

UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

LZLABS GMBH

Petitioner

v.

INTERNATIONAL BUSINESS MACHINES CORP.

Patent Owner

Case IPR2023-00274

U.S. Patent No. 8,713,289 B2

DECLARATION OF RICHARD M. GOODIN

TABLE OF CONTENTS

	<u>Page</u>
I. INTRODUCTION	1
II. BASIS OF OPINIONS	1
III. MATERIALS REVIEWED	2
IV. EDUCATION AND EXPERIENCE.....	3
V. LEGAL STANDARDS	6
A. Claim Construction.....	6
B. Obviousness.....	7
VI. SUMMARY OF OPINIONS	9
VII. LEVEL OF ORDINARY SKILL IN THE ART.....	9
VIII. OVERVIEW OF THE '289 PATENT	10
A. Technology Background	10
1. Condition Codes.....	10
2. Branch Conditions.....	12
B. The Specification of the '289 Patent.....	14
C. Claim 1 of the '289 Patent.....	16
D. Prosecution History	19
IX. OVERVIEW OF THE CITED REFERENCES.....	23
A. Loderer (EX1006) And Translation (EX1005).....	23
B. Enterprise Systems Architecture/390 Principles of Operation (EX1008)	29
C. MIPS RISC Architecture (EX1009).....	31

X.	CLAIM CONSTRUCTION	32
XI.	GROUNDS 1 AND 2: INDEPENDENT CLAIMS 1, 11, AND 16 ARE PATENTABLE OVER LODERER ALONE OR IN COMBINATION WITH POO, KANE, AND THE KNOWLEDGE OF A POSITA	33
	A. Petitioner Fails To Demonstrate How The Prior Art Teaches A “Target Machine Condition Code”	33
XII.	GROUNDS 1 AND 2: DEPENDENT CLAIMS 4, 6, AND 10 (GROUND 1) AND DEPENDENT CLAIMS 2-4, 6, 10, 12, AND 17 (GROUND 2) ARE PATENTABLE OVER LODERER ALONE OR IN COMBINATION WITH POO, KANE, AND THE KNOWLEDGE OF A POSITA.....	40
XIII.	DECLARATION IN LIEU OF OATH	41

I. INTRODUCTION

1. I, Richard M. Goodin, declare as follows:
2. I have personal knowledge of all the matters about which I testify in this declaration.
3. I have been retained as an expert in this case on behalf of International Business Machines Corporation (“IBM” or “Patent Owner”) to provide expert opinions and testimony regarding U.S. Patent No. 8,713,289 (the “’289 Patent”).
4. I am being compensated for my work done in connection with this matter at my standard rate plus expenses. My compensation does not depend on the outcome of this case.

II. BASIS OF OPINIONS

5. I have reviewed and considered the documents and other materials listed below in Section III in light of my specialized knowledge provided in my education, training, research, and experience, as summarized in Section IV and described in detail in my CV, which is provided as Exhibit 2002. My analysis of those materials, combined with the specialized knowledge that I have obtained over the course of my education and career, form the bases of my opinions in this declaration.

III. MATERIALS REVIEWED

6. In forming the opinions set forth in this declaration, I have relied upon my experience and knowledge in the relevant fields of art, as well as my review of certain documents in this matter, including:

- Ex. 1001: U.S. Patent No. 8,713,289 (“289 Patent”);
- Ex. 1004: Prosecution history of U.S. Patent No. 8,713,289;
- Petition (Paper 1);
- Institution Decision (“DI”) (Paper 7);
- Ex. 1003: Declaration of Ian Jestice;
- Ex. 1002: Curriculum Vitae of Ian Jestice;
- Ex. 1005: Translation of EP0843256A2 to Loderer (“Loderer”);
- Ex. 1007: U.S. Patent 6,049,864 to Liu (“Liu”);
- Ex. 1008: Enterprise Systems Architecture/390 Principles of Operation (“PoO”);
- Ex. 1009: Kane, G., MIPS RISC Architecture, to Kame (“Kane”);
- Ex. 1024: Hennessey, J. et al., “Hardware/Software Tradeoffs for Increased Performance,” ACM SIGARCH Computer Architecture News, Vol. 10, Issue 2, March 1982 (c) 1982;
- Ex. 2003: Transcript of Mr. Jestice’s 9/13/2023 Deposition;

- Ex. 2004: Hennessy, John L., et al. Computer Architecture: A Quantitative Approach. India, Elsevier Science, 2006 (excerpt);
- Ex. 2005, Pentium™ Processor User's Manual, 1994 (excerpts);
- All other materials cited herein.

IV. EDUCATION AND EXPERIENCE

7. As demonstrated in my professional experience described in my curriculum vitae (attached as Exhibit 2002) and the summary of my qualifications below, I have significant experience in computer science, electrical and computer engineering, and computer systems.

8. I have worked for over 40 years in the field of computer engineering, including work on both hardware and software. I am fluent in many computer languages including C, C++, C#, Objective-C, Swift, Kotlin, Java, Perl, 8086 Assembly, and various flavors of GPU languages. I am also licensed as a Professional Engineer in the state of North Carolina, Registration Number 036347.

9. I graduated from the University of Delaware in 1978 with a degree in electrical engineering with a minor in mechanical engineering. My senior research project set forth an implementation of a computer controlled vehicular suspension system, what has now become the field of "active" suspensions.

10. I am currently the President and Chief Consultant of Goodin & Associates, Inc. I have worked or consulted for Fortune 500 companies such as

Sperry Univac, Sun Microsystems, Apple Computer, Data General, Tektronix, Mitsubishi, and nVidia. I have worked in roles ranging from a Developer to Chief Scientist.

11. My initial focus in college was communication engineering. I continued in that focus for my first two jobs. At Sperry Univac, I wrote communications microcode for a terminal system connected to a mainframe computer. Also at this time, I implemented microcode for a standalone terminal implementing the ISO communications model.

12. At Evans & Sutherland, I wrote communication microcode to integrate the PS300 graphics system into the IBM environment. My work also included writing communications software to emulate the IBM 3270, IBM 3250 and IBM 5080 graphic displays.

13. While working at Sun Microsystems, I architected OpenGL and DirectX firmware for massively parallel MIMD implementation. My responsibilities included development of software architecture, software functional simulator and development of interface software for Verilog simulation. I also co-architected and implemented the Renderman compliant, high-quality rendering software component of Sun's SunVision visualization product.

14. While at Apple Computer, I implemented Intel software drivers for two generations of Intel graphics architectures. I also implemented many small projects in Objective-C.

15. As a consultant for Goodin & Associates, I have implemented hardware and software projects for clients ranging from low-level embedded software and hardware for an Electronic Rebreather to complete Application Program Interfaces for various client's hardware. I have written system simulation software and testing software for design verification. I have also written software applications and designed hardware for systems ranging from desktop computer systems to smartphones. Most apps I created during this time featured some form of networked communication both wired and wireless.

16. I am a Senior Member of the Institute of Electrical and Electronics Engineers (IEEE). The IEEE is a hardware society whose core purpose is to foster technological innovation and excellence for the benefit of humanity. For admission to the grade of Senior Member, a candidate must be an engineer in IEEE-designated fields for a total of 10 years and have demonstrated 5 years of significant performance.

17. I am a Senior Member of the Association for Computing Machinery (ACM). The Association for Computing Machinery is the world's largest educational and scientific computing society. Senior Member is an earned membership grade

awarded to approximately 25% of members who have demonstrated performance that sets them apart from their peers.

V. LEGAL STANDARDS

18. IBM's attorneys have explained to me the legal standards that apply in this case. My understanding of those standards is described below. I am not an attorney. I have used my understanding of the following legal principles set forth in this section in reaching my opinions.

A. Claim Construction

19. I have been informed and I understand that claim construction is a matter of law and the construction of a claim term will be determined by the Board. In this declaration, I have been informed and I understand that the claim terms should be interpreted under "the *Phillips* standard." I have been informed and I understand that under the *Phillips* standard, claim terms are given their plain and ordinary meaning as understood by a person of ordinary skill in the art at the time of the invention in light of the claim language and the patent specification. I have been informed and I understand that the claim terms, as construed, encompass the embodiments disclosed in the specification. I have been informed and I understand that the patentee can serve as their own lexicographer. As such, if the specification provides a definition of a claim term, I understand that I should interpret that claim term in light of the particular definition provided in the specification.

B. Obviousness

20. I understand that a claim is invalid for obviousness if the differences between the invention 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 of ordinary skill in the art (“POSITA”) to which the subject matter pertains. Similarly, I understand that a claim is invalid for obviousness if two or more prior art references in combination disclose, expressly or inherently, every claim limitation so as to render the claim, as a whole, obvious to a POSITA at the time the invention was made.

21. I understand that obviousness is ultimately a question of law based on a determination of a number of factual issues. Those factual issues include: (1) the scope and content of the prior art, (2) the differences between the prior art and the claimed invention as a whole, (3) the level of ordinary skill in the art at the time the invention was made, and (4) objective secondary considerations that reflect the contemporaneous response to the invention at the time, such as commercial success, long-felt need, and failure of others.

22. I understand that the obviousness inquiry takes place at the time of the invention. Therefore, care must be taken to avoid the impermissible use of hindsight in an obviousness analysis. I understand that it is improper to use the invention as a plan or template for hindsight reconstruction of bits and pieces of the prior art to

form the invention. For example, the inventive contribution of a patent may lie in defining a problem in a new way. By merely presenting someone of skill in the art with the identical problem and telling him or her to make the patented invention, it often becomes virtually certain that the artisan will succeed in making the invention. However, the obviousness inquiry must show by clear and convincing evidence that a POSITA at the time of the invention would have recognized the specific problem recognized by the inventor and found it obvious to perform the inventor's methods to solve that problem.

23. It is my further understanding that a patent claim composed of several elements is not proved obvious merely by demonstrating that each of its elements was, independently, known in the prior art. There must be an apparent reason why it would have been obvious to modify the prior art to arrive at the claimed invention. Counsel for IBM has informed me that the patent challenger has the burden to show that a person of ordinary skill in the relevant field had a reason to combine the elements in the manner claimed when asserting obviousness in view of a combination of references.

24. I also understand that conclusory statements are insufficient to support the legal conclusion of obviousness. Instead, I understand that the Petitioner must articulate a basis on which it concludes that it would have been obvious to make the claimed invention.

25. I understand that there is no suggestion or motivation to make a modification to a prior art reference if the proposed modification would render the prior art invention unsatisfactory for its intended purpose. I also understand that an obviousness allegation cannot be supported by a combination of references that would require a substantial reconstruction and redesign of the elements shown in the primary reference as well as a change in the basic principle under which the primary reference was designed to operate. I further understand that a reference qualifies as prior art for an obviousness determination only when it is analogous to the claimed invention, meaning that it is in the field of the inventor's endeavor or if a person of ordinary skill would reasonably have consulted the reference and applied its teachings in seeking a solution to the problem that the inventor was attempting to solve.

VI. SUMMARY OF OPINIONS

26. It is my opinion that claims 1-4, 6, 10-12, and 16-17 (the "challenged claims") of the '289 Patent are not disclosed in the prior art, and are not rendered obvious by the prior art.

VII. LEVEL OF ORDINARY SKILL IN THE ART

27. I have been informed by IBM's attorneys that obviousness is considered from the perspective of a person of ordinary skill in the art at the time of the invention.

28. I understand that the Petition has proposed that a POSITA would have had at least the equivalent of undergraduate degree or equivalent in computer science, electrical or computer engineering or related training in computer systems, or equivalent training from experience working in the field of computer systems. The [POSITA] would also have at least two years of combined engineering, development, or other technical experience related to the development or test of system hardware, software and emulation systems at an assembler or machine code level. *See* Petition, 17.

29. For my declaration, I have applied this definition in my analysis. I met and/or exceeded these requirements for a POSITA at the earliest priority date of the '289 Patent (January 30, 2007).

VIII. OVERVIEW OF THE '289 PATENT

A. Technology Background

1. Condition Codes

30. Condition codes are “special bits” that represent hardware state information used in computer architectures to hold information about the results of various hardware operations. EX2004, 8 (“[S]pecial bits set by ALU operations, possibly under program control.”); EX1005, 2 (“[A] special hardware flag, known as condition code or and usually comprising several bits,...”); EX2003, 76:22-23 (“[T]he condition code is typically implemented in special hardware.”). These “special bits” are set by the outcomes of certain arithmetic and logical operations in

the processor, and particularly by the processor's Arithmetic and Logic Unit (ALU).

EX2004, 8 (“[S]pecial bits set by ALU operations, possibly under program control.”). In conjunction with branch instructions, the condition codes, referred to sometimes as “flags,” control the flow of instruction execution of the processor. For example, in Intel's Pentium processor, condition codes would include, among others:

- “O” the overflow flag. This indicates an overflow occurred as the result of a previous operation.
- “S” the sign flag. This indicates the sign (positive or negative) of a previous operation.
- “Z” the zero flag. This indicates the result of a previous instruction produced a zero output.
- “C” the carry flag. This indicates the previous mathematical operation produced a carry bit for addition or a borrow bit for subtraction.

EX2005, 4.

31. These condition codes had pretty wide acceptance in the industry until the introduction of Reduced Instruction Set Computers (RISC). There were two goals of RISC processor design: The first was to simplify the instruction set to reduce the overall processor gate count. The second, which was somewhat related, was to simplify the overall data paths and produce higher clock rate execution. Condition codes were viewed as unnecessary complications to RISC designs, which replaced the dedicated condition codes with register values. *See* EX1005, 2 (“New computer

generations, especially those with RISC architecture, usually have *no condition code*.”)

32. While condition codes continued to be designed into non-RISC architectures, their implementation was further complicated by having to deal with more than the four basic conditions. The introduction of integrated IEEE 754 floating point units essentially doubled the basic condition codes to account for the vagaries of floating-point arithmetic. The problem was further complicated when processor architectures, in their quest for increased performance, started to feature out of order execution and speculative execution. EX2004, 7.

33. As of the priority date of the '289 Patent, some forward-looking processor architectures had eliminated condition codes altogether as state that persists across multiple instructions in favor of fused instructions which combine the arithmetic and logical operations with the branch operations into single unified compare/operate and branch instructions. EX2004, 8

2. Branch Conditions

34. Using condition codes is one way to specify branch conditions. In “Computer Architecture, A Quantitative Approach,” Hennesey and Patterson explain that “[s]ince most changes in control flow are branches, deciding how to specify the branch condition is important. Figure B.16 shows the three primary techniques in use today and their advantages and disadvantages.” EX2004, 8.

Name	Examples	How condition is tested	Advantages	Disadvantages
Condition code (CC)	80x86, ARM, PowerPC, SPARC, SuperH	Tests special bits set by ALU operations, possibly under program control.	Sometimes condition is set for free.	CC is extra state. Condition codes constrain the ordering of instructions since they pass information from one instruction to a branch.
Condition register	Alpha, MIPS	Tests arbitrary register with the result of a comparison.	Simple.	Uses up a register.
Compare and branch	PA-RISC, VAX	Compare is part of the branch. Often compare is limited to subset.	One instruction rather than two for a branch.	May be too much work per instruction for pipelined execution.

Figure B.16 The major methods for evaluating branch conditions, their advantages, and their disadvantages. Although condition codes can be set by ALU operations that are needed for other purposes, measurements on programs show that this rarely happens. The major implementation problems with condition codes arise when the condition code is set by a large or haphazardly chosen subset of the instructions, rather than being controlled by a bit in the instruction. Computers with compare and branch often limit the set of compares and use a condition register for more complex compares. Often, different techniques are used for branches based on floating-point comparison versus those based on integer comparison. This dichotomy is reasonable since the number of branches that depend on floating-point comparisons is much smaller than the number depending on integer comparisons.

EX2004, 8.

35. As EX2004 discloses, condition codes are “special bits set by ALU operations.” EX2004, 8. Exemplary architectures that implement condition codes include Intel’s 80x86 (e.g., IA32) architecture and Apple-IBM-Motorola’s PowerPC architecture.

36. Condition codes have some disadvantages, however. First, condition codes take up an extra state. EX2004, 8. Second, “condition codes constrain the ordering of instructions since they pass information from one instruction to a branch.” EX2004, 8. Instead of using condition codes, branch conditions can be specified using another technique, which involves the use of a “condition register.” EX2004, 8. Any of the registers available for a particular architecture can be arbitrarily selected as a condition register. EX2004, 8. Condition registers are

simple to implement, compared to condition codes. EX2004, 8. *See also*, EX1024, 4 (“Condition codes are difficult to implement primarily because they are an irregular structure. *As opposed to registers*, where the references of updates of objects are explicit, condition codes are updated as a side-effect.”)

37. Exemplary architectures that implement “condition registers” include Digital Equipment Corporation’s (DEC) Alpha architecture and the MIPS architecture. EX2004, 8.

B. The Specification of the ’289 Patent

38. The ’289 Patent “relates in general to the emulation of computer system architectures, and more particularly, to methods and systems for handling condition code settings encountered in the emulation process.” EX1001, 1:8-11. Specifically, the ’289 Patent “is directed to providing sequences of instructions that produce valid condition code settings without the use of branching instructions from the target architecture.” EX1001, 1:12-14. As the ’289 patent explains, executing operations, such as arithmetic or logical operations, can result in the generation of condition codes, which indicate the outcome status of the instruction execution. EX1001, 1:18-23.

39. The ’298 Patent provides “[a] classic example of an instruction which produces condition code changes upon execution,” namely, “the compare instruction which sets a condition code to ‘zero’ if the operands are equal, to ‘one’ if the first

operand is strictly less than the second operand and to ‘two’ if the first operand is strictly greater than the second operand.” EX1001, 1:29–34. As the ’289 Patent explains, “[i]n conventional approaches to the emulation of a compare instruction, the result is the construction of a sequence of instructions, which include three branch instructions.” EX1001, 2:7–10. According to the ’289 patent, “the presence of branch instructions in the target architecture instruction stream is undesirable.” EX1001, 2:11–12; *see also, id.*, 2:13–32.

40. “In a method for emulating computer instructions from a source machine to produce sequences of instructions on a target machine,” the ’289 patent explains that “*the present invention* generates a sequence of target machine instructions which together operate *to directly calculate target machine condition codes* from carry, sign and overflow codes *without the use of branch instructions from the target machine.*” Ex. 1001, 3:12–18. *See also, id.*, 3:20–22 (“*The present invention* provides specific guiding techniques and several sequences derived from these techniques *to efficiently set conditions codes...*”). The generated target machine instructions produce the same results, when executed on the target architecture. These results include setting the generated condition codes in the target architecture, which can be stored, for example, in a condition code register. EX1001, 1:50–62.

C. Claim 1 of the '289 Patent

41. Petitioner challenges claims 1–4, 6, 10–12, 16, and 17 of the '289 Patent. Claims 1, 11, and 16 are independent. Claim 1, reproduced below, is illustrative of the challenged claims.

1. A computer program product for emulating computer instructions from a source machine to produce sequences of instructions on a target machine, said computer program product comprising:

a non-transitory computer readable storage medium readable by a processor and storing instructions for execution by the processor for performing a method comprising:

obtaining by the target machine a computer instruction from the source machine, said source machine having a different architecture from said target machine; and

generating a sequence of target machine instructions which together operate to derive an encoding for a target machine condition code for the computer instruction, wherein the sequence of target machine instructions provides distinguishing information to distinguish between a plurality of possible outcomes for *the target machine condition code*, and *directly calculates the target machine condition code* without using branch instructions, the directly calculating comprising:

determining an intermediate condition code value, the intermediate condition code value being a provisional value for *the target machine*

condition code and subject to change based on the distinguishing information; and determining, based on the intermediate condition code value and based on the distinguishing information, *the target machine condition code*, wherein at least part of said distinguishing information is separate from the intermediate condition code value.

42. As emphasized above, claim 1 requires a “*target machine condition code*.” Accordingly, the '289 Patent discloses specifying branch conditions in a target architecture that uses “condition codes,” for example, as shown in the first row in the Figure B.16 below.

Name	Examples	How condition is tested	Advantages	Disadvantages
Condition code (CC)	80x86, ARM, PowerPC, SPARC, SuperH	Tests special bits set by ALU operations, possibly under program control.	Sometimes condition is set for free.	CC is extra state. Condition codes constrain the ordering of instructions since they pass information from one instruction to a branch.
Condition register	Alpha, MIPS	Tests arbitrary register with the result of a comparison.	Simple.	Uses up a register.
Compare and branch	PA-RISC, VAX	Compare is part of the branch. Often compare is limited to subset.	One instruction rather than two for a branch.	May be too much work per instruction for pipelined execution.

Figure B.16 The major methods for evaluating branch conditions, their advantages, and their disadvantages. Although condition codes can be set by ALU operations that are needed for other purposes, measurements on programs show that this rarely happens. The major implementation problems with condition codes arise when the condition code is set by a large or haphazardly chosen subset of the instructions, rather than being controlled by a bit in the instruction. Computers with compare and branch often limit the set of compares and use a condition register for more complex compares. Often, different techniques are used for branches based on floating-point comparison versus those based on integer comparison. This dichotomy is reasonable since the number of branches that depend on floating-point comparisons is much smaller than the number depending on integer comparisons.

EX2004, 8.¹

43. Indeed, the '289 Patent discloses that “[t]he present invention provides specific guiding techniques and several sequences derived from these techniques to efficiently set conditions codes or detect exceptional cases in an emulated binary translation environment for the z/Architecture. *These techniques are specifically directed to situations in which the PowerPC architecture and the Intel IA32 architectures are employed to emulate the z/Architecture.*” EX1001, 3:20-26.

44. That is, independent claim 1 relates to emulating program code from a source architecture that supports condition codes to a target architecture that also supports condition codes to specify the branch conditions, and explicitly requires “directly calculat[ing] the target machine condition code.”. EX1001, 9:34-35.

¹ All annotations to figures have been added.

D. Prosecution History

45. The '289 Patent issued from U.S. Application No. 11/668,623, which was filed on January 30, 2007. The '289 Patent went through a rigorous and lengthy examination, which included four rejections and multiple claim amendments, before the claims were allowed over the prior art considered by the Examiner. Specifically, in a first Office Action, dated March 24, 2009 (EX1004, 226), the Examiner rejected the pending claims under multiple grounds, including 35 USC §§101, 102, 103, 112 ¶2. *See* EX1004, 226-237. Regarding the rejections under 35 USC §102, the Examiner rejected independent claim 9, which later issued as patent claim 1, as anticipated by U.S. Patent 6,049,864 to Liu. EX1004, 226-237.

46. In response to the prior art rejections, IBM did not amend the claims but instead argued that Liu “is totally devoid of any reference whatsoever to anything that even remotely resembles emulation.” EX1004, 212 (Response dated June 16, 2009). In a second Office Action dated August 20, 2009 (EX1004, 178), the Examiner maintained the rejections. EX1004, 178-191.

47. In response to the second Office Action, IBM amended the claims and argued—among others—that Liu failed to describe, teach, or suggest the claims as amended. EX1004, 158-170 (Response dated December 21, 2009).

48. In a third Office Action dated March 14, 2012 (EX1004, 133), the Examiner once again maintained the rejections. EX1004, 135-142. In response to

the third Office Action, IBM amended further the claims, for example, as shown below in connection with independent claim 9:

9. (Currently Amended) A computer program product for emulating computer instructions from a source machine to produce sequences of instructions on a target machine, said computer program product comprising:

a non-transitory computer readable storage medium readable by a processor and storing instructions for execution by the processor for performing a method comprising:

obtaining by a target machine a computer instruction from a source machine, said source machine having a different architecture from said target machine; and

generating a sequence of target machine instructions which together operate to derive an encoding for a target machine condition code for the computer instruction, wherein the sequence of target machine instructions ~~directly calculates the condition code and~~ provides distinguishing information to distinguish between a plurality of possible outcomes for the condition code and directly calculates the condition code based, at least in part, on the distinguishing information without using branch instructions, at least part of said distinguishing information being separate from the condition code.

EX1004, 114 (Response dated June 13, 2012).

49. Significantly, IBM distinguished claim 9—as amended—from the teachings of Liu arguing, among others, that Liu did not teach or suggest claim 9 because “*Liu places indications of carry, sign, overflow, etc in an AFLAGS register...*” EX1004, 123. Specifically, IBM argued:

Thus, in accordance with claim 9 presented herewith, a sequence of target machine instructions provides

distinguishing information to distinguish between a plurality of possible outcomes for a condition code and ***directly calculates the condition code***, based, at least in part, on this distinguishing information. Applicants respectfully submit that ***a careful reading of the applied art fails to uncover any such teaching or suggestion. Liu places indications of carry, sign, overflow, etc [sic] in an AFLAGS register*** (see FIG. 3).

EX1004, 123. IBM also argued that Liu also fails to teach or suggest “obtaining by a target machine a computer instruction from a source computer.” EX1004, 123.

50. In a fourth Office Action dated July 31, 2012 (EX1004, 95), the Examiner once again maintained the prior art rejections over Liu. In rejecting the claims, the Examiner noted that transmission of a source instruction to Liu’s machine, while not disclosed by Liu, “is not a patentable feature” (EX1004, 104) and urged IBM to focus on the inventive concept, which, among others, included the condition code. EX1004, 104 (“[T]he examiner recommends focusing more on ***the inventive concept involving the condition code*** and distinguishing information, than on minor details such as how the instruction is obtained.”).

51. In response to the fourth Office Action and the Examiner’s comments, IBM further amended the claims, for example, as shown below in connection with independent claim 9:

9. (Currently Amended) A computer program product for emulating computer instructions from a source machine to produce sequences of instructions on a target machine, said computer program product comprising:

a non-transitory computer readable storage medium readable by a processor and storing instructions for execution by the processor for performing a method comprising:

obtaining by ~~a~~the target machine a computer instruction from ~~a~~the source machine, said source machine having a different architecture from said target machine; and

generating a sequence of target machine instructions which together operate to derive an encoding for a target machine condition code for the computer instruction, wherein the sequence of target machine instructions provides distinguishing information to distinguish between a plurality of possible outcomes for the target machine condition code, and directly calculates the target machine condition code based, at least in part, on the distinguishing information without using branch instructions, the directly calculating comprising:

determining an intermediate condition code value, the intermediate condition code value being a provisional value for the target machine condition code and subject to change based on the distinguishing information; and

determining, based on the intermediate condition code value and based on the distinguishing information, the target machine condition code, wherein at least part of said distinguishing information being is separate from the intermediate condition code value.

EX1004, 74-75. (Response dated October 29, 2012)

52. As indicated by the claim amendments, IBM amended independent claim 9 to recite that the claimed “condition code” is a “*target machine* condition

code.” IBM further argued that “according to *an aspect of applicants’ recited invention*, a *target machine condition code* is calculated without using branch instructions.” EX1004, 83. IBM focused further their arguments on the claimed “intermediate condition code value,” specifically distinguishing Liu in view of it:

[A] careful reading of Liu also fails to teach or suggest determining, based on an intermediate condition code value as recited above and based on the distinguishing information, a target machine condition code, as is further recited in claim 1 [sic] presented herewith.

EX1004, 84.

53. In view of IBM’s claim amendments and comments, the Examiner issued a first Notice of Allowance on August 13, 2013. EX1004, 53. IBM filed an Information Disclosure statement, asking the Examiner to consider a couple of prior art references. EX1004, 28-29. The Examiner issued a second Notice of Allowance on December 12, 2013. EX1004, 7.

IX. OVERVIEW OF THE CITED REFERENCES

A. Loderer (EX1006) And Translation (EX1005)

54. EP0843256A2 (“Loderer” (EX1006)-Translation (EX1005)) is a European Publication that was published on May 20, 1998. EX1005, 1.

55. Loderer discloses “a method for simulating a condition code when porting hardware-specific program code from a source hardware to *a target*

hardware, in which no hardware support of a condition code is provided.”

EX1005, 2. *See also, id.*, Abstract (“The invention relates to a method of simulating a condition code (CC) when porting hardware-specific program code (PCU) from a source hardware (M1) ***to a target hardware (M2) where no hardware support of a condition code (CC) is provided.***”); claim 1.

56. Loderer explains that “[n]ew computer generations, especially those with RISC architecture, usually have ***no condition code.***” EX1005, 2. “As a result, when transforming hardware-specific program code from a source hardware with condition code to program code ***for such a target hardware without condition code,*** a software level conditional code must be simulated.” EX1005, 2.

57. Therefore, Loderer ports program code from a source architecture that supports condition codes to a target architecture “in which ***no hardware support of a condition code is provided.***” EX1005, 2. As Mr. Jestice, LzLabs’ expert, testified, Loderer discloses a target architecture with no condition codes:

Q. ... The invention described in Loderer relates to ***a target hardware that is not providing a condition code, correct?***

A. ***That’s correct.***

EX2003, 89:4-7.

58. Instead, Loderer teaches that “a memory area or a separate register is provided on the target hardware.” EX1005, 2. Accordingly, Loderer discloses

specifying branch conditions in a target architecture that uses a “condition register” as shown in the second row in Figure B.16 below.

Name	Examples	How condition is tested	Advantages	Disadvantages
Condition code (CC)	80x86, ARM, PowerPC, SPARC, SuperH	Tests special bits set by ALU operations, possibly under program control.	Sometimes condition is set for free.	CC is extra state. Condition codes constrain the ordering of instructions since they pass information from one instruction to a branch.
Condition register	Alpha, <u>MIPS</u>	Tests arbitrary register with the result of a comparison.	Simple.	Uses up a register.
Compare and branch	PA-RISC, VAX	Compare is part of the branch. Often compare is limited to subset.	One instruction rather than two for a branch.	May be too much work per instruction for pipelined execution.

Figure B.16 The major methods for evaluating branch conditions, their advantages, and their disadvantages. Although condition codes can be set by ALU operations that are needed for other purposes, measurements on programs show that this rarely happens. The major implementation problems with condition codes arise when the condition code is set by a large or haphazardly chosen subset of the instructions, rather than being controlled by a bit in the instruction. Computers with compare and branch often limit the set of compares and use a condition register for more complex compares. Often, different techniques are used for branches based on floating-point comparison versus those based on integer comparison. This dichotomy is reasonable since the number of branches that depend on floating-point comparisons is much smaller than the number depending on integer comparisons.

EX2004, 8.

59. Indeed, as the Petition admits, “Loderer discloses emulation of condition codes for IBM/390 instructions on a computer *having a MIPS processor*” Petition, 18. The MIPS architecture provides no condition codes. *See* EX1009, 43 (“The R2000² architecture also provides no condition code register containing status bits such as Carry and Overflow.”)

² “The R2000 Processor and R2010 Floating Point Accelerator are the original implementation of the [MIPS RISC] architecture...” EX1009, 2.

60. Accordingly, the target architecture contemplated in Loderer is a MIPS architecture, which implements a “condition register” instead of “condition codes,” as shown in Figure B.16 of EX2004.

61. Because in Loderer there is no target architecture condition code, Loderer teaches a “method for simulating a condition code.” EX1005, 2. Specifically, Loderer discloses that a “mapping is defined which uniquely associates each possible value of the condition code on the source hardware with *a bit pattern* on the target hardware.” EX1005, 2. Loderer stores the bit pattern in a register. *See* EX1005, 2 (“To store the bit pattern associated with the respective current value of the condition code, *a memory area or a separate register* is provided on the target hardware.”).

62. But, as Mr. Jestice admitted, Loderer’s bit pattern is not a register:

Q. Loderer uses the term "condition code on the source hardware," correct?

A. Yes.

Q. Loderer also uses the term "a bit pattern on the target hardware," correct?

A. Yes.

...

Q. So you're saying because the bit pattern in the target architecture uses a single bit for its condition, Loderer chooses to call it a bit pattern instead of a condition code?

A. *It isn't a condition code. It's a bit pattern.*

EX2003, 86:5-87:2.

63. Figure 1 of Loderer “shows schematically the basic principle of the method according to the invention for simulating a condition code.” EX1005, 3.

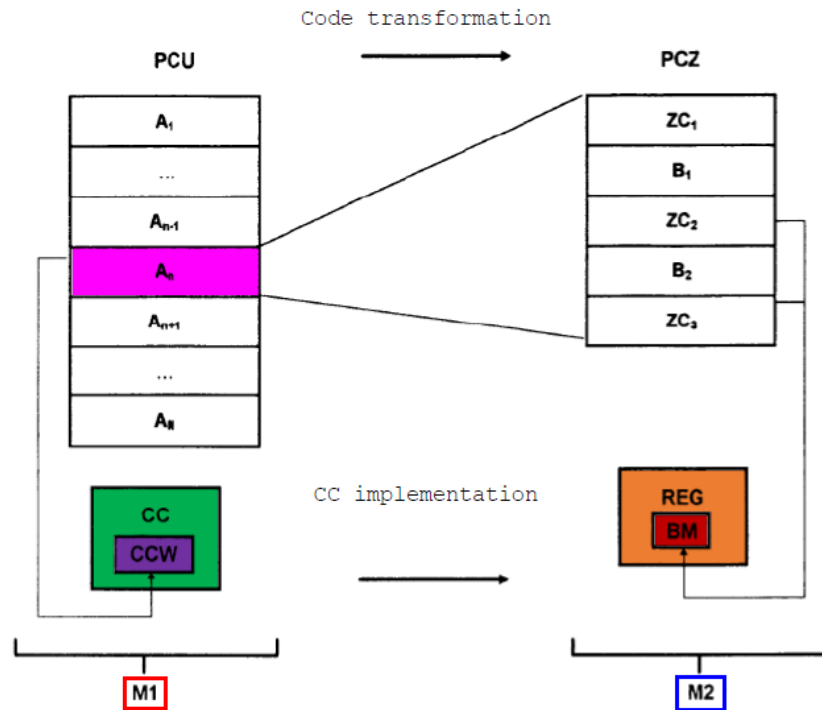


FIG. 1

EX1005, Figure 1.

64. In connection with Figure 1, Loderer discloses that “[a] program code PCU to be ported from a source hardware **M1** to a target hardware **M2** is made up of individual instructions $A_1, A_2, \dots, A_N$. Among these instructions is at least one, e. g. A_n , which influences the value **CCW** of the condition code **CC** provided in the machine architecture of the source hardware **M1**.” EX1005, 3. As Loderer explains, “[i]n the case of computers using a condition code [like source architecture **M1**], this

is almost always the case in practice, since most of the computer commands influence the value *CCW* of the condition code *CC* and the conditional branches which are often required at least in large part rely on the evaluation of the condition code *CC*.” EX1005, 3.

65. “In the machine architecture of the source hardware *M1*, *the condition code CC is typically implemented as a special hardware display.*” EX1005, 3. Because the target architecture *M2* has no condition code, “[t]he condition code must be simulated on the target hardware.” EX1005, 3-4.

66. Loderer discloses that when transforming the program code PCU into “program code PCZ for a target hardware *M2*, each of the instructions A1, A2, ..., AN is implemented as a command (or a semantically equivalent command sequence) in the program code PCZ. For instructions potentially influencing the value *CCW* of the condition code *CC*, additional program code (additional code) is generated *for the simulation of the condition code CC on the target hardware M2.*” EX1005, 3.

67. Loderer also notes that “[i]n addition to the above-mentioned generation of additional code in the code transformation, it is also necessary to provide for a memory area - for performance reasons, *a processor register REG is advantageous - in which the respective current value of the condition code simulated by the additional code is stored in the form of an associated bit pattern BM.*” EX1005, 4.

**B. Enterprise Systems Architecture/390 Principles of Operation
(EX1008)**

68. Enterprise Systems Architecture/390 Principles of Operation (“PoO”) (9th Edition) filed in this proceeding as EX1008, “provides, for reference purposes, a detailed Enterprise Systems Architecture/390* (ESA/390*) description.” EX1008, 21.

69. The ESA/390 system detailed in EX1008 includes two central processing units (CPUs), main and extended storage, a program-status word (PSW), and various registers, such as general registers, floating-point registers, floating-point-control registers, and access registers, a channel subsystem, and input/output (I/O) devices. EX1008, 49-51.

70. In the ESA/390 system, “*the program-status word (PSW) includes the instruction address, **condition code**, and other information used to control instruction sequencing and to determine the state of the CPU.*” EX1008, 51. *See also, id.*, 113:

Program-Status Word. The current program-status word (PSW) in the CPU contains information required for the execution of the currently active program. The PSW is 64 bits in length and includes the instruction address, **condition code**, and other control fields. In general, *the PSW is used to control instruction sequencing and to hold and indicate much of the status of the CPU* in relation to the program currently being executed. Additional control and status

information is contained in control registers and permanently assigned storage locations.

EX1008, 113. *See also, id.*, 941-945.

71. The format of the PSW in the ESA/390 system is shown in Figure 4-2 below:

Program-Status-Word Format

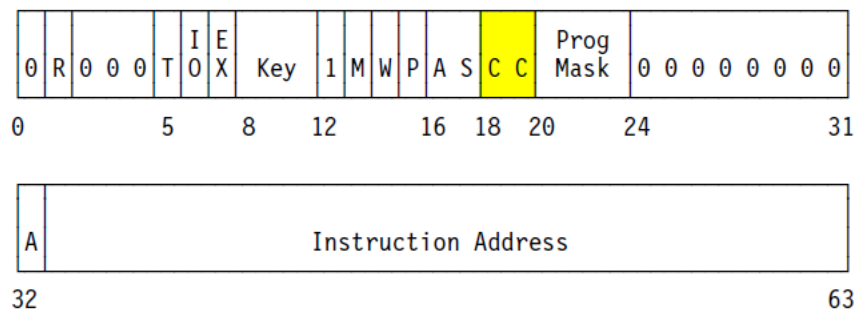


Figure 4-2. PSW Format

EX1008, 115.

72. In the ESA/390 system, bits 18 and 19 of the PSW are the special bits set, for example, by the ALU operations:

Condition Code (CC): *Bits 18 and 19 are the two bits of the condition code.* The condition code is set to 0, 1, 2, or 3, depending on the result obtained in executing certain instructions. *Most arithmetic and logical operations, as well as some other operations, set the condition code.* The instruction BRANCH ON CONDITION can specify any selection of the condition-code values as a criterion for branching. A table in Appendix C summarizes the condition-

code values that may be set for all instructions which set the condition code of the PSW [Program-Status-Word].

EX1008, 116. *See also*, EX2004, 8 (“[S]pecial bits set by ALU operations...”)

C. MIPS RISC Architecture (EX1009)

73. “MIPS RISC Architecture” (“Kane” or EX1009) is a book that describes the MIPS architecture:

This book describes the MIPS RISC Architecture. The R2000 Processor and R2010 Floating Point Accelerator are the original implementation of the architecture and the R2000/R2010 numbering notation is used throughout the text. Higher-performance versions of the two machines, the R3000 and R3010, are the newest implementation of the MIPS RISC architecture. All references in this book to the R2000/R2010 Processors apply fully to the new R3000/R3010 Processors.

EX1009, 2.

74. As already discussed in Section VIII.A.2, the MIPS RISC Architecture does not support condition codes (or a “condition code register”). Rather, in the MIPS RISC Architecture, the conditions generated by the execution of instructions are loaded into a general-purpose register. Specifically, as EX1009 describes:

The R2000 architecture also *provides no condition code register* containing status bits such as *Carry* and *Overflow*. Instead, the conditions generated by the Set instructions (slt, sltu) are *loaded directly into a general-purpose register*. This

approach greatly simplifies handling of the instruction pipeline and eases many compiler tasks but does require that programs explicitly check for conditions such as overflow and carry.

EX1009, 43.

X. CLAIM CONSTRUCTION

75. As I further described in Section V.A, I have been informed that, in my analysis, I should give claim terms their ordinary and customary meaning, as understood by a POSITA at the time of the invention. I was informed and I understand that the specification and file history provides the primary basis for construing the claims, and that dictionaries, encyclopedias, and treatises can also inform the ordinary meaning of a claim term, so long as this extrinsic evidence does not contradict any definition found in or ascertained by a reading of the patent documents. I have been informed that the only exceptions to giving the words in a claim their ordinary meaning are 1) when the applicant acts as his own lexicographer; and 2) when the applicant disavows or disclaims the full scope of a claim term in the specification.

76. For my declaration, I have applied the ordinary and customary meaning of such claim as understood by one of ordinary skill in the art in my analysis provided herein.

XI. GROUNDS 1 AND 2: INDEPENDENT CLAIMS 1, 11, AND 16 ARE PATENTABLE OVER LODERER ALONE OR IN COMBINATION WITH POO, KANE, AND THE KNOWLEDGE OF A POSITA

A. Petitioner Fails To Demonstrate How The Prior Art Teaches A “Target Machine Condition Code”

77. All three independent claims 1, 11, and 16 of the '289 Patent require a “target machine condition code.” EX1001, claims 1, 11, 16. For example, independent claim 1 recites, among others:

[G]enerating a sequence of target machine instructions which together operate to derive *an encoding for a target machine condition code* for the computer instruction, wherein the sequence of target machine instructions provides distinguishing information to distinguish between a plurality of possible outcomes for the *target machine condition code*, and directly calculates the *target machine condition code* without using branch instructions, the directly calculating comprising:

determining an intermediate condition code value, the intermediate condition code value being a provisional value for the *target machine condition code* and subject to change based on the distinguishing information; and

determining, based on the intermediate condition code value and based on the distinguishing information, the *target machine condition code*, wherein at least part of said

distinguishing information is separate from the intermediate condition code value.

EX1001, 9:28-48; *see also, id.*, 11:1-21, 12:4-22.

78. But Loderer does not disclose a “target machine condition code,” as required by independent claims 1, 11, and 16. As discussed in Section IX.A, Loderer discloses “a method for simulating a condition code when porting hardware-specific program code from a source hardware to *a target hardware, in which no hardware support of a condition code is provided.*” EX1005, 2.

79. In fact, Loderer makes clear that “[t]he object of the invention is therefore to specify a method for simulating a condition code when porting hardware-specific program code from a source hardware *to a target hardware that is not providing for condition code...*” EX1005, 2.

80. Moreover, both in the Abstract and the claims, Loderer emphasizes that the target architecture operates *without a condition code*:

The invention relates to a method of simulating a condition code (CC) when porting hardware-specific program code (PCU) from a source hardware (M1) *to a target hardware (M2) where no hardware support of a condition code (CC) is provided.*”

EX1005, Abstract.

A method for simulating a condition code (CC), used for the execution of hardware-specific program code (PCU) for a

source hardware (M1), in the program code (PCZ), obtained following transformation, for a differing target hardware (M2) *with a machine architecture which operates without a condition code (CC).*

EX1005, claim 1.

81. Even Mr. Jestice, LzLabs' expert, admitted that Loderer discloses that the target architecture has no condition codes:

Q. ... The invention described in Loderer relates to *a target hardware that is not providing a condition code, correct?*

A. *That's correct.*

EX2003, 89:4-7

82. Loderer explains that “[n]ew computer generations, especially those with RISC architecture, usually have *no condition code.*” EX1005, 2. “As a result, when transforming hardware-specific program code from a source hardware with condition code to program code *for such a target hardware without condition code,* a software level conditional code must be simulated.” EX1005, 2.

83. Therefore, Loderer ports program code from a source architecture that supports condition codes (e.g., ESA/390 system³) to a target architecture “in which

³ As discussed in Section IX.B, the IBM/390 system includes a special hardware structure, PSW, which has the condition codes.

no hardware support of a condition code is provided.” EX1005, 2. Instead of a “target machine condition code,” Loderer teaches that “a memory area or a separate register is provided on the target hardware.” EX1005, 2. Accordingly, Loderer discloses specifying branch conditions in a target architecture that uses a “condition register,” as shown in the second row in Figure B.16 below.

Name	Examples	How condition is tested	Advantages	Disadvantages
Condition code (CC)	80x86, ARM, PowerPC, SPARC, SuperH	Tests special bits set by ALU operations, possibly under program control.	Sometimes condition is set for free.	CC is extra state. Condition codes constrain the ordering of instructions since they pass information from one instruction to a branch.
Condition register	Alpha, <u>MIPS</u>	Tests arbitrary register with the result of a comparison.	Simple.	Uses up a register.
Compare and branch	PA-RISC, VAX	Compare is part of the branch. Often compare is limited to subset.	One instruction rather than two for a branch.	May be too much work per instruction for pipelined execution.

Figure B.16 The major methods for evaluating branch conditions, their advantages, and their disadvantages. Although condition codes can be set by ALU operations that are needed for other purposes, measurements on programs show that this rarely happens. The major implementation problems with condition codes arise when the condition code is set by a large or haphazardly chosen subset of the instructions, rather than being controlled by a bit in the instruction. Computers with compare and branch often limit the set of compares and use a condition register for more complex compares. Often, different techniques are used for branches based on floating-point comparison versus those based on integer comparison. This dichotomy is reasonable since the number of branches that depend on floating-point comparisons is much smaller than the number depending on integer comparisons.

EX2004, 8.

84. Indeed, as the Petition and Mr. Jestice admit, “Loderer discloses emulation of condition codes for IBM/390 instructions on a computer *having a MIPS processor....*” Petition, 18; EX1003, ¶75. The MIPS architecture provides no condition codes.

85. Accordingly, the target architecture contemplated in Loderer is a MIPS architecture, which implements a “condition register” instead of “condition codes,”

as shown in Figure B.16 of EX2004. *See also*, EX2004, 6 (“There are usually two ways of specifying the branch condition in RISC architectures: with a set of condition bits (sometimes called a condition code) or ***by a limited set of comparisons between a pair of registers or between a register and zero. MIPS uses the latter.***”)

86. Even the evidence submitted by the Petitioner confirms this. As already discussed in Section VIII.A.2, the MIPS RISC Architecture does not support condition codes (or a “condition code register”). Rather, in the MIPS RISC Architecture, the conditions generated by the execution of instructions are loaded into a general-purpose register. Specifically, as EX1009 describes:

The R2000 architecture [MIPS RISC Architecture] also ***provides no condition code register*** containing status bits such as *Carry* and *Overflow*. Instead, the conditions generated by the Set instructions (slt, sltu) are ***loaded directly into a general-purpose register***. This approach greatly simplifies handling of the instruction pipeline and eases many compiler tasks but does require that programs explicitly check for conditions such as overflow and carry.

EX1009, 43.

87. LzLabs maps the claimed “target machine condition code” to Loderer’s “bit pattern” **BM** stored in register **REG**. *See* Petition, 34; EX1003, ¶111.

Q. So it is your opinion that the target machine condition code in Loderer is the bit pattern BM, correct?

A. Right.

EX2003, 99:19-22.

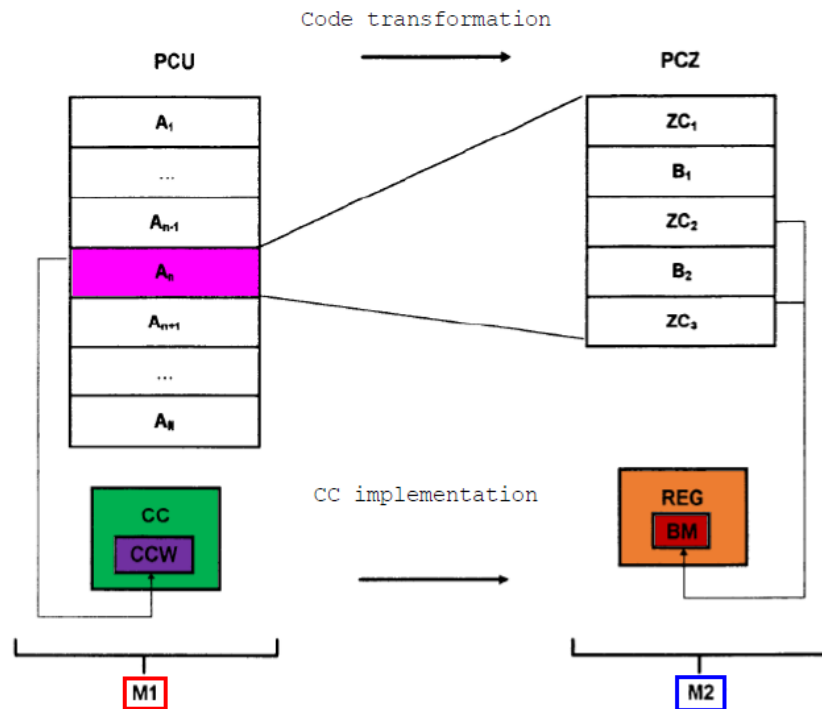


FIG. 1

EX1005, Figure 1.

88. Loderer discloses that an “IBM/390 computer system is selected as source hardware **MI**.” EX1005, 4. As discussed in Section IX.B, in the ESA/390 system, “*the program-status word (PSW) includes the instruction address, condition code*, and other information used to control instruction sequencing and to determine the state of the CPU.” EX1008, 51. That is, Loderer discloses a source machine condition code.

89. Because the target architecture disclosed in Loderer provides no support for a condition code, Loderer teaches instead a “method for simulating a

condition code.” EX1005, 2. Specifically, Loderer discloses that a “mapping is defined which uniquely associates each possible value of the condition code on the source hardware with a bit pattern on the target hardware.” EX1005, 2. Loderer stores the bit pattern in a general-purpose register. *See* EX1005, 2 (“To store the bit pattern associated with the respective current value of the condition code, ***a memory area or a separate register*** is provided on the target hardware.”).

90. But, as Mr. Jestice admitted, Loderer’s bit pattern is not a condition code:

Q. Loderer uses the term “condition code on the source hardware,” correct?

A. Yes.

Q. Loderer also uses the term “a bit pattern on the target hardware,” correct?

A. Yes.

...

Q. So you're saying because the bit pattern in the target architecture uses a single bit for its condition, Loderer chooses to call it a bit pattern instead of a condition code?

A. ***It isn't a condition code. It's a bit pattern.***

EX2003, 86:5-87:2.

91. Loderer’s teaching of a bit pattern stored in a general-purpose register instead of a “target machine condition code” is fatal to LzLabs’ Petition. Indeed, as discussed in Section VIII.D, in the last claim amendments before the ’289 Patent

was allowed, IBM amended the claims to specify that the claimed “condition code” is a “*target machine* condition code.” EX1004, 74-75.

92. Therefore, Loderer neither teaches nor renders obvious the claimed “target machine condition code” of the challenged independent claims 1, 11, and 16. Thus, Petitioner’s Ground 1 fails with regard to the challenged independent claims 1, 11, and 16.

93. In Ground 2, Petitioner combined Loderer with PoO, Kane, and the knowledge of a POSITA. But none of PoO, Kane, or the knowledge of a POSITA is used in Petitioner’s combinations as disclosing or rendering obvious the claimed “target machine condition code.” Thus, Petitioner’s Ground 2 fails with regard to the challenged independent claims 1, 11, and 16, for the same reasons that Ground 1 fails.

XII. GROUNDS 1 AND 2: DEPENDENT CLAIMS 4, 6, AND 10 (GROUND 1) AND DEPENDENT CLAIMS 2-4, 6, 10, 12, AND 17 (GROUND 2) ARE PATENTABLE OVER LODERER ALONE OR IN COMBINATION WITH POO, KANE, AND THE KNOWLEDGE OF A POSITA

94. Challenged dependent claims 2-4, 6, 10, 12, and 17 depend (directly or indirectly) from one of the independent claims 1, 11, and 16. Accordingly, neither Loderer alone nor the combination of Loderer with PoO, Kane, and the knowledge of a POSITA renders obvious the challenged dependent claims for the reasons discussed in Section XI.A, in connection with independent claims 1, 11, and 16.

XIII. DECLARATION IN LIEU OF OATH

95. I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under § 1001 of Title 18 of the United States Code. If called to testify as to the truth of the matters stated herein, I could and would testify competently.

I declare under penalty of perjury that the foregoing is true and correct.



Richard M. Goodin

Date: 09/22/2023