimize the interpreter's memory footprint.
6. Compiler optimizations: Utilize compiler optimization flags (e.g., `-Os` in GCC) to generate more compact code for the interpreter.

While designing a compact interpreter for an 8-bit MCU with an 8kB flash memory is challenging, it is possible with careful planning and optimization. By following the design principles mentioned above, you can create a virtual processor that meets your requirements while fitting within the available memory constraints.