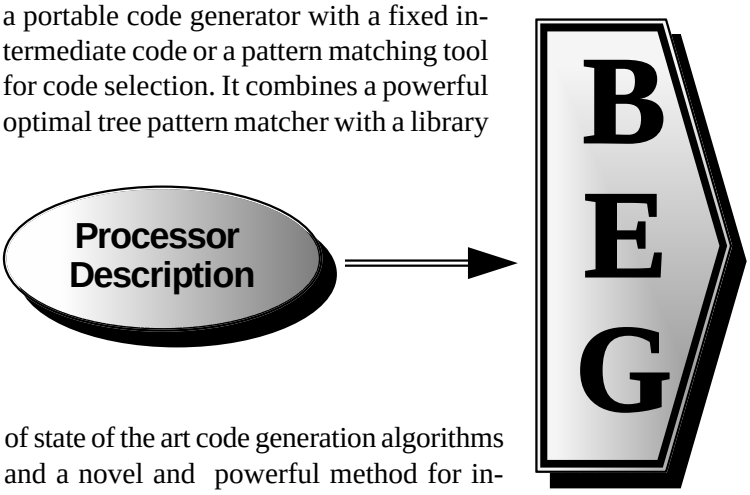
The BEG Code Generator Technology is part of the leading edge, industry proved compiler code generation technology. For you, as a compiler developer, it allows making better compilers for more target processors with a fraction of the effort needed before.

The BEG Code Generator Technology

Every new target processor requires developping new expensive code generators; a difficult and error prone task --- especially for the emerging high performance super scalar, requiring more and more difficult optimizations to be done. The BEG technology solves this problem for you with a revolutionary approach: the machine generation of the code generator. You develop a codegenerator by simply formulating a description of the target processor. The BEG program reads this description and automatically assembles the codegenerator (see figure).

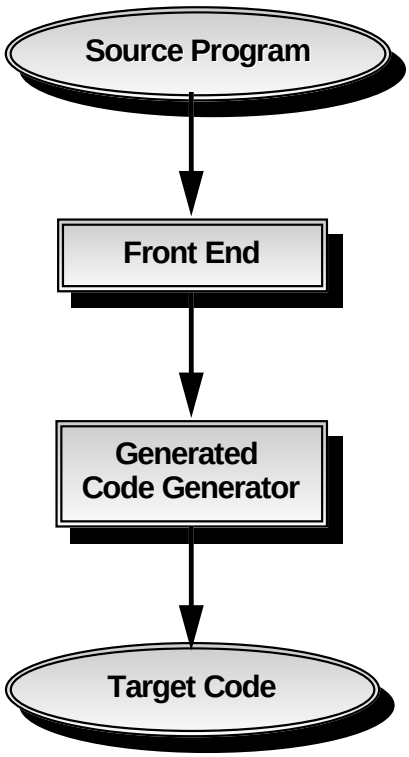
The BEG technology is much more than just a portable code generator with a fixed intermediate code or a pattern matching tool for code selection. It combines a powerful optimal tree pattern matcher with a library of state of the art code generation algorithms and a novel and powerful method for instruction scheduling.



BEG uses a variety of technologies to generate the parts of the code generator, tree automata for code selection, DAG matching for integration with SSA based optimizers (e.g. the firm optimizer) machine simulation automata for scheduling, integrated global/local register allocation plus an automatic selection and parameterization of components from a BEG internal library. All generation is guided by one single integrated processor description, which simplifies formulating processor descriptions and guarantees the integrity of the whole system.

BEG technology allows you to build a high quality code generator in a fraction of the time and effort needed by hand writing. The resulting code generator is very reliable because it is generated from an automatically checked formal specification in addition to which carefully tested components from the library are reused. The BEG technology has proved mature in several research and industrial compiler projects, for example an industrial compiler family for SPARC™ processors. Code generators for INTEL™ 80x86, MC68020, and MIPS™ R3000 have also been produced. With BEG you can have both very fast code generators with reasonable code quality as well as code generators producing very high code quality, from almost the same processor description. The latter code generator contains a global register allocator and a combined list and resource scheduler.

The BEG technology is currently available in C for UNIX™ and WINDOWS™. The generated compilers are C programs, which run in almost every C environment.



Technical Details

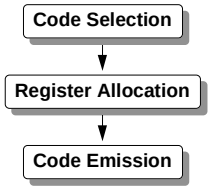
Machine Generation Machine generation means that the desired code generator program (a collection of C files) is generated from a processor description by the generator program BEG (see figure). You write a processor description instead of the programs itself. Writing a processor description is much easier, faster, and safer than writing the programs themselves. Your processor description is checked for consistency leading to an improved reliability of the code generators produced.

Intermediate Language You can interface to existing compiler front ends without changing the front end or translating the intermediate code by simply describing the intermediate code in the processor description.

Flexibility To cope with the enormous variety of features offered by current processors great flexibility is needed. The BEG technology offers this using a general tree matching mechanism, by a graph colouring based and highly parameterizable global register allocator, and the resource scheduling technique. Conditions and actions in the processor description use the full power of the C language. Good interfaces support hand programming of special parts of the code generator, should this be required.

Code Generator Architecture A hand written code generator has a fixed architecture, which is set up during design according to the desired code quality and compilation speed. With BEG you can produce code generators of very different architectures with little changes in the processor descriptions. The diagrams show the architecture of a simple, high compilation speed code generator in contrast to a much more complex optimizing code generator.

Integrated Code Generator As there are many parts of a code generator, there are several different methods to generate these parts. In contrast to other tools, BEG generates all components from a single integrated processor description. This guarantees consistency among the parts, facilitates integration, and simplifies the development of processor descriptions.



Structure High Speed Code Generator

Generation using Automata Theoretic Methods Automata theory offers algorithms to produce programs („automata“) based on certain sets of rules. These techniques have been used successfully in parser generators for a long time. The extension to trees, the tree automata theory, now makes is possible to automatically produce code selectors. Novel techniques use the automata to simulate the target processor, so making the resource scheduling feasible.

Generation by Parameterization Other parts of the code generator (e.g. register allocation) are taken from a BEG internal library and parameterized according to the processor description. This allows reusing these highly complex algorithms in any BEG generated code generator. Consistency is automatically maintained by the BEG generator program.

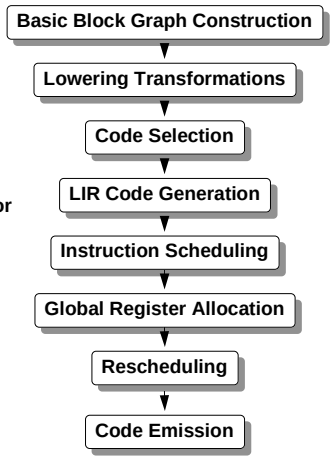
Code Selection by Tree Pattern Matching Code selection selects machine instructions for an intermediate code tree from the processor description, which describes each instruction and addressing mode by a tree pattern matching rule and a cost value. The machine generated code selector selects the instructions for an intermediate code tree in an *optimal* way according to the costs given in the description. So a processor description just describes what code can be selected, an algorithm selecting the optimal code is deduced automatically and needs not be programmed.

Code Selection by DAG Matching is an option. It is very powerful, if the intermediate language is not in tree form but represented as DAG. This is typlically the case if an SSA based optimizer is used. DAG matching provides a much better code quality, because the pattern matching now works on a complete basic block rather than a single expression tree.

Register Allocation You can select between local and global register allocation. Local register allocation works on the scope of an intermediate code statement, making it very fast. The global register allocator works on a complete procedure at once. So intermediate results can stay much longer in registers resulting in high quality code. The global register allocator itself uses in an integrated local/global allocation method. The machine's register set and the legal registers for each machine instruction can be specified, covering a wide range of target machines.

Instruction Scheduling A combined list and resource scheduler is generated. List scheduling reorders instructions, so that the pipeline of the processor needs to wait as seldom as possible for an intermediate result not yet computed. Resource scheduling takes a much deeper knowledge of the machine’s pipeline into account to avoid possible pipeline conflicts based on the limited number of functional units on the processor. Especially, it can handle multiple issue processors, which can perform more than one instruction in parallel. To make resource scheduling possible, the target processor is simulated while scheduling. This became practically possible only after applying finite automata techniques, as invented in [3].The instruction scheduler still requires some adaptation to new environments.

Uses of BEG technology BEG was first used in the very fast GMD Modula-2 compiler Mocka™. Code generators were produced for Motorola 68020, for SPARC™, for MIPS™, and INTEL 386, INMOS™ T800 and PowerPC™. This compiler is widely used in research and education. The Linux version is a major inroll available for free in source code. Its processor description was completely done by a student previously not familiar with BEG within 3 months.



Structure Optimizing Code Generator

BEG was used to produce an industrial compiler family for C, ANSI C, Modula-2, Fortran-77, and Pascal, for the SPARC processors.

Publications More information about BEG is contained in

[1] *Emmelmann, Schröer, Landwehr:* BEG-a Generator for Efficient Back Ends, Sigplan´89 Conference, Sigplan Notices, Vol.24 Nr.7
[2] Aßmann,Emmelmann,Grosch: Stein auf Stein, Cocktail: Eine Compiler-Compiler-Toolbox, IX Magazin 2/92
[3] T.Müller: Employing Finite Automata for Resource Scheduling, MICRO-26; Austin Texas, December 93
[4] Codeselektion mit regulär gesteuerter Termersetzung, PhD-Thesis, GMD Bericht NR 241, R.Oldenbourg Verlag, 1994
[5] BEG User Manual, 2003, availble at H.E.I.

Support and Maintainance BEG is supported and maintained by H.E.I.

Distribution BEG is available as a basic package to generate non-optimizing compilers. Options are the DAG matching, the global register allocator, and the instruction scheduler.

Usually BEG is sold on a processor architecture based pricing or for a flat fee. BEG is available as object or as C source code. With these licenses usually the generated code generators can be distibuted without royalties.

For more information please contact

H.E.I. Informationssysteme GmbH
Wimpfenerstraße 23
68259 Mannheim, Germany
Tel: +49 621 795141, Fax: +49 621 795161
E-mail: info@h-e-i.de

Mocka is a trademark of the German national research center for computer science (GMD)
The BEG technology was supported by the ESPRIT research program of the European Community in the project COMPARE (#5399)