machine-dependent parts of the language, the editor should flag any usage errors.

Simulators. A high-level simulator will interpret HLML statements and let the user trace a microprogram in terms of the HLML. A low-level simulator could simulate each microinstruction produced by the compiler and report on the effects of the microcode execution in both high-level and low-level terms.

This low-level simulator is necessary to debug a compiler for a new processor and to help the microprogrammer at the hardware interface. It is necessary when debugging the hardware of a new processor to have complete information about the effects of the microprogram at a low level. The simulator will provide this information.

The low-level simulator must be table-driven to simplify reconfiguring the simulator (and thus the programming environment) for other targets. Work in this area includes the E-Machine workbench\(^\text{13}\) and research on using models of micro-operation format and semantics in simulator generation.\(^\text{14}\)

In addition to a resource checking capability, these simulators will provide execution profiles and timing information. These are a great help in determining where the microprogram spends its time and are far easier to implement in a simulator than in a compiler.

Assertions should also be a feature of the language and simulator. Instead of being executable (the usual case), these assertions should compile into breakpoint specifications for the simulator. If this is done, the real microprogram—not one altered by the presence of code for the assertions—can be simulated.

Performance requirements specification. Another part of the environment, which might be migrated into the language itself, should be a way of specifying the microprogram's response to asynchronous events. The specification must also be able to specify microprogram timing in the sense that a certain microcode segment must execute within a certain amount of time. Specifying this information in an understandable way is a difficult problem. Going from such a specification to code that satisfies requirements is a still more difficult problem. Certainly, research in real-time languages and systems will be valuable for microprogramming in the future.

Verification. A verification system might be included for some applications, but in general the cost of such a system—at least in the near future—would be excessive. This system would prove the correctness of the microprogram with respect to the given specifications. It will probably be interactive and will communicate with the microprogrammer through an interface similar to those used by the simulators.

A great deal of work has already been done on verification of microcode. This works suggests that a verification system, though expensive now, will become feasible for critical microprograms. An aid to making verification feasible is axiomatization of the microprogramming language. Work has been done in this area for SY(QM-I).\(^\text{15}\)