

Case IPR2023-00274
U.S. Patent No. 8,713,289 B2

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. 6,785,289 B2

INTERNATIONAL BUSINESS MACHINES CORPORATION'S
PATENT OWNER'S RESPONSE

Mail Stop Patent Board
Patent Trial and Appeal Board
U.S. Patent and Trademark Office
P.O. Box 1450
Alexandria, VA 22313-1450

TABLE OF CONTENTS

	<u>Page</u>
I. INTRODUCTION	1
II. OVERVIEW OF THE '289 PATENT	2
A. Technology Background	2
1. Condition Codes.....	2
2. Branch Conditions.....	5
B. The '289 Patent Inventions	7
C. Illustrative Claim.....	8
D. Prosecution History	11
III. OVERVIEW OF THE CITED REFERENCES.....	16
A. Loderer (EX1006) And Translation (EX1005).....	16
B. Enterprise Systems Architecture/390 Principles of Operation (EX1008).....	22
C. MIPS RISC Architecture (EX1009).....	25
IV. PERSON OF ORDINARY SKILL IN THE ART (“POSITA”).....	26
V. CLAIM CONSTRUCTION	26
VI. 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	26
A. Petitioner Fails To Demonstrate How The Prior Art Teaches A “Target Machine Condition Code”	26

VII. GROUND 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.....	34
VIII. CONCLUSION.....	34

EXHIBIT LIST

Exhibit No.	DESCRIPTION
2001	Declaration of Richard M. Goodin
2002	Curriculum Vitae of Richard M. Goodin
2003	Transcript of Mr. Jestice's 9/13/2023 Deposition
2004	Hennessy, John L., et al. <i>Computer Architecture: A Quantitative Approach</i> . India, Elsevier Science, 2006 (excerpt)
2005	Pentium™ Processor User's Manual, Volume 3: Architecture and Programming Manual, 1994

All emphasis added, unless specified otherwise.

All annotations have been added, unless specified otherwise.

All objections in deposition transcripts have been removed.

I. INTRODUCTION

LzLabs GMBH (“LzLabs” or Petitioner) fails to prove that any of claims 1-4, 6, 10-12, and 16-17 (the “challenged claims”) of U.S. Patent No. 8,713,289 (the “’289 Patent” (“EX1001”)) patent is unpatentable over the two Grounds presented in the Petition.

The ’289 Patent relates to emulation of computer system architectures, and more particularly, to methods and systems for handling condition code settings encountered in the emulation process. Specifically, the claimed inventions emulate computer instructions from a source machine to generate a sequence of instructions on a target machine, where the sequence of instructions *directly calculates target machine condition codes*, without using branch conditions.

As explained below, during a lengthy examination of the application that issued as the ’289 Patent, International Business Machines Corporation (“IBM” or “Patent Owner”) amended the independent claims of the ’289 Patent to specifically recite that the claimed “condition code” that is calculated by the sequence of instructions is a “*target machine* condition code.”

Accordingly, all challenged claims require a “*target machine* condition code.” But the prior art relied in the Petition, namely Loderer, fails to teach or suggest the claimed “*target machine* condition code.” Indeed, the Abstract, the Specification, and the claims of Loderer confirm that in Loderer the target machine

has no condition codes. “*The 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; Abstract (“[A] target hardware (M2) where no hardware support of a condition code (CC) is provided”); claim 1 (“[A] differing target hardware (M2) with a machine architecture which operates *without a condition code (CC)...*”).

Even Mr. Jestice, LzLabs’ expert, admitted during examination that 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.

This Response, supported by the Declaration of Richard M. Goodin filed herewith as EX2001, demonstrates that Petitioner has failed to show the unpatentability of the challenged claims.

II. OVERVIEW OF THE ’289 PATENT

A. Technology Background

1. Condition Codes

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 control the flow of instruction execution of the processor. For example, in Intel’s Pentium processor, condition codes, referred to sometimes as “flags,” 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. EX2001, ¶30.

These condition codes were accepted fairly widely in the industry until the introduction of Reduced Instruction Set Computers (RISC). The RISC architecture had two goals: 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. “New computer generations, especially those with RISC architecture, usually have *no condition code*.”

EX1005, 2. EX2001, ¶31.

While condition codes continued to be designed into non-RISC architectures, their implementation was further complicated with the addition of new conditions, beyond the four basic conditions. The introduction of integrated IEEE754 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, began to feature out of order execution and speculative execution.

EX2004, 7. EX2001, ¶32.

As of the priority date of the '289 Patent, some forward-looking processor architectures had eliminated condition codes in favor of fused instructions. Fused

instructions combine arithmetic and logical operations with branch operations into single unified compare and branch instructions. EX2004, 8. EX2001, ¶33.

2. Branch Conditions

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. EX2001, ¶34.

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. EX2001, ¶34.

As EX2004 discloses and Mr. Goodin explains, condition codes are “special bits set by ALU operations.” EX2004, 8; EX2001, ¶35. Exemplary architectures that implement condition codes include Intel’s 80x86 (e.g., IA32) architecture and Apple-IBM-Motorola’s PowerPC architecture. EX2001, ¶35.

Condition codes have some disadvantages, however. First, condition codes take up an extra state, e.g., store state information about a previous instruction. 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.”) EX2001, ¶36.

Exemplary architectures that implement “condition registers” include Digital Equipment Corporation’s (DEC) Alpha architecture and the MIPS architecture. EX2004, 8. EX2001, ¶37.

B. The '289 Patent Inventions

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. EX2001, ¶38.

The '289 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. EX2001, ¶39.

“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. EX2001, ¶40.

C. Illustrative Claim

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.

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 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 EX2001, ¶42.

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***

¹ All annotations in figures have been added, unless specified otherwise.

*directed to situations in which the **PowerPC** architecture and the **Intel IA32** architectures are employed to emulate the z/Architecture.”* EX1001, 3:20-26.
EX2001, ¶43.

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.
EX2001, ¶44.

D. Prosecution History

The '289 Patent issued from U.S. Application No. 11/668,623, filed on January 30, 2007. The '289 Patent underwent 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.

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.

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).

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).

Significantly, IBM argued that claim 9—as amended—is distinguished from Liu because, among other reasons, Liu does 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. Additionally, IBM argued that Liu also fails to teach or suggest “obtaining by a target machine a computer instruction from a source computer.”

EX1004, 123. EX2001, ¶49.

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.”). EX2001, ¶50.

In response to the fourth Office Action and the Examiner’s comments, IBM further amended the claims. IBM amended independent claim 9 (and others) to clarify the requirement that the claimed “condition code” is a “*target machine* condition code.” For example, IBM amended independent claim 9 as follows:

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.)

IBM argued that “according to *an aspect of applicants’ recited invention*, a target machine condition code is calculated without using branch instructions.”

EX1004, 83. Indeed, IBM focused on the claimed “intermediate condition code value,” which IBM argued further distinguished the claims from Liu:

[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.

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

III. OVERVIEW OF THE CITED REFERENCES

A. Loderer (EX1006) And Translation (EX1005)

EP0843256A2 (“Loderer” (EX1006)-Translation (EX1005)) is a European Publication that was published on May 20, 1998. EX1005, 1. EX2001, ¶54.

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. EX2001, ¶55.

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 [sic] code must be simulated.” EX1005, 2. EX2001, ¶56.

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. Mr. Jestice, LzLabs’ expert, confirmed during his testimony that 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. EX2001, ¶57.

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

specification of branch conditions in a target architecture that uses a “condition register” as shown in the second row in the Figure 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. EX2001, ¶58.

Indeed, as Petitioner 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.”). Accordingly, the target architecture

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

contemplated in Loderer is a MIPS architecture, which, as shown in Figure B.16 of EX2004, implements a “condition register” instead of “condition codes.” EX2001, ¶60.

Because Loderer lacks a 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.”). EX2001, ¶61.

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

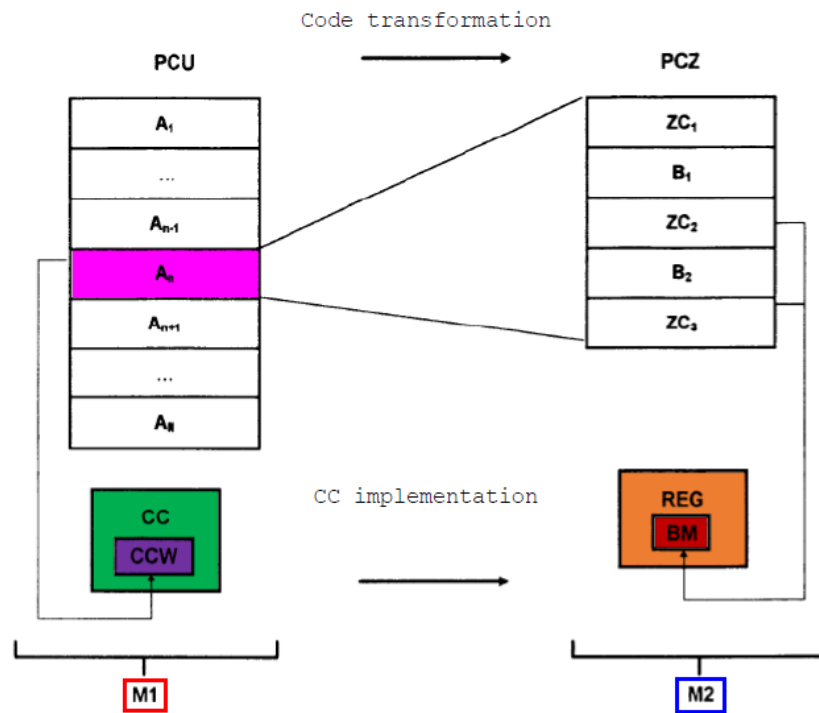


FIG. 1

EX1005, Figure 1. EX2001, ¶63.

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. EX2001, ¶64.

“In the machine architecture of the source hardware **M1**, *the condition code CC is typically implemented as a special hardware display*³.” 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. EX2001, ¶65.

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. EX2001, ¶66.

³ The word “display” is apparently a translation error. As Mr. Jestice confirmed “There isn’t actually a display. I suspect it’s a translation problem.” EX2003, 76:15-17; *see also, id.*, 76:22-25 (“[T]he condition code is typically implemented in special hardware. I think the display is the only redundant. It’s probably incorrect.”)

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. EX2001, ¶67.

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. EX2001, ¶62.

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

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. EX2001, ¶68.

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. EX2001, ¶69.

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. EX2001, ¶70.

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

Program-Status-Word Format

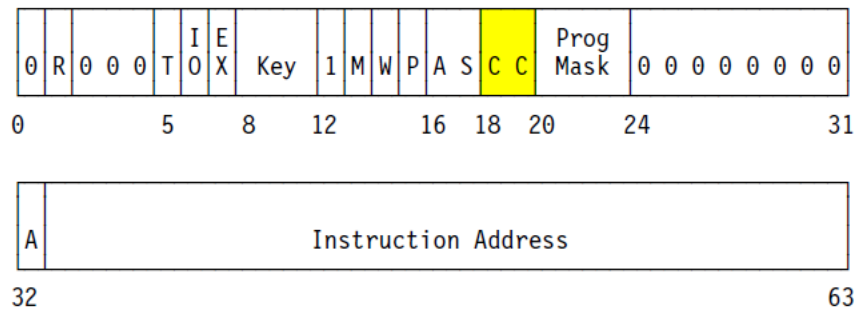


Figure 4-2. PSW Format

EX1008, 115. EX2001, ¶71.

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...”).

EX2003, 61:7-16; 62:18-63:2. EX2001, ¶72.

C. MIPS RISC Architecture (EX1009)

“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. EX2001, ¶73.

As already discussed in Section II.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. EX2001, ¶74.

IV. PERSON OF ORDINARY SKILL IN THE ART (“POSITA”)

For purposes of this proceeding, IBM adopts Petitioner’s proposed level of ordinary skill in the art. Petition, 17. EX2001, ¶¶27-29.

V. CLAIM CONSTRUCTION

In the Institution Decision, the Board determined that no claim terms require express construction. Institution Decision (Paper 7), 7. Solely for purposes of this IPR, IBM adopts the Board’s application of the plain and ordinary meaning of the claim terms. EX2001, ¶¶75-76.

VI. 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”

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 other limitations:

[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. EX2001, ¶77.

But Loderer does not disclose a “target machine condition code,” as required by independent claims 1, 11, and 16. Rather, as discussed in Section III.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. EX2001, ¶78.

In fact, Loderer makes clear that “[t]he ***object of the invention*** is ... 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. EX2001, ¶79.

Moreover, Loderer emphasizes in the Abstract and the claims 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. EX2001, ¶80.

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. EX2001, ¶80.

Even Mr. Jestice, LzLabs’ expert, admitted that Loderer discloses that the target architecture does not provide 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. EX2001, ¶81.

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 [sic] code must be simulated.” EX1005, 2. *See also*, EX1009, 43; 2003, 51:6-52:1; 52:9-14; 81:22-82:2. EX2001, ¶82.

Therefore, Loderer ports program code from a source architecture that supports condition codes (e.g., ESA/390 system⁴) to a target architecture “in which *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 the Figure below. EX2001, ¶83.

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. EX2001, ¶83.

⁴ As discussed in Section III.B, the IBM/390 system includes a special hardware structure, PSW, which has the condition codes. EX2001, ¶83 n.3.

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. EX2001, ¶84.

Accordingly, the target architecture contemplated in Loderer is a MIPS architecture—a type of RISC 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.*”) EX2001, ¶85.

Even Petitioner’s evidence confirms that to be true. As already discussed in Section II.A.2, the MIPS RISC Architecture does not support condition codes (or a “condition code register”). To the contrary, 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. EX2001, ¶86.

LzLabs maps the claimed “target machine condition code” to Loderer’s “bit pattern” **BM** stored in register **REG**.⁵ 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. EX2001, ¶87.

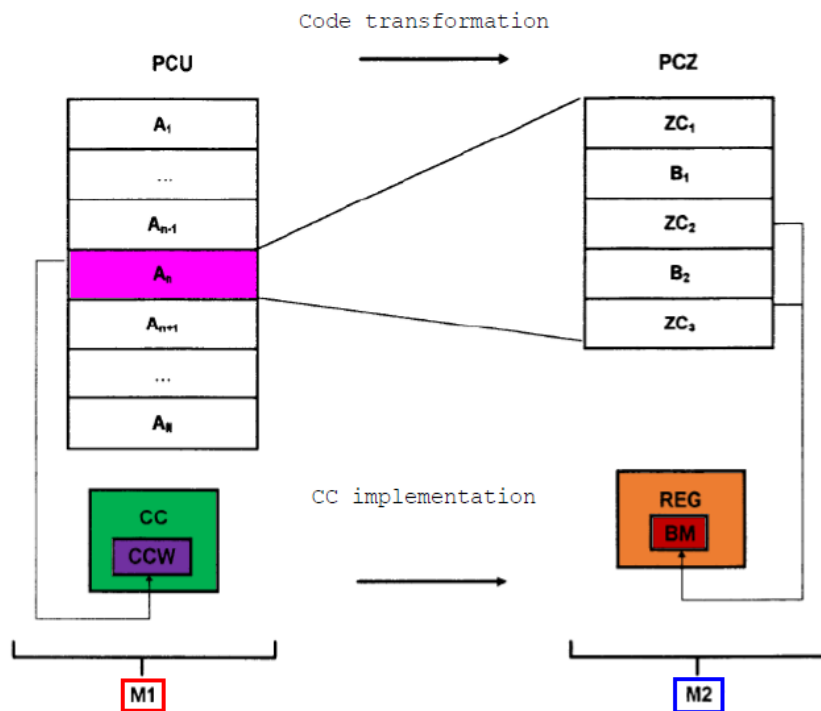


FIG. 1

EX1005, Figure 1. EX2001, ¶87.

⁵ Register REG is a general-purpose register, like the ones described in EX1009.

Loderer discloses that an “IBM/390 computer system is selected as source hardware **MI**.” EX1005, 4. As discussed in Section III.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, which is distinguished from Loderer’s target machine bit pattern. EX2001, ¶88.

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.”). EX2001, ¶89.

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. EX2001, ¶90.

As discussed in Section II.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. That Loderer teaches a bit pattern stored in a general-purpose register—not a “target machine condition code”—is fatal to LzLabs' Petition. EX2001, ¶91.

Specifically, 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. EX2001, ¶92.

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. EX2001, ¶93.

VII. 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

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 VI.A, in connection with independent claims 1, 11, and 16. EX2001, ¶94.

VIII. CONCLUSION

For the reasons set forth above, Petitioner has failed to demonstrate the unpatentability of the challenged claims.

Case IPR2023-00274
U.S. Patent No. 8,713,289 B2

Dated: September 22, 2023

Respectfully submitted,

/Theodoros Konstantakopoulos/
Theodoros Konstantakopoulos
(Reg. No. 74,155)
tkonstantakopoulos@desmaraisllp.com
DESMARAIS LLP
230 Park Ave
New York, NY 10169
212-808-1060 (telephone)
212-351-3401 (facsimile)

Counsel for Patent Owner
International Business Machines Corp.

CERTIFICATION UNDER 37 C.F.R. § 42.24(d)

I hereby certify that this Patent Owner's Response, excluding the portions exempted under 37 C.F.R. § 42.24(a), has:

- 5,630 words as counted by the word-processing system used to prepare this document; and
- 371 words in the illustrations, counted manually.

This document's total word count, excluding the portions exempted under 37 C.F.R. § 42.24(a), is 6,001 words, in compliance with 37 C.F.R. §42.24(b)(2) and §42.24(a)(1)(i).

Dated: September 22, 2023

Respectfully submitted,

/Theodoros Konstantakopoulos/
Theodoros Konstantakopoulos
(Reg. No. 74,155)
tkonstantakopoulos@desmaraisllp.com
DESMARAIS LLP
230 Park Ave
New York, NY 10169
212-808-1060 (telephone)
212-351-3401 (facsimile)

Counsel for Patent Owner
International Business Machines Corp.

CERTIFICATE OF SERVICE

Pursuant to 37 C.F.R. § 42.6(e), the undersigned certifies that a complete copy of the foregoing materials:

- Patent Owner's Response;
- Exhibit List;
- Exhibits 2001-2005;
- Word Count Certification Under 37 CFR § 42.24(d)

was served on counsel of record for Petitioner by filing this document through P-TACTS on September 22, 2023, and by sending this document via electronic mail to the following addresses:

Joseph D. Gray (Reg. No. 61,803)
Slayden Grubert Beard PLLC
401 Congress Ave., Suite 1650
Austin, Texas 78701
Tel. 512-402-3550
jgray@sgbfirm.com

Chris V. Ryan (Reg. No. 54,759)
RyanIPLaw, PC
1508 Bay Hill Drive
Austin, TX 78701-4078
chris.ryan@ryaniplaw.com

Tecuan Flores (Reg. No. 77,668)
Slayden Grubert Beard PLLC
401 Congress Ave., Suite 1650
Austin, Texas 78701
Tel. 512-402-3550
tflores@sgbfirm.com

litigation@sgbfirm.com

Case IPR2023-00274
U.S. Patent No. 8,713,289 B2

Dated: September 22, 2023

Respectfully submitted,

/Theodoros Konstantakopoulos/
Theodoros Konstantakopoulos
(Reg. No. 74,155)
tkonstantakopoulos@desmaraisllp.com
DESMARAIS LLP
230 Park Ave
New York, NY 10169
212-808-1060 (telephone)
212-351-3401 (facsimile)

Counsel for Patent Owner
International Business Machines Corp.