UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
90/011,490	02/15/2011	RE38104	13557.112021	8186

25226 7590 02/16/2012

MORRISON & FOERSTER LLP
755 PAGE MILL RD
PALO ALTO, CA 94304-1018

EXAMINER	
ART UNIT	PAPER NUMBER

DATE MAILED: 02/16/2012

Please find below and/or attached an Office communication concerning this application or proceeding.

UNITED STATES PATENT AND TRADEMARK OFFICE

Commissioner for Patents
United States Patent and Trademark Office
P.O. Box 1450
Alexandria, VA 22313-1450
www.uspto.gov

DO NOT USE IN PALM PRINTER

(THIRD PARTY REQUESTER'S CORRESPONDENCE ADDRESS)

KING & SPALDING

1180 PEACHTREE STREET , NE

ATLANTA, GA 30309-3521

EX PARTE REEXAMINATION COMMUNICATION TRANSMITTAL FORM

REEXAMINATION CONTROL NO. *90/011,490*.

PATENT NO. *RE38104 ET AL.*.

ART UNIT *3992*.

Enclosed is a copy of the latest communication from the United States Patent and Trademark Office in the above identified *ex parte* reexamination proceeding (37 CFR 1.550(f)).

Where this copy is supplied after the reply by requester, 37 CFR 1.535, or the time for filing a reply has passed, no submission on behalf of the *ex parte* reexamination requester will be acknowledged or considered (37 CFR 1.550(g)).

Office Action in Ex Parte Reexamination	Control No. 90/011,490	Patent Under Reexamination RE38104 ET AL.
	Examiner ERIC B. KISS	Art Unit 3992

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

a☐ Responsive to the communication(s) filed on _____ . b☐ This action is made FINAL.
c☒ A statement under 37 CFR 1.530 has not been received from the patent owner.

A shortened statutory period for response to this action is set to expire 2 month(s) from the mailing date of this letter. Failure to respond within the period for response will result in termination of the proceeding and issuance of an *ex parte* reexamination certificate in accordance with this action. 37 CFR 1.550(d). **EXTENSIONS OF TIME ARE GOVERNED BY 37 CFR 1.550(c).** If the period for response specified above is less than thirty (30) days, a response within the statutory minimum of thirty (30) days will be considered timely.

Part I THE FOLLOWING ATTACHMENT(S) ARE PART OF THIS ACTION:

1. ☐ Notice of References Cited by Examiner, PTO-892.
2. ☒ Information Disclosure Statement, PTO/SB/08.
3. ☐ Interview Summary, PTO-474.
4. ☐ _____.

Part II SUMMARY OF ACTION

1a. ☒ Claims *11-41* are subject to reexamination.
1b. ☐ Claims _____ are not subject to reexamination.
2. ☐ Claims _____ have been canceled in the present reexamination proceeding.
3. ☐ Claims _____ are patentable and/or confirmed.
4. ☒ Claims *11-41* are rejected.
5. ☐ Claims _____ are objected to.
6. ☐ The drawings, filed on _____ are acceptable.
7. ☐ The proposed drawing correction, filed on _____ has been (7a)☐ approved (7b)☐ disapproved.
8. ☐ Acknowledgment is made of the priority claim under 35 U.S.C. § 119(a)-(d) or (f).
   a)☐ All b)☐ Some* c)☐ None of the certified copies have
   1☐ been received.
   2☐ not been received.
   3☐ been filed in Application No. _____ .
   4☐ been filed in reexamination Control No. _____.
   5☐ been received by the International Bureau in PCT application No. _____.
   * See the attached detailed Office action for a list of the certified copies not received.
9. ☐ Since the proceeding appears to be in condition for issuance of an *ex parte* reexamination certificate except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte* Quayle, 1935 C.D. 11, 453 O.G. 213.
10. ☐ Other: _____

cc: Requester (if third party requester)

DETAILED ACTION

Claims 11-41 of U.S. Patent RE38,104 are subject to reexamination.

Information Disclosure Statements

The information disclosure statements filed on April 28, November 4, and November 28, 2011, have been given due consideration.

Where patents, publications, and other such items of information are submitted by a party (patent owner or requester) in compliance with the requirements of the rules, the requisite degree of consideration to be given to such information will be normally limited by the degree to which the party filing the information citation has explained the content and relevance of the information. The initials of the examiner placed adjacent to the citations on the form PTO/SB/08A and 08B or its equivalent, without an indication to the contrary in the record, do not signify that the information has been considered by the examiner any further than to the extent noted above.

Patents and Printed Publications Cited in the Request

The request cites the following prior art patents and printed publications:

1. Gries, David, *Compiler Construction for Digital Computers*, (John Wiley & Sons, Inc., 1971), (hereinafter "Gries").
2. U.S. Patent 4,571,678 (Chaitin).
3. Gabriel, Richard P., *Performance and Evaluation of Lisp Systems*, (MIT Press, 1985), (hereinafter "Gabriel").

The request proposes that the *Gries* reference anticipates claims 11-41 of the '104 patent under 35 U.S.C. § 102(b), (Request at 15).

Claims 27-32 of the '104 patent require generating a set of new instructions for the program that contain numeric references resulting from invocation of a routine to resolve any symbolic data references in the set of original instructions. Claims 30-32 of the '104 patent require replacing each instruction in the program with a symbolic data reference with a new instruction containing a numeric reference resulting from invocation of a dynamic field reference routine to resolve the symbolic data reference.

Although *Gries* does disclose invocation of a routine to resolve symbolic references into numerical references (see the rejection of claims 11-26 and 33-41 below), *Gries* does not disclose generating a new instruction set or replacing instructions as a result of the resolution of symbolic references. Instead, *Gries* discloses reading instructions from a program stack P and manipulating the symbolic references in a separate stack S in the interpreter, without changing the program stored in P.

The request proposes that the '678 patent anticipates claims 11-41 of the '104 patent under § 102(b), (Request at 16).

Claims 12 of the '104 patent requires interpreting intermediate form instructions in accordance with a program execution control. Claims 24-26 of the '104 patent require determining immediately prior to execution whether a bytecode of the program contains a symbolic data reference.

Although the '678 patent teaches the translation of high-level source code to intermediate form object code (bytecode) and suggests eventual execution of code (see the rejection of claims 11, 13-23, and 27-41 below), its teachings are directed to compilers rather than interpreters.

The request proposes that the *Gabriel* reference anticipates claims 11-41 of the '104 patent under § 102(b), (Request at 16).

However, upon review of the claim chart provided with the request appears to inconsistently construe the disclosure of *Gabriel* and fails to provide adequate evidence to support a rejection of any claim of the '104 patent based on *Gabriel*. Specifically, the request construes the interpreted code of Gabriel as corresponding to the claimed intermediate form object code, but points to a portion of Gabriel discussing analysis of function calls from within compiled (not interpreted) code for the feature of resolving symbolic references, (Exhibit 7, p. 1). The examiner disagrees with the requester's conclusion that the sentence, "Calls from compiled functions involve either the same lookup followed by a transfer of control to the code or a simple, machine-specific subroutine call; usually a Lisp will attempt to transform the former into the latter once the function has been looked up," means "[U]sually a Lisp will attempt to transform the [interpreted call] into the [a compiled call] once the function has been looked up," (*id.*). Instead, a more reasonable reading appears to be that a Lisp will attempt to transform the lookup followed by transfer of control to the code into a machine-specific subroutine call. This would be consistent with the discussion of how transfer tables are updated on pp. 51-52 of *Gabriel*. Additionally, although the request characterizes the link smashing of Gabriel as replacing the original instruction, it appears that link smashing involves only changing the address stored in the transfer table entry, *i.e.*, from a pointer to QLINKER to a pointer to the binary code. *See Gabriel* at 51-52.

Claim Rejections - 35 USC § 102

The following is a quotation of the appropriate paragraphs of 35 U.S.C. 102 that form the basis for the rejections under this section made in this Office action:

A person shall be entitled to a patent unless –

(b) the invention was patented or described in a printed publication in this or a foreign country or in public use or on sale in this country, more than one year prior to the date of application for patent in the United States.

Claims 11-26 and 33-41are rejected under 35 U.S.C. 102(b) as being anticipated by *Gries*.

Pages 328-35 (Ch 16. Interpreters) form the most pertinent section of *Gries* for the purpose of this rejection. Note, however, that *Gries* refers to previous sections for more detailed descriptions of various subject matter and examples necessary to understand the discussion in this section. Pages 213-43 (Chapters 9 and 10) describe the organization and content of symbol tables. Pages 245-52 (Chapter 11) describe Polish notation and how it can be represented inside a computer. Compare *Gries* Fig. 11.1 (source program), Fig. 11.4 (Polish form), and Fig. 11.5 (internal representation). Further, *Gries* describes chapters 9, 10, 11, and 16 as forming a sequential unit. *Gries* at ix.

Accordingly, although portions of multiple chapters of the *Gries* reference are cited in the rejection below, care has been taken to ensure that the cited elements are still "arranged as in the claim" as required for anticipation. *Net Money IN. Inc. v. Verisign, Inc.*, 545 F.3d 1359, 1369 (Fed. Cir. 2008).

The following claim chart provides a comparison of the features of *Gries* with the features of the claimed invention.

'104 Patent	***Gries***
11. An apparatus comprising: a memory containing intermediate form object code constituted by a set of instructions, certain of said instructions containing one or more symbolic references; and a processor configured to execute said instructions containing one or more symbolic references by determining a numerical reference corresponding to said symbolic reference, storing said numerical references, and obtaining data in accordance to said numerical references.	"We use the term interpreter for a program which performs two functions: 1. Translates a source program written in the source language (e.g. ALGOL) into an internal form; and 2. Executes (interprets, or simulates) the program in this internal form. "The first part of the interpreter is like the first part of a multi-pass compiler, and we will call it the 'compiler'. The internal form into which it translates should be designed to make the second part, the interpreter proper, as efficient as possible. Polish notation is often used here, and this is what we will describe." Gries at 328. "Internally, the Polish form of the source program is stored in an integer array P. During interpretation, an integer p, initially 1, contains the index in P of the symbol currently being processed. Thus, p is an 'instruction counter'. As described in section 11.2, to execute the Polish program we use a stack S with counter i. Initially, the stack is empty (i = 0)." *Id.* The *Polish notation internal form* disclosed by Gries corresponds to the claimed *intermediate form object code*. Gries further describes symbolic references in the intermediate form object code. For example, in the Polish form code in Fig. 11.4, I, J, A, K, and L are symbolic references, and in the internal representation of Fig. 11.4, they are designated by two fields: an indicator (the integer '2') that the word contains an identifier and its symbol table entry addresses. See Gries at 251-52 and 328-29. The *identifiers* disclosed by Gries, represented in the intermediate form code by an identifier substatement and a symbol table entry address, correspond to the claimed *symbolic references*. Gries discloses the execution of such code containing symbolic references as follows:

'104 Patent	*Gries*
	"Each stack element needs two fields, which we call KIND and VALUE. S(i).KIND is 1, 2 or 3, depending on whether S(i).VALUE is an integer value, a symbol table entry address, or the address of a variable. Each operator, when executed, must check its operands on the stack and transform them to the correct KIND, before executing its operation. "For example, the operation * would perform the following steps (its operands are in S(i) and S(i-1): 1. If S(i).KIND = 2, then put the value of the variable described by the symbol table entry at the address contained in S(i).VALUE, into S(i).VALUE. If S(i).KIND = 3, then put the value of the variable at address S(i).VALUE, into S(i).VALUE. 2. Perform the same operation as in (1), but on stack element S(i-1). 3. S(i-1).VALUE := S(i-1).VALUE*S(i).VALUE; S(i-1).KIND := 1. (Perform the multiplication and fix the kind.) 4. i := i-1. (Adjust the stack)." Gries at 330 (emphasis added). In order to obtain the value of the variable as discussed above, the interpreter must determine the runtime address of the corresponding data by referring to the symbol table entry, thus resolving the symbolic references by determining a numerical reference, *i.e.*, the variable's address in memory. This numerical reference must be stored in some form of memory, e.g., RAM or a processor register, in order to be available for use by the interpreter. The storage and execution described by *Gries* inherently require memory and a processor to realize the disclosed functionality. The step of resolving symbolic references has already been addressed as set forth above. However, Gries also discloses an alternative more efficient embodiment of the interpreter that lets the compiler insert conversion operators into the Polish notation, which more explicitly describe the step of *resolving symbolic references*:

'104 Patent	*Gries*
	"Instead, we now allow three possible numbers 2, 3 and 4 as the first location to tell us how to process the identifier, as follows: 2. Put the pointer to the symbol table entry on the stack. We represent this symbolically by P:I. 3. Put the address of the variable on the stack. We represent this by A:I. 4. Put the value of the variable described by the symbol table entry on the stack. We represent this by V:I "For example, the statement C := B+A(I) (C B I A SUBS + := in Polish form) would appear in this more explicit notation as: A:C We need C's address to store into V:B B's value, for addition V:I I's value (the subscript) P:A Pointer to array A's symbol table entry SUBS This produces the address of A(I) on the stack...." Gries at 331.
12. A computer-readable medium containing instructions for controlling a data processing system to perform a method for interpreting intermediate form object code comprised of instructions, certain of said instructions containing one or more symbolic references, said method comprising the steps of: interpreting said instructions in accordance with a program execution control; and resolving a symbolic reference in an instruction being interpreted, said step of resolving said symbolic reference including the substeps of: determining a numerical reference corresponding to said symbolic reference, and storing said numerical reference in a memory.	See the discussion of claim 11 above. Gries discloses the interpreter as an executable program (which in turn executes the internal form program), i.e., interpreting said instructions in accordance with a program execution control. A computer-readable medium, *e.g.*, a memory, is inherent in realizing the disclosed functionality. See Gries at 328.

'104 Patent	***Gries***
13. A computer-implemented method for executing instructions, certain of said instructions containing one or more symbolic references, said method comprising the steps of: resolving a symbolic reference in an instruction, said step of resolving said symbolic reference including the substeps of: determining a numerical reference corresponding to said symbolic reference, and storing said numerical reference in a memory.	See the discussion of claim 11 above, wherein such method steps have already been addressed.
14. The method of claim 13, wherein said substep of storing said numerical reference comprises the substep of replacing said symbolic reference with said numerical reference.	See the discussion of claim 13 above. Gries further discloses replacing said symbolic reference with said numerical reference. See, e.g., Gries at 331 (illustrating processing an identifier, 'C,' by putting C's address on the stack prior to processing an assignment operator).
15. The method of claim 13, wherein said step of resolving said symbolic reference further comprises the substep of executing said instruction containing said symbolic reference using the stored numerical reference.	See the discussion of claim 13 above.
16. The method of claim 13, wherein said step of resolving said symbolic reference further comprises the substep of advancing program execution control after said substep of executing said instruction containing said symbolic reference.	See the discussion of claim 13 above. Gries further discloses advancing program execution control after said substep of executing said instruction containing said symbolic reference. See Gries at 328-29 (describing the use of integer p as an instruction counter and incrementing p after processing a symbol).
17. In a computer system	See the discussion of claim 11 above.

'104 Patent	***Gries***
comprising a program, a method for executing said program comprising the steps of: receiving intermediate form object code for said program with symbolic data references in certain instructions of said intermediate form object code; and converting the instructions of the intermediate form object code having symbolic data references, said converting step comprising the substeps of: resolving said symbolic references to corresponding numerical references, storing said numerical references, and obtaining data in accordance to said numerical references.	Gries further discloses converting the instructions of the intermediate form object code having symbolic data references by resolving the symbolic references to corresponding numerical references, storing the numerical reference, and obtaining data in accordance to the numerical reference. See, e.g., Gries at 331 (illustrating processing an identifier, 'C,' by putting C's address on the stack prior to processing an assignment operator). See also Gries at 330 (as discussed above).
18. A computer-implemented method for executing program operations, each operation being comprised of a set of instructions, certain of said instructions containing one or more symbolic references, said method comprising the steps of: receiving a set of instructions reflecting an operation; and performing the operation corresponding to the received set of instructions, wherein at least one of said symbolic references is resolved by determining a numerical reference corresponding to said symbolic reference, storing said numerical reference, and obtaining data in	See the discussion of claim 11 above.

'104 Patent	***Gries***
accordance to said stored numerical reference.	
19. A memory for use in executing a program by a processor, the memory comprising: intermediate form code containing symbolic field references associated with an intermediate representation of source code for the program, the intermediate representation having been generated by lexically analyzing the source code and parsing output of said lexical analysis, and wherein the symbolic field references are resolved by determining a numerical reference corresponding to said symbolic reference, and storing said numerical reference in a memory.	See the discussion of claim 11 above. Gries further discloses the intermediate representation having been generated by lexically analyzing the source code and parsing output of said lexical analysis: "We use the term interpreter for a program which performs two functions: 1. Translates a source program written in the source language (e.g. ALGOL) into an internal form; and 2. Executes (interprets, or simulates) the program in this internal form. "The first part of the interpreter is like the first part of a multi-pass compiler, and we will call it the 'compiler'. The internal form into which it translates should be designed to make the second part, the interpreter proper, as efficient as possible. Polish notation is often used here, and this is what we will describe." Gries at 328. See also Gries at 4-6 (describing the analysis portion of the compiler, including the scanner, *i.e.*, lexical analyzer, and syntax analyzer, *i.e.*, parser, used to translate source code into an internal representation such as Polish notation). Gries discloses the interpreter as an executable program (which in turn executes the internal form program), i.e., interpreting said instructions in accordance with a program execution control. A computer-readable medium, *e.g.*, a memory, is inherent in realizing the disclosed functionality. See Gries at 328.
20. A computer-implemented method for executing a compiled program containing instructions in an intermediate form code, at least one of the instructions containing a symbolic reference, said method comprising the steps of:	See the discussion of claim 11 above. Gries further discloses performing an operation in accordance with the instruction and data obtained in accordance with the numerical reference without recompiling the program or any portion thereof. See, e.g., Gries at 330-32 (describing interpretation time conversion

'104 Patent	***Gries***
resolving the symbolic reference in the instruction by determining a numerical reference corresponding to the symbolic reference; and performing an operation in accordance with the instruction and data obtained in accordance with the numerical reference without recompiling the program or any portion thereof.	of symbolic references to perform associated operations).
21. A memory encoded with a compiled program, the memory comprising: intermediate form code containing symbolic field references associated with an intermediate representation of source code for the program, the intermediate representation having been generated by lexically analyzing the source code and parsing output of said lexical analysis, such that when the program is executed by a processor each symbolic field reference is resolved by determining a numerical reference corresponding to the symbolic field reference and data is obtained in accordance with the numerical reference without recompiling the program or any portion thereof.	See the discussion of claims 19 and 20 above.
22. An apparatus comprising: a memory containing a compiled program in intermediate form	See the discussion of claim 20 above.

'104 Patent	***Gries***
object code constituted by a set of instructions, at least one of the instructions containing a symbolic reference; and a processor configured to execute the instruction by determining a numerical reference corresponding to the symbolic reference, and performing an operation in accordance with the instruction and data obtained in accordance with the numerical reference without recompiling the program or any portion thereof.	
23. A computer-readable medium containing instructions for controlling a data processing system to perform a method for interpreting a compiled program in intermediate form object code comprised of instructions, at least one of the instructions containing a symbolic reference, said method comprising the steps of: resolving the symbolic reference in the instruction by determining a numerical reference corresponding to the symbolic reference; and performing an operation in accordance with the instruction and data obtained in accordance with the numerical reference without recompiling the program or any portion thereof.	See the discussion of claim 20 above. Gries discloses the interpreter as an executable program (which in turn executes the internal form program), i.e., interpreting said instructions in accordance with a program execution control. A computer-readable medium, *e.g.*, a memory, is inherent in realizing the disclosed functionality. See Gries at 328.
24. A computer-implemented method for executing a program comprised of bytecodes, the method comprising:	See the discussion of claim 11 above. The *Polish notation internal form* disclosed by Gries corresponds to the claimed *program comprised of*

'104 Patent	*Gries*
determining immediately prior to execution whether a bytecode of the program contains a symbolic data reference; when it is determined that the bytecode of the program contains a symbolic data reference, invoking a dynamic field reference routine to resolve the symbolic data reference; and executing thereafter the bytecode using stored data located using a numeric reference resulting from the resolution of the symbolic reference.	*bytecodes*. *E.g.*, Gries at 328. Gries further discloses determining immediate prior to execution whether a bytecode of the program contains a symbolic reference: "Each stack element needs two fields, which we call KIND and VALUE. S(i).KIND is 1, 2 or 3, depending on whether S(i).VALUE is an integer value, a symbol table entry address, or the address of a variable. Each operator, when executed, must check its operands on the stack and transform them to the correct KIND, before executing its operation." Gries at 330. Gries further discloses invoking a dynamic field reference routine to resolve the symbolic data reference and executing thereafter the bytecode using stored data located using a numeric reference resulting from the resolution of the symbolic reference: "For example, the operation * would perform the following steps (its operands are in S(i) and S(i-1): 1. If S(i).KIND = 2, then put the value of the variable described by the symbol table entry at the address contained in S(i).VALUE, into S(i).VALUE. If S(i).KIND = 3, then put the value of the variable at address S(i).VALUE, into S(i).VALUE. 2. Perform the same operation as in (1), but on stack element S(i-1). 3. S(i-1).VALUE := S(i-1).VALUE*S(i).VALUE; S(i-1).KIND := 1. (Perform the multiplication and fix the kind.) 4. i := i-1. (Adjust the stack)." Gries at 330 (emphasis added). In order to obtain the value of the variable as discussed above, the interpreter must determine the runtime address of the corresponding data by referring to the symbol table entry, thus resolving the symbolic references by determining a numerical reference, *i.e.*, the variable's address in memory. This numerical reference must be stored in some form of memory, e.g., RAM or a processor register, in order to be available for use by the interpreter.

'104 Patent	***Gries***
	The storage and execution described by *Gries* inherently require memory and a processor to realize the disclosed functionality.
25. A data processing system, comprising: a processor; and a memory comprising a program comprised of bytecodes and instructions for causing the processor to (i) determine immediately prior to execution of the program whether a bytecode of the program contains a symbolic data reference, (ii) when it is determined that the bytecode of the program contains a symbolic data reference, invoke a dynamic field reference routine to resolve the symbolic data reference, and (iii) execute thereafter the bytecode using stored data located using a numeric reference resulting from the resolution of the symbolic reference.	See the discussion of claim 24 above.
26. A computer program product containing instructions for causing a computer to perform a method for executing a program comprised of bytecodes, the method comprising: determining immediately prior to execution whether a bytecode of the program contains a symbolic data	See the discussion of claim 24 above. Gries discloses the interpreter as an executable program (which in turn executes the internal form program), i.e., interpreting said instructions in accordance with a program execution control. A computer program product, *e.g.*, a memory containing instructions, is inherent in realizing the disclosed functionality. See Gries at 328.

'104 Patent	***Gries***
reference; when it is determined that the bytecode of the program contains a symbolic data reference, invoking a dynamic field reference routine to resolve the symbolic data reference; and executing thereafter the bytecode using stored data located using a numeric reference resulting from the resolution of the symbolic reference.	
33. A computer-implemented method, comprising: receiving a program with a set of instructions written in an intermediate form code; analyzing each instruction of the program to determine whether the instruction contains a symbolic reference to a data object; and executing the program, wherein when it was determined that an instruction contains a symbolic reference, data from a storage location identified by a numeric reference corresponding to the symbolic reference is used thereafter to perform an operation corresponding to that instruction.	See the discussion of claim 11 above. Gries further discloses analyzing each instruction to determine whether the instruction contains a symbolic reference to a data object: "Each stack element needs two fields, which we call KIND and VALUE. S(i).KIND is 1, 2 or 3, depending on whether S(i).VALUE is an integer value, a symbol table entry address, or the address of a variable. Each operator, when executed, must check its operands on the stack and transform them to the correct KIND, before executing its operation." Gries at 330. Gries further discloses using data from a storage location identified by a numeric reference corresponding to the symbolic reference thereafter to perform an operation corresponding to that instruction: "For example, the operation * would perform the following steps (its operands are in S(i) and S(i-1): 1. If S(i).KIND = 2, then put the value of the variable described by the symbol table entry at the address contained in S(i).VALUE, into S(i).VALUE. If S(i).KIND = 3, then put the value of the variable at address S(i).VALUE, into S(i).VALUE. 2. Perform the same operation as in (1), but on stack element S(i-1).

'104 Patent	*Gries*
	3. S(i-1).VALUE := S(i-1).VALUE*S(i).VALUE; S(i-1).KIND := 1. (Perform the multiplication and fix the kind.) 4. i := i-1. (Adjust the stack)." Gries at 330 (emphasis added). In order to obtain the value of the variable as discussed above, the interpreter must determine the runtime address of the corresponding data by referring to the symbol table entry, thus resolving the symbolic references by determining a numerical reference, *i.e.*, the variable's address in memory. This numerical reference must be stored in some form of memory, e.g., RAM or a processor register, in order to be available for use by the interpreter. The storage and execution described by *Gries* inherently require memory and a processor to realize the disclosed functionality.
34. A data processing system, comprising: a processor; and a memory comprising a control program for causing the processor to (i) receive a program with a set of instructions written in an intermediate form code, (ii) analyze each instruction of the program to determine whether the instruction contains a symbolic reference to a data object, and (iii) execute the program, wherein when it was determined that an instruction contains a symbolic reference, data from a storage location identified by a numeric reference corresponding to the symbolic reference is used	See the discussion of claim 33 above.

'104 Patent	***Gries***
thereafter to perform an operation corresponding to that instruction.	
35. A computer program product containing control instructions for causing a computer to perform a method, the method comprising: receiving a program with a set of instructions written in an intermediate form code; analyzing each instruction of the program to determine whether the instruction contains a symbolic reference to a data object; and executing the program, wherein when it was determined that an instruction contains a symbolic reference, data from a storage location identified by a numeric reference corresponding to the symbolic reference is used thereafter to perform an operation corresponding to that instruction.	See the discussion of claim 33 above. Gries discloses the interpreter as an executable program (which in turn executes the internal form program), i.e., interpreting said instructions in accordance with a program execution control. A computer program product, *e.g.*, a memory containing instructions, is inherent in realizing the disclosed functionality. See Gries at 328.
36. A computer-implemented method for executing a program comprised of bytecodes, the method comprising: determining whether a bytecode of the program contains a symbolic reference; when it is determined that the bytecode contains a symbolic reference, invoking a dynamic field reference routine to resolve the symbolic reference; and performing an operation identified	See the discussion of claim 24 above.

'104 Patent	***Gries***
by the bytecode thereafter using data from a storage location identified by a numeric reference resulting from the invocation of the dynamic field reference routine.	
37. A data processing system, comprising: a processor; and a memory comprising a program comprised of bytecodes and instructions for causing the processor to (i) determine whether a bytecode of the program contains a symbolic reference, (ii) when it is determined that the bytecode contains a symbolic reference, invoke a dynamic field reference routine to resolve the symbolic reference, and (iii) perform an operation identified by the bytecode thereafter using data from a storage location identified by a numeric reference resulting from the invocation of the dynamic field reference routine.	See the discussion of claim 24 above.
38. A computer program product containing instructions for causing a computer to perform a method for executing a program comprised of bytecodes, the method comprising: determining whether a bytecode of the program contains a symbolic reference;	See the discussion of claim 24 above. Gries discloses the interpreter as an executable program (which in turn executes the internal form program), i.e., interpreting said instructions in accordance with a program execution control. A computer program product, *e.g.*, a memory containing instructions, is inherent in realizing the disclosed functionality. See Gries at 328.

'104 Patent	***Gries***
when it is determined that the bytecode contains a symbolic reference, invoking a dynamic field reference routine to resolve the symbolic reference; and performing an operation identified by the bytecode thereafter using data from a storage location identified by a numeric reference resulting from the invocation of the dynamic field reference routine.	
39. A computer-implemented method comprising: receiving a program formed of instructions written in an intermediate form code compiled from source code; analyzing each instruction to determine whether it contains a symbolic field reference; and executing the program by performing an operation identified by each instruction, wherein data from a storage location identified by a numeric reference is thereafter used for the operation when the instruction contains a symbolic field reference, the numeric reference having been resolved from the symbolic field reference.	See the discussion of claim 33 above. Gries further discloses the intermediate form code being compiled from source code. Gries at 328.
40. A data processing system, comprising: a processor; and a memory comprising a control program for causing the processor	See the discussion of claim 39 above. The storage and execution described by *Gries* inherently require memory and a processor to realize the disclosed functionality.

'104 Patent	***Gries***
to (i) receive a program formed of instructions written in an intermediate form code compiled from source code, (ii) analyze each instruction to determine whether it contains a symbolic field reference, and (iii) execute the program by performing an operation identified by each instruction, wherein data from a storage location identified by a numeric reference is thereafter used for the operation when the instruction contains a symbolic field reference, the numeric reference having been resolved from the symbolic field reference.	
41. A computer program product containing control instructions for causing a computer to perform a method, the method comprising: receiving a program formed of instructions written in an intermediate form code compiled from source code; analyzing each instruction to determine whether it contains a symbolic field reference; and executing the program by performing an operation identified by each instruction, wherein data from a storage location identified by a numeric reference is used thereafter for the operation when the instruction contains a symbolic	See the discussion of claim 39 above. Gries discloses the interpreter as an executable program (which in turn executes the internal form program), i.e., interpreting said instructions in accordance with a program execution control. A computer program product, *e.g.*, a memory containing instructions, is inherent in realizing the disclosed functionality. See Gries at 328.

'104 Patent	***Gries***
field reference, the numeric reference having been resolved from the symbolic field reference.	

Claim Rejections - 35 USC § 103

The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

> (a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

Claims 11, 13-23, and 27-41 are rejected under 35 U.S.C. 103(a) as being unpatentable over the Chaitin '678 patent.

The following claim chart provides a comparison of the features of the Chaitin '678 patent with the features of the claimed invention.

'104 Patent	**'678 Patent**
11. An apparatus comprising: a memory containing intermediate form object code constituted by a set of instructions, certain of said instructions containing one or more symbolic references; and a processor configured to execute said instructions containing one or more symbolic references by determining a numerical reference corresponding to said symbolic reference, storing said numerical references, and obtaining data in accordance to said numerical references.	"The register allocation phase of the compiler stands between the optimization phase and the final code assembly and emission phase. When the intermediate or internal language (IL) enters register allocation, it is written assuming a hypothetical target machine having an unlimited number of high-speed general-purpose CPU registers." '678 patent at col. 4, lines 49-55. The *intermediate language code* of the '678 patent corresponds to the claimed *intermediate form object code*. The instructions in the intermediate form code contain *symbolic registers*, which correspond to the claimed *symbolic references*: "It is the responsibility of the register allocation phase to map the unlimited number of symbolic registers assumed during optimization into the 32 registers which are actually present in the CPU." '678 patent at col. 4, lines 57-61. A set of new instructions is generated (and stored) that contains *real machine register numbers*, which correspond to the claimed *numeric references*:

'104 Patent	'678 Patent
	"The procedure then continues to block 34 wherein the complete intermediate language program is rewritten replacing all symbolic register in the input intermediate language program by real machine register numbers and the procedure exits via the other arrow marked 'end.'" '678 patent at col. 9, lines 47-52. The '678 patent teaches that the output of the register allocation process is modified intermediate language code, which may be subsequently passed to the final code assembly and emission phase of the optimizing compiler to produce object code. See, *e.g.*, '678 patent at col. 4, lines 49-51; col. 4, lines 5-11 col. 17, lines 21-31. The '678 patent further teaches that the purpose of an optimizing compiler is "to facilitate the use of very high level source program languages on the input side and to, hopefully, assure that the object program produced by the compiler will run on the target CPU in the most efficient manner possible. '678 patent at col. 1, lines 29-36. The '678 patent further teaches that real machine registers are used to obtain data during program execution. See, e.g., '678 patent at col. 1, lines 36-43; col. 4, lines 12-33. In view of the teachings and expressed purpose cited above, it would have been obvious to one of ordinary skill in the art at the time of invention to execute the program using the set of new instructions, thereby obtaining data in accordance to the numerical references, in order to meaningfully enjoy the benefits of the optimized output. The optimizing compiler disclosed in the '678 patent is computer-implemented, and the compilation process and execution of the generated code, as discussed above, inherently require a processor and memory to realize the described functionality. See, *e.g.*, '678 patent at col. 16, line 62, through col. 17, line 11.
13. A computer-implemented method for executing instructions,	See the discussion of claim 11 above, wherein such method steps have already been addressed.

'104 Patent	'678 Patent
certain of said instructions containing one or more symbolic references, said method comprising the steps of: resolving a symbolic reference in an instruction, said step of resolving said symbolic reference including the substeps of: determining a numerical reference corresponding to said symbolic reference, and storing said numerical reference in a memory.	For reasons stated above, such a claim also would have been obvious.
14. The method of claim 13, wherein said substep of storing said numerical reference comprises the substep of replacing said symbolic reference with said numerical reference.	See the discussion of claim 13 above. As discussed above, the '678 patent teaches replacing said symbolic reference with said numerical reference: "The procedure then continues to block 34 wherein the complete intermediate language program is rewritten replacing all symbolic register in the input intermediate language program by real machine register numbers and the procedure exits via the other arrow marked 'end.'" '678 patent at col. 9, lines 47-52. For reasons stated above, such a claim also would have been obvious.
15. The method of claim 13, wherein said step of resolving said symbolic reference further comprises the substep of executing said instruction containing said symbolic reference using the stored numerical reference.	See the discussion of claim 13 above. For reasons stated above, such a claim also would have been obvious.
16. The method of claim 13, wherein said step of resolving said symbolic reference further comprises the substep of advancing program execution control after said substep of executing said	See the discussion of claim 13 above. As discussed above, the '678 patent teaches executing an object program produced by a compiler. '678 patent at col. 1, lines 29-36. The nature of the data-flow analysis performed on the IL program implies conventional

'104 Patent	'678 Patent
instruction containing said symbolic reference.	execution of the program with execution control advancing after each instruction executed. See '678 patent at col. 5, lines 13-20. For reasons stated above, such a claim also would have been obvious.
17. In a computer system comprising a program, a method for executing said program comprising the steps of: receiving intermediate form object code for said program with symbolic data references in certain instructions of said intermediate form object code; and converting the instructions of the intermediate form object code having symbolic data references, said converting step comprising the substeps of: resolving said symbolic references to corresponding numerical references, storing said numerical references, and obtaining data in accordance to said numerical references.	See the discussion of claim 11 above. As discussed above, the '678 patent teaches converting the instructions of the intermediate form object code having symbolic data references by resolving said symbolic references to corresponding numerical references and storing said numerical references: "The procedure then continues to block 34 wherein the complete intermediate language program is rewritten replacing all symbolic register in the input intermediate language program by real machine register numbers and the procedure exits via the other arrow marked 'end.'" '678 patent at col. 9, lines 47-52. For reasons stated above, such a claim also would have been obvious.
18. A computer-implemented method for executing program operations, each operation being comprised of a set of instructions, certain of said instructions containing one or more symbolic references, said method comprising the steps of: receiving a set of instructions reflecting an operation; and performing the operation corresponding to the received set of	See the discussion of claim 11 above. For reasons stated above, such a claim also would have been obvious.

'104 Patent	'678 Patent
instructions, wherein at least one of said symbolic references is resolved by determining a numerical reference corresponding to said symbolic reference, storing said numerical reference, and obtaining data in accordance to said stored numerical reference.	
19. A memory for use in executing a program by a processor, the memory comprising: intermediate form code containing symbolic field references associated with an intermediate representation of source code for the program, the intermediate representation having been generated by lexically analyzing the source code and parsing output of said lexical analysis, and wherein the symbolic field references are resolved by determining a numerical reference corresponding to said symbolic reference, and storing said numerical reference in a memory.	See the discussion of claim 11 above. As discussed above, the '678 patent teaches resolving said symbolic references to corresponding numerical references and storing said numerical references: "The procedure then continues to block 34 wherein the complete intermediate language program is rewritten replacing all symbolic register in the input intermediate language program by real machine register numbers and the procedure exits via the other arrow marked 'end.'" '678 patent at col. 9, lines 47-52. The '678 patent further teaches the intermediate representation having been generated by lexically analyzing the source code and parsing output of said lexical analysis. Specifically, the '678 patent teaches, within an optimizing compiler that converts a high-level source language program into a machine-executable program, a register allocation phase that stands between the optimization phase and the final code assembly and emission phase, *i.e.*, after the lexical analysis and parsing phases as well, and receives the intermediate code as its input. See '678 patent at col. 4, lines 38-55. The '678 patent further teaches the intermediate language utilized being of a type that is conventional to optimizing compilers. '678 patent at col. 5, lines 22-30; col. 16, line 62, through col. 17, line 9. For reasons stated above, such a claim also would have been obvious.

'104 Patent	'678 Patent
20. A computer-implemented method for executing a compiled program containing instructions in an intermediate form code, at least one of the instructions containing a symbolic reference, said method comprising the steps of: resolving the symbolic reference in the instruction by determining a numerical reference corresponding to the symbolic reference; and performing an operation in accordance with the instruction and data obtained in accordance with the numerical reference without recompiling the program or any portion thereof.	See the discussion of claim 11. The performing an operation in accordance with the instruction and data obtained in accordance with the numerical reference would take place once the compiled program of the '678 patent is executed and any register assignments and spill code would be used in the form outputted by the final assembly and emission phase, i.e., after all compilation is finished. See '678 patent at col. 4, line 49, through col. 5, line 4. For reasons stated above, such a claim also would have been obvious.
21. A memory encoded with a compiled program, the memory comprising: intermediate form code containing symbolic field references associated with an intermediate representation of source code for the program, the intermediate representation having been generated by lexically analyzing the source code and parsing output of said lexical analysis, such that when the program is executed by a processor each symbolic field reference is resolved by determining a numerical reference corresponding to the symbolic field reference and data is obtained in accordance with the numerical reference without	See the discussion of claims 19 and 20 above. For reasons stated above, such a claim also would have been obvious.

'104 Patent	'678 Patent
recompiling the program or any portion thereof.	
22. An apparatus comprising: a memory containing a compiled program in intermediate form object code constituted by a set of instructions, at least one of the instructions containing a symbolic reference; and a processor configured to execute the instruction by determining a numerical reference corresponding to the symbolic reference, and performing an operation in accordance with the instruction and data obtained in accordance with the numerical reference without recompiling the program or any portion thereof.	See the discussion of claim 20 above. As noted above, the optimizing compiler disclosed in the '678 patent is computer-implemented, and the compilation process and execution of the generated code, as discussed above, inherently require a processor and memory to realize the described functionality. See, *e.g.*, '678 patent at col. 16, line 62, through col. 17, line 11. For reasons stated above, such a claim also would have been obvious.
23. A computer-readable medium containing instructions for controlling a data processing system to perform a method for interpreting a compiled program in intermediate form object code comprised of instructions, at least one of the instructions containing a symbolic reference, said method comprising the steps of: resolving the symbolic reference in the instruction by determining a numerical reference corresponding to the symbolic reference; and performing an operation in accordance with the instruction and data obtained in accordance with the numerical reference without	See the discussion of claim 20 above. A computer-readable medium, *e.g.*, a memory containing instructions, is inherent in realizing the disclosed functionality discussed above. For reasons stated above, such a claim also would have been obvious.

'104 Patent	'678 Patent
recompiling the program or any portion thereof.	
27. A computer-implemented method comprising: receiving a program with a set of original instructions written in an intermediate form code; generating a set of new instructions for the program that contain numeric references resulting from invocation of a routine to resolve any symbolic data references in the set of original instructions; and executing the program using the set of new instructions.	"The register allocation phase of the compiler stands between the optimization phase and the final code assembly and emission phase. When the intermediate or internal language (IL) enters register allocation, it is written assuming a hypothetical target machine having an unlimited number of high-speed general-purpose CPU registers." '678 patent at col. 4, lines 49-55. The *intermediate language code* of the '678 patent corresponds to the claimed *intermediate form code*. It is *received* by the register allocation portion of the optimizing compiler. The original instructions in the intermediate form code contain *symbolic registers*, which correspond to the claimed *symbolic data references*: "It is the responsibility of the register allocation phase to map the unlimited number of symbolic registers assumed during optimization into the 32 registers which are actually present in the CPU." '678 patent at col. 4, lines 57-61. A set of new instructions is generated that contains *real machine register numbers*, which correspond to the claimed *numeric references*: "The procedure then continues to block 34 wherein the complete intermediate language program is rewritten replacing all symbolic register in the input intermediate language program by real machine register numbers and the procedure exits via the other arrow marked 'end.'" '678 patent at col. 9, lines 47-52. The register allocation process illustrated in Figs. 3 and 4 of the '678 patent correspond to the claimed *routine to resolve any symbolic data references in the set of original references*. The optimizing compiler disclosed in the '678 patent is

'104 Patent	'678 Patent
	computer-implemented. See, *e.g.*, '678 patent at col. 16, line 62, through col. 17, line 11. The '678 patent teaches that the output of the register allocation process is modified intermediate language code, which may be subsequently passed to the final code assembly and emission phase of the optimizing compiler to produce object code. See, *e.g.*, '678 patent at col. 4, lines 49-51; col. 4, lines 5-11 col. 17, lines 21-31. The '678 patent further teaches that the purpose of an optimizing compiler is "to facilitate the use of very high level source program languages on the input side and to, hopefully, assure that the object program produced by the compiler will run on the target CPU in the most efficient manner possible. '678 patent at col. 1, lines 29-36. In view of the teachings and expressed purpose cited above, it would have been obvious to one of ordinary skill in the art at the time of invention to execute the program using the set of new instructions in order to meaningfully enjoy the benefits of the optimized output.
28. A data processing system, comprising: a processor; and a memory comprising a control program for causing the processor to (i) receive a program with a set of original instructions written in an intermediate form code, (ii) generate a set of new instructions for the program that contain numeric references resulting from invocation of a routine to resolve any symbolic data references in the set of original instructions, and	See the discussion of claim 27 above. The compilation process and execution of the generated code, as discussed above, inherently require a processor and memory to realize the described functionality. For reasons stated above, such a claim also would have been obvious.

'104 Patent	'678 Patent
(iii) executing the program using the set of new instructions.	
29. A computer program product containing instructions for causing a computer to perform a method, the method comprising: receiving a program with a set of original instructions written in an intermediate form code; generating a set of new instructions for the program that contain numeric references resulting from invocation of a routine to resolve any symbolic data references in the set of original instructions; and executing the program using the set of new instructions.	See the discussion of claim 27 above. A computer program product, *e.g.*, a memory containing instructions, is inherent in realizing the disclosed functionality discussed above. For reasons stated above, such a claim also would have been obvious.
30. A computer-implemented method comprising: receiving a program that comprises a set of instructions written in an intermediate form code; replacing each instruction in the program with a symbolic data reference with a new instruction containing a numeric reference resulting from invocation of a dynamic field reference routine to resolve the symbolic data reference; and executing the program by performing an operation in accordance with each instruction or	"The register allocation phase of the compiler stands between the optimization phase and the final code assembly and emission phase. When the intermediate or internal language (IL) enters register allocation, it is written assuming a hypothetical target machine having an unlimited number of high-speed general-purpose CPU registers." '678 patent at col. 4, lines 49-55. The *intermediate language code* of the '678 patent corresponds to the claimed *intermediate form code*. It is *received* by the register allocation portion of the optimizing compiler. The original instructions in the intermediate form code contain *symbolic registers*, which correspond to the claimed *symbolic data references*: "It is the responsibility of the register allocation phase to map the unlimited number of symbolic registers assumed

'104 Patent	'678 Patent
new instruction, depending upon whether an instruction has been replaced with a new instruction in accordance with the replacing step.	during optimization into the 32 registers which are actually present in the CPU." '678 patent at col. 4, lines 57-61. The instructions containing symbolic references are replaced with instructions that contain *real machine register numbers*, which correspond to the claimed *numeric references*: "The procedure then continues to block 34 wherein the complete intermediate language program is rewritten replacing all symbolic register in the input intermediate language program by real machine register numbers and the procedure exits via the other arrow marked 'end.'" '678 patent at col. 9, lines 47-52. The register allocation process illustrated in Figs. 3 and 4 of the '678 patent correspond to the claimed *dynamic field reference routine to resolve the symbolic data reference.* The optimizing compiler disclosed in the '678 patent is computer-implemented. See, *e.g.*, '678 patent at col. 16, line 62, through col. 17, line 11. The '678 patent teaches that the output of the register allocation process is modified intermediate language code, which may be subsequently passed to the final code assembly and emission phase of the optimizing compiler to produce object code. See, *e.g.*, '678 patent at col. 4, lines 49-51; col. 4, lines 5-11 col. 17, lines 21-31. The '678 patent further teaches that the purpose of an optimizing compiler is "to facilitate the use of very high level source program languages on the input side and to, hopefully, assure that the object program produced by the compiler will run on the target CPU in the most efficient manner possible. '678 patent at col. 1, lines 29-36. In view of the teachings and expressed purpose cited above, it would have been obvious to one of ordinary skill in the art at the time of invention to execute the program in accordance with the replaced instructions in order to meaningfully enjoy the benefits of the optimized output.

'104 Patent	'678 Patent
31. A data processing system, comprising: a processor; and a memory comprising a control program for causing the processor to (i) receive a program that comprises a set of instructions written in an intermediate form code, (ii) replace each instruction in the program with a symbolic data reference with a new instruction containing a numeric reference resulting from invocation of a dynamic field reference routine to resolve the symbolic data reference, and (iii) execute the program by performing an operation in accordance with each instruction or new instruction, depending upon whether an instruction has been replaced with a new instruction in accordance with the replacing step.	See the discussion of claim 30 above. The compilation process and execution of the generated code, as discussed above, inherently require a processor and memory to realize the described functionality. For reasons stated above, such a claim also would have been obvious.
32. A computer program product containing control instructions for causing a computer to perform a method, the method comprising: receiving a program that comprises a set of instructions written in an intermediate form code; replacing each instruction in the program with a symbolic data reference with a new instruction	See the discussion of claim 30 above. A computer program product, *e.g.*, a memory containing instructions, is inherent in realizing the disclosed functionality discussed above. For reasons stated above, such a claim also would have been obvious.

'104 Patent	'678 Patent
containing a numeric reference resulting from invocation of a dynamic field reference routine to resolve the symbolic data reference; and executing the program by performing an operation in accordance with each instruction or new instruction, depending upon whether an instruction has been replaced with a new instruction in accordance with the replacing step.	
33. A computer-implemented method, comprising: receiving a program with a set of instructions written in an intermediate form code; analyzing each instruction of the program to determine whether the instruction contains a symbolic reference to a data object; and executing the program, wherein when it was determined that an instruction contains a symbolic reference, data from a storage location identified by a numeric reference corresponding to the symbolic reference is used thereafter to perform an operation corresponding to that instruction.	"The register allocation phase of the compiler stands between the optimization phase and the final code assembly and emission phase. When the intermediate or internal language (IL) enters register allocation, it is written assuming a hypothetical target machine having an unlimited number of high-speed general-purpose CPU registers." '678 patent at col. 4, lines 49-55. The *intermediate language code* of the '678 patent corresponds to the claimed *intermediate form code*. It is *received* by the register allocation portion of the optimizing compiler. The instructions in the intermediate form code contain *symbolic registers*, which correspond to the claimed *symbolic references*: "It is the responsibility of the register allocation phase to map the unlimited number of symbolic registers assumed during optimization into the 32 registers which are actually present in the CPU." '678 patent at col. 4, lines 57-61. Each instruction is analyzed to determine whether the instruction contains a symbolic reference to a data object: "The first step in processing the program involves using well-known optimizing compiler techniques to do a global data-flow analysis. Which symbolic registers are live at

'104 Patent	'678 Patent
	each point in the IL program must be known. This is done by indicating at the beginning of each basic block which computations are live going into it, and by marking each operand of each instruction in the IL to indicate if it goes dead." '678 patent at col. 5, lines 13-20. A set of new instructions is generated (and stored) that contains *real machine register numbers*, which correspond to the claimed *numeric references*: "The procedure then continues to block 34 wherein the complete intermediate language program is rewritten replacing all symbolic register in the input intermediate language program by real machine register numbers and the procedure exits via the other arrow marked 'end.'" '678 patent at col. 9, lines 47-52. The '678 patent teaches that the output of the register allocation process is modified intermediate language code, which may be subsequently passed to the final code assembly and emission phase of the optimizing compiler to produce object code. See, *e.g.*, '678 patent at col. 4, lines 49-51; col. 4, lines 5-11 col. 17, lines 21-31. The '678 patent further teaches that the purpose of an optimizing compiler is "to facilitate the use of very high level source program languages on the input side and to, hopefully, assure that the object program produced by the compiler will run on the target CPU in the most efficient manner possible. '678 patent at col. 1, lines 29-36. The '678 patent further teaches that real machine registers are used to obtain data during program execution. See, e.g., '678 patent at col. 1, lines 36-43; col. 4, lines 12-33. In view of the teachings and expressed purpose cited above, it would have been obvious to one of ordinary skill in the art at the time of invention to execute the program using the set of new instructions, thereby obtaining data in accordance to the numerical references, in order to meaningfully enjoy the benefits of the optimized output. The optimizing compiler disclosed in the '678 patent is

'104 Patent	'678 Patent
	computer-implemented, and the compilation process and execution of the generated code, as discussed above, inherently require a processor and memory to realize the described functionality. See, *e.g.*, '678 patent at col. 16, line 62, through col. 17, line 11.
34. A data processing system, comprising: a processor; and a memory comprising a control program for causing the processor to (i) receive a program with a set of instructions written in an intermediate form code, (ii) analyze each instruction of the program to determine whether the instruction contains a symbolic reference to a data object, and (iii) execute the program, wherein when it was determined that an instruction contains a symbolic reference, data from a storage location identified by a numeric reference corresponding to the symbolic reference is used thereafter to perform an operation corresponding to that instruction.	See the discussion of claim 33 above. For reasons stated above, such a claim also would have been obvious.
35. A computer program product containing control instructions for causing a computer to perform a method, the method comprising: receiving a program with a set of instructions written in an intermediate form code;	See the discussion of claim 33 above. A computer program product, *e.g.*, a memory containing instructions, is inherent in realizing the disclosed functionality discussed above. For reasons stated above, such a claim also would have been obvious.

'104 Patent	'678 Patent
analyzing each instruction of the program to determine whether the instruction contains a symbolic reference to a data object; and executing the program, wherein when it was determined that an instruction contains a symbolic reference, data from a storage location identified by a numeric reference corresponding to the symbolic reference is used thereafter to perform an operation corresponding to that instruction.	
36. A computer-implemented method for executing a program comprised of bytecodes, the method comprising: determining whether a bytecode of the program contains a symbolic reference; when it is determined that the bytecode contains a symbolic reference, invoking a dynamic field reference routine to resolve the symbolic reference; and performing an operation identified by the bytecode thereafter using data from a storage location identified by a numeric reference resulting from the invocation of the dynamic field reference routine.	"The register allocation phase of the compiler stands between the optimization phase and the final code assembly and emission phase. When the intermediate or internal language (IL) enters register allocation, it is written assuming a hypothetical target machine having an unlimited number of high-speed general-purpose CPU registers." '678 patent at col. 4, lines 49-55. The *intermediate language code* of the '678 patent corresponds to the claimed *program comprised of bytecodes*. The instructions in the intermediate form code contain *symbolic registers*, which correspond to the claimed *symbolic references*: "It is the responsibility of the register allocation phase to map the unlimited number of symbolic registers assumed during optimization into the 32 registers which are actually present in the CPU." '678 patent at col. 4, lines 57-61. The '678 patent further teaches analyzing each bytecode instruction to determine whether the instruction contains a symbolic reference to a data object: "The first step in processing the program involves using well-known optimizing compiler techniques to do a global

'104 Patent	'678 Patent
	data-flow analysis. Which symbolic registers are live at each point in the IL program must be known. This is done by indicating at the beginning of each basic block which computations are live going into it, and by marking each operand of each instruction in the IL to indicate if it goes dead." '678 patent at col. 5, lines 13-20. The register allocation process illustrated in Figs. 3 and 4 of the '678 patent correspond to the claimed *dynamic field reference routine to resolve the symbolic reference*. A set of new instructions is generated (and stored) that contains *real machine register numbers*, which correspond to the claimed *numeric references*: "The procedure then continues to block 34 wherein the complete intermediate language program is rewritten replacing all symbolic register in the input intermediate language program by real machine register numbers and the procedure exits via the other arrow marked 'end.'" '678 patent at col. 9, lines 47-52. The '678 patent teaches that the output of the register allocation process is modified intermediate language code, which may be subsequently passed to the final code assembly and emission phase of the optimizing compiler to produce object code. See, *e.g.*, '678 patent at col. 4, lines 49-51; col. 4, lines 5-11 col. 17, lines 21-31. The '678 patent further teaches that the purpose of an optimizing compiler is "to facilitate the use of very high level source program languages on the input side and to, hopefully, assure that the object program produced by the compiler will run on the target CPU in the most efficient manner possible. '678 patent at col. 1, lines 29-36. The '678 patent further teaches that real machine registers are used to obtain data during program execution. See, e.g., '678 patent at col. 1, lines 36-43; col. 4, lines 12-33. In view of the teachings and expressed purpose cited above, it would have been obvious to one of ordinary skill in the art at the time of invention to execute the program

'104 Patent	'678 Patent
	using the set of new instructions, obtaining data in accordance to the numerical references, in order to meaningfully enjoy the benefits of the optimized output. The claimed *performing an operation identified by the bytecode* would take place once the compiled program of the '678 patent is executed and any register assignments and spill code would be used in the form outputted by the final assembly and emission phase, i.e., after all compilation is finished. See '678 patent at col. 4, line 49, through col. 5, line 4. Therefore, the execution of the final program would be *thereafter using data from a storage location identified by a numeric reference resulting from the invocation of the dynamic field reference routine.* The optimizing compiler disclosed in the '678 patent is computer-implemented, and the compilation process and execution of the generated code, as discussed above, inherently require a processor and memory to realize the described functionality. See, *e.g.*, '678 patent at col. 16, line 62, through col. 17, line 11.
37. A data processing system, comprising: a processor; and a memory comprising a program comprised of bytecodes and instructions for causing the processor to (i) determine whether a bytecode of the program contains a symbolic reference, (ii) when it is determined that the bytecode contains a symbolic reference, invoke a dynamic field reference routine to resolve the symbolic reference, and	See the discussion of claim 36 above. For reasons stated above, such a claim also would have been obvious.

'104 Patent	'678 Patent
(iii) perform an operation identified by the bytecode thereafter using data from a storage location identified by a numeric reference resulting from the invocation of the dynamic field reference routine.	
38. A computer program product containing instructions for causing a computer to perform a method for executing a program comprised of bytecodes, the method comprising: determining whether a bytecode of the program contains a symbolic reference; when it is determined that the bytecode contains a symbolic reference, invoking a dynamic field reference routine to resolve the symbolic reference; and performing an operation identified by the bytecode thereafter using data from a storage location identified by a numeric reference resulting from the invocation of the dynamic field reference routine.	See the discussion of claim 36 above. A computer program product, *e.g.*, a memory containing instructions, is inherent in realizing the disclosed functionality discussed above. For reasons stated above, such a claim also would have been obvious.
39. A computer-implemented method comprising: receiving a program formed of instructions written in an intermediate form code compiled from source code; analyzing each instruction to determine whether it contains a symbolic field reference; and executing the program by	See the discussion of claim 33 above. The '678 patent further teaches the intermediate form code being compiled from source code. '678 patent at col. 17, lines 3-9. For reasons stated above, such a claim also would have been obvious.

'104 Patent	'678 Patent
performing an operation identified by each instruction, wherein data from a storage location identified by a numeric reference is thereafter used for the operation when the instruction contains a symbolic field reference, the numeric reference having been resolved from the symbolic field reference.	
40. A data processing system, comprising: a processor; and a memory comprising a control program for causing the processor to (i) receive a program formed of instructions written in an intermediate form code compiled from source code, (ii) analyze each instruction to determine whether it contains a symbolic field reference, and (iii) execute the program by performing an operation identified by each instruction, wherein data from a storage location identified by a numeric reference is thereafter used for the operation when the instruction contains a symbolic field reference, the numeric reference having been resolved from the symbolic field reference.	See the discussion of claim 39 above. As noted above, The optimizing compiler disclosed in the '678 patent is computer-implemented, and the compilation process and execution of the generated code, as discussed above, inherently require a processor and memory to realize the described functionality. See, *e.g.*, '678 patent at col. 16, line 62, through col. 17, line 11. For reasons stated above, such a claim also would have been obvious.
41. A computer program product containing control instructions for causing a computer to perform a method, the method comprising:	See the discussion of claim 39 above. A computer program product, *e.g.*, a memory containing instructions, is inherent in realizing the disclosed

'104 Patent	'678 Patent
receiving a program formed of instructions written in an intermediate form code compiled from source code; analyzing each instruction to determine whether it contains a symbolic field reference; and executing the program by performing an operation identified by each instruction, wherein data from a storage location identified by a numeric reference is used thereafter for the operation when the instruction contains a symbolic field reference, the numeric reference having been resolved from the symbolic field reference.	functionality discussed above. For reasons stated above, such a claim also would have been obvious.

Conclusion

In order to ensure full consideration of any amendments, affidavits or declarations, or other documents as evidence of patentability, such documents must be submitted in response to this Office action. Submissions after the next Office action, which is intended to be a final action, will be governed by the requirements of 37 CFR 1.116, after final rejection and 37 CFR 41.33 after appeal, which will be strictly enforced.

Extensions of time under 37 CFR 1.136(a) will not be permitted in these proceedings because the provisions of 37 CFR 1.136 apply only to "an applicant" and not to parties in a

reexamination proceeding. Additionally, 35 U.S.C. 305 requires that reexamination proceedings "will be conducted with special dispatch" (37 CFR 1.550(a)). Extension of time in *ex parte* reexamination proceedings are provided for in 37 CFR 1.550(c).

The patent owner is reminded of the continuing responsibility under 37 CFR 1.565(a) to apprise the Office of any litigation activity, or other prior or concurrent proceeding, involving Patent No. RE38,104 throughout the course of this reexamination proceeding. The third party requester is also reminded of the ability to similarly apprise the Office of any such activity or proceeding throughout the course of this reexamination proceeding. See MPEP §§ 2207, 2282 and 2286.

All correspondence relating to this ex parte reexamination proceeding should be directed:

By Mail to: Mail Stop *Ex Parte* Reexam
Central Reexamination Unit
Commissioner for Patents
United States Patent & Trademark Office
P.O. Box 1450
Alexandria, VA 22313-1450

By FAX to: (571) 273-9900
Central Reexamination Unit

By hand: Customer Service Window
Randolph Building
401 Dulany Street
Alexandria, VA 22314

Registered users of EFS-Web may alternatively submit such correspondence via the electronic filing system EFS-Web, at https://efs.uspto.gov/efile/myportal/efs-registered

Any inquiry concerning this communication should be directed to Central Reexamination Unit at telephone number (571) 272-7705.

/Eric B. Kiss/
Primary Examiner, Art Unit 3992

Conferees: /Mary Steelman/
Primary Examiner, Art Unit 3992

ABK