

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

UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

LZLABS GMBH,
Petitioner

v.

INTERNATIONAL BUSINESS MACHINES CORPORATION
Patent Owner

IPR2023-00274

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

INTERNATIONAL BUSINESS MACHINES CORP.'s
DEMONSTRATIVES

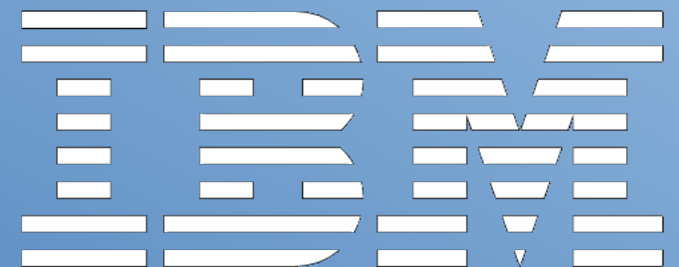
Mail Stop Patent Board
Patent Trial and Appeal Board
U.S. Patent and Trademark Office
P.O. Box 1450
Alexandria, VA 22313-145

UNITED STATES PATENT AND TRADEMARK OFFICE
BEFORE THE PATENT TRIAL AND APPEAL BOARD

**LzLabs GMBH v.
International Business Machines Corp.**

March 21, 2024

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



Overview

- 1 Overview of the '289 Patent
- 2 Petitioner's Grounds
- 3 Disputed Issues

Overview

- 1 Overview of the '289 Patent
- 2 Petitioner's Grounds
- 3 Disputed Issues

Emulation Of Condition Code Settings Without Branch Instructions

(12) **United States Patent**
Copeland et al.

(10) **Patent No.:** **US 8,713,289 B2**
(45) **Date of Patent:** **Apr. 29, 2014**

(54) **EFFICIENTLY EMULATING COMPUTER ARCHITECTURE CONDITION CODE SETTINGS WITHOUT EXECUTING BRANCH INSTRUCTIONS**

(75) Inventors: **Reid T. Copeland**, Toronto (CA);
Patrick R. Doyle, Toronto (CA);
Charles B. Hall, Calgary (CA); **Andrew Johnson**, Hampshire (GB); **Ali I. Sheikh**, Toronto (CA)

(73) Assignee: **International Business Machines Corporation**, Armonk, NY (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 1187 days.

(21) Appl. No.: **11/668,623**

(22) Filed: **Jan. 30, 2007**

(65) **Prior Publication Data**

US 2008/0184014 A1 Jul. 31, 2008

(51) **Int. Cl.**

G06F 9/455 (2006.01)

(52) **U.S. Cl.**

USPC **712/220**; 703/26

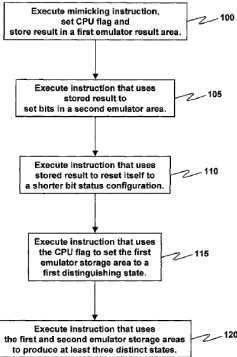
(58) **Field of Classification Search**

USPC 712/209, 221, 227, 220; 717/124, 134,

717/136, 137, 138; 703/26

See application file for complete search history.

20 Claims, 4 Drawing Sheets



LzLabs GmbH. Ex. 1001-1

Ex. 1001 ('289 Patent), 1

(54) **EFFICIENTLY EMULATING COMPUTER ARCHITECTURE CONDITION CODE SETTINGS WITHOUT EXECUTING BRANCH INSTRUCTIONS**

This invention 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**. Even more particularly, the present invention is directed to providing sequences of instructions that produce **valid condition code settings without the use of branching instructions from the target architecture**.

Ex. 1001 ('289 Patent), 1:8-14

'289 Patent: Directly Calculate Target Machine Condition Codes

A classic example of an instruction which produces condition code changes upon execution is 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. The compare instruction

Ex. 1001 ('289 Patent), 1:29-34

This invention 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. Even more particularly, the present invention is directed to **providing sequences of instructions that produce valid condition code settings** without the use of branching instructions from the target architecture.

Ex. 1001 ('289 Patent), 1:8-14

operand, the condition code is set to two. In conventional approaches to the emulation of a **compare instruction**, the result is the construction of a sequence of instructions, which include **three branch instructions**. For the reasons set forth immediately below the presence of branch instructions in the target architecture instruction stream is undesirable.

Ex. 1001 ('289 Patent), 2:7-12

In a method for emulating computer instructions from a source machine to produce sequences of instructions on a target machine, 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. The direct calculation avoids the use of branching instructions whose disadvantages are cited above.

Ex. 1001 ('289 Patent), 3:8-14

POR, 7-8

Claim 1 Of The '289 Patent

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.

Ex. 1001 ('289 Patent), claim 1; POR, 8-9

Overview

- 1 Overview of the '289 Patent
- 2 Petitioner's Grounds**
- 3 Disputed Issues

Petitioner's Grounds

Ground	Summary
1	Claims 1, 4, 6, 10-11, and 16 are obvious in view of Loderer
2	Claims 1-4, 6, 10-12, and 16-17 are obvious to Loderer in view of PoO, Kane, and the knowledge of a POSA

Petition, 10



Europäisches Patentamt
European Patent Office
Office européen des brevets



(11) EP 0 843 256 A2

(12) EUROPEAN PATENT SPECIFICATION

(43) Date of publication: 20.05.1998 Bulletin 1998/21

(21) Application number: 97117565.8

(22) Date of filing: 10.10.1997

(84) Designated Contracting States: AT BE CH DE DK ES FI FR GB GR IE IT LI LU MC NL PT SE Designated Extension States: AL LT LV RO SI (30) Priority: 18.11.1996 DE 19647628	(71) Applicant: Siemens Nixdorf Informationssysteme AG 33106 Paderborn (DE) (72) Inventor: Loderer, Johann 85250 Altomünster (DE)
--	--

(54) **Simulation of a condition code during code transformations**

(57) 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.

Mapping is defined which uniquely associates each possible value (CCW) of the condition code (CC) on the source hardware (M1) with a bit pattern (BM) on the target hardware (M2). To store the bit pattern (BM) associated with the respective current value (CCW), a memory area (REG) is provided on the target hardware (M2). When porting the program code (PCU), any instruction (A_n) potentially influencing the value (CCW) is transformed by being simulated in the

program code (PCZ) by means of a command (B1, B2) generating a result value in the course of the execution of the program and without using branching commands. Also, additional code (ZC₁, ZC₂, ZC₃) for the target hardware (M2) is produced which ascertains at runtime the bit pattern (BM) corresponding to the value (CCW) on the source hardware (M1) following execution of the instruction (A_n) by virtue of a prescribed initial bit pattern being subjected to a bit operation whose effect is dependent on the at least one result value generated in connection with the instruction (A_n). The advantage of this method is that for its implementation on the target hardware (M2) no time-consuming jump commands are required.

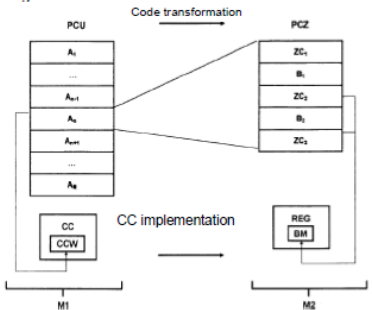


FIG. 1

Printed by Xerox (UK) Business Services
2.16.3/3.4

LzLabs GmbH, Ex. 1005-1

Simulation of a condition code during code transformations

(57) 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.

Mapping is defined which uniquely associates each possible value (CCW) of the condition code (CC) on the source hardware (M1) with a bit pattern (BM) on the target hardware (M2). To store the bit pattern (BM) associated with the respective current value (CCW), a memory area (REG) is provided on the target hardware (M2). When porting the program code (PCU), any instruction (A_n) potentially influencing the value (CCW) is transformed by being simulated in the

program code (PCZ) by means of a command (B1, B2) generating a result value in the course of the execution of the program and without using branching commands. Also, additional code (ZC₁, ZC₂, ZC₃) for the target hardware (M2) is produced which ascertains at runtime the bit pattern (BM) corresponding to the value (CCW) on the source hardware (M1) following execution of the instruction (A_n) by virtue of a prescribed initial bit pattern being subjected to a bit operation whose effect is dependent on the at least one result value generated in connection with the instruction (A_n). The advantage of this method is that for its implementation on the target hardware (M2) no time-consuming jump commands are required.

Overview

- 1 Overview of the '289 Patent
- 2 Petitioner's Grounds
- 3 Disputed Issues**

Disputed Issues

“Target Machine Condition Code”

- The prior art fails to teach or suggest the claimed “target machine condition code”

LzLabs’ New Proposed Claim Construction

- LzLabs’ new claim construction is untimely and improper
- LzLabs’ new claim construction fails on the merits

LzLabs’ Remaining Arguments

- LzLabs’ remaining arguments fail

Disputed Issues

“Target Machine Condition Code”

- The prior art fails to teach or suggest the claimed “target machine condition code”

LzLabs’ New Proposed Claim Construction

- LzLabs’ new claim construction is untimely and improper
- LzLabs’ new claim construction fails on the merits

LzLabs’ Remaining Arguments

- LzLabs’ remaining arguments fail

Claim 1 Of The '289 Patent

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.

Ex. 1001 ('289 Patent), claim 1; POR, 8-9

Loderer Does Not Disclose A “Target Machine Condition Code”

The invention relates to 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.

Ex. 1005 (Loderer), 2

(57) 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.

Ex. 1005 (Loderer), 1 (Abstract)

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, in which the above-mentioned disadvantages are avoided.

Ex. 1005 (Loderer), 2

1. 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),

Ex. 1005 (Loderer), 5 (claim 1)

POR, 27-28

LzLabs' Expert Admitted That Loderer Provides No Condition Codes



Mr. Jestice
Petitioner's Expert

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

A. ***That's correct.***

Ex. 2003 (Jestice Dep. Tr.), 89:4-7

IBM's Expert Also Testified That Loderer Provides No Condition Codes

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.

Ex. 2001 (Goodin Decl.), ¶¶78-79

POR, 27

Loderer's MIPS Processor Does Not Provide Condition Codes

Like the '289 patent, Loderer describes an emulation system that generates emulated condition codes. Ex. 1005, 1. As explained in §VI.A above, Loderer discloses emulation of condition codes for IBM/390 instructions **on a computer having a MIPS processor** by generating sequences of instructions on the target machine that calculate the codes without branch instructions.

Petition, 18

POR, 30

Loderer's MIPS Processor Does Not Provide Condition Codes

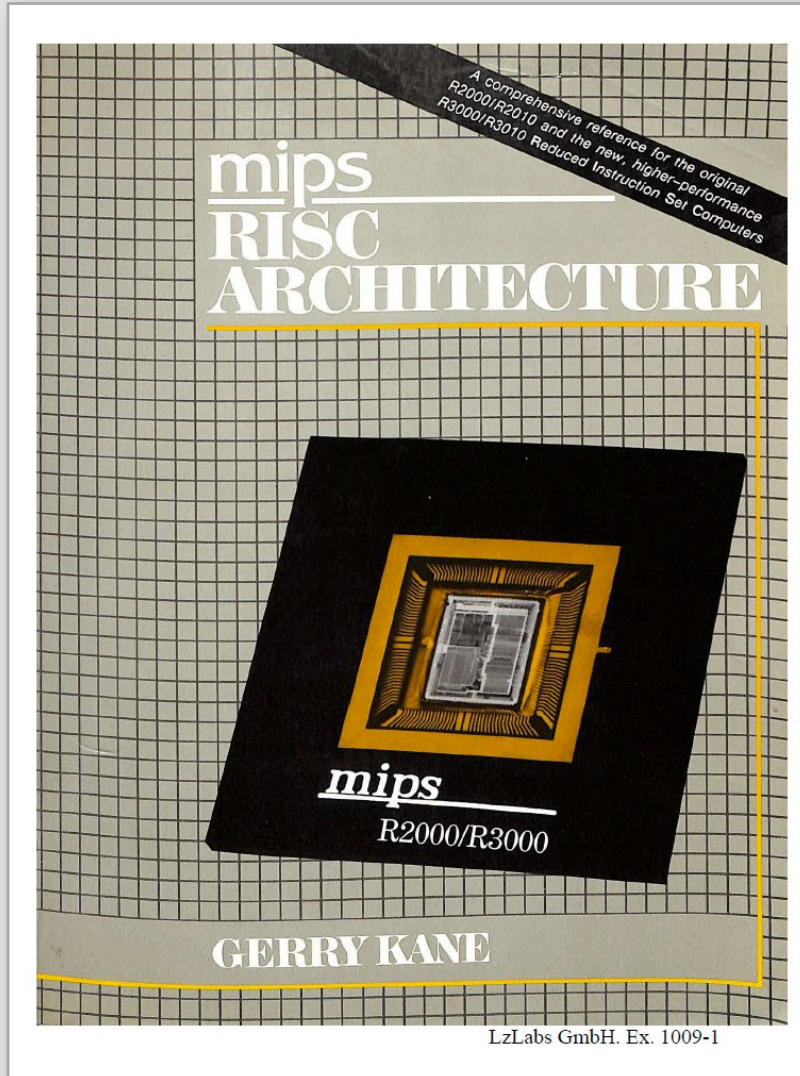
Name	Examples	How condition is tested	Advantages	Disadvantages
Condition code (CC)	80x86, ARM, PowerPC, SPARC, SuperH	Tests special bits set by ALU operations, possibly under program control.	Sometimes condition is set for free.	CC is extra state. Condition codes constrain the ordering of instructions 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

POR, 18, 29

Kane - MIPS RISC Architecture



The RISC architecture implemented by the R2000 Processor provides an efficient, uniform, and streamlined instruction set to obtain maximum efficiency for the most commonly performed operations. As a result, some operations that would require single, multi-cycle instructions in more traditional architectures require multiple, single-cycle R2000 instructions.

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.

Ex. 1009 (Kane-MIPS), 43; POR, 25

IBM Enterprise Systems Architecture/390 Principles of Operation

Enterprise Systems Architecture/390



Principles of Operation

SA22-7201-08
LzLabs GmbH. Ex. 1008-1

PSW

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. The active

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

Program-Status-Word Format

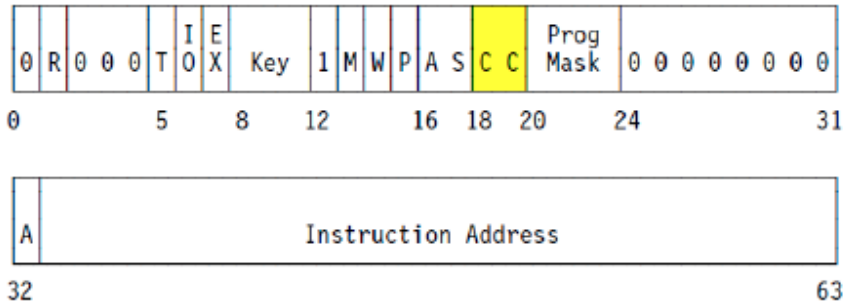


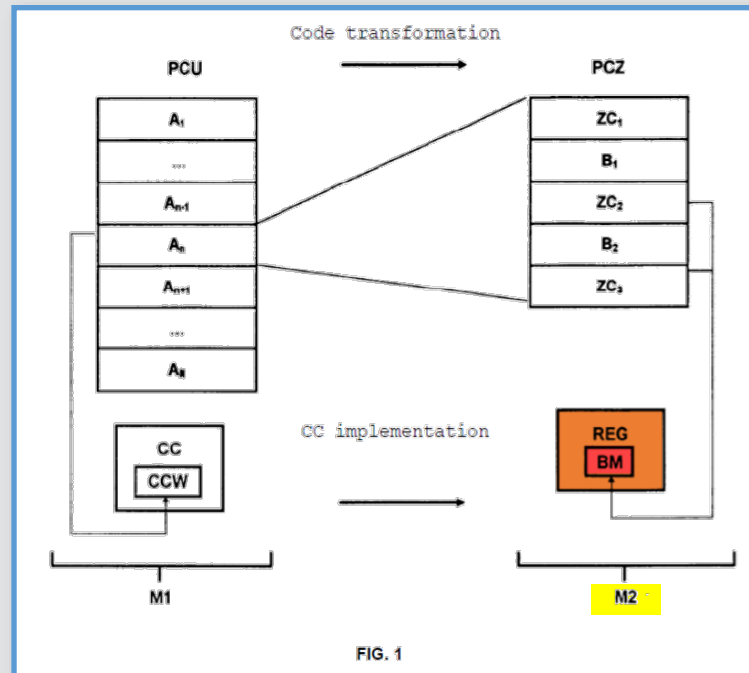
Figure 4-2. PSW Format

Ex. 1008 (IBM-PoP), 51, 113, 115; POR, 23-24

LzLabs Maps “Target Machine Condition Code” To A “Bit Pattern”



Mr. Jestice
Petitioner's Expert



Ex. 1005 (Loderer), Figure 1

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

A. ***Right.***

Ex. 2003 (Jestice Dep. Tr.), 99:19-22

POR, 31

LzLabs' Expert Admitted That A Bit Pattern Is Not A "Condition Code"



Mr. Jestice
Petitioner's Expert

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.**

Ex. 2003 (Jestice Dep. Tr.), 86:5-87:2

POR, 32-33

Condition Code Architectures Are Fundamentally Different From Architectures With No Condition Codes

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

POR, 5-6, 29

LzLabs Relies On Attorney Argument Ignoring Testimony From Both Parties' Experts

- In its Reply, LzLabs ignores the record
 - LzLabs does not dispute that both parties' experts agree Loderer discloses a target machine that does **not** provide condition codes
 - LzLabs does not dispute that the Petition maps the claimed "target machine condition code" to Loderer's "bit pattern"
 - LzLabs does not dispute that both parties' experts agree Loderer's bit pattern is **not** a condition code

Surreply, 1-2, 6

Disputed Issues

“Target Machine Condition Code”

- The prior art fails to teach or suggest the claimed “target machine condition code”

LzLabs’ New Proposed Claim Construction

- LzLabs’ new claim construction is untimely and improper
- LzLabs’ new claim construction fails on the merits

LzLabs’ Remaining Arguments

- LzLabs’ remaining arguments fail

LzLabs' Reply Introduces A New Claim Construction Argument

discloses a “target machine condition code” of the challenged claims. As the petition explains, Loderer’s target machine condition codes are just like those of the ’289 patent—*i.e.*, encoded values, stored as bits in a register, corresponding to condition codes that would have been generated from executing instructions on the source machine. Patent Owner’s (“PO”) position in this IPR is directly contrary to PO’s

Reply, 1

extrinsic textbook to distinguish Loderer. To the extent the Board determines a construction is necessary to resolve the parties’ dispute, the Board should interpret “target machine condition code” consistent with the intrinsic record and PO’s litigation position to include bits of data stored on the target machine that indicate the outcome status of executing the source machine instruction on the source machine (*i.e.*, “an IBM mainframe condition code representation on the target machine”). The Board should reject PO’s attempt to limit the target machine

Reply, 26

Sur-Reply, 2-3

Both Parties And The Board Agreed No Construction Was Required

- LzLabs' position in the Petition:

Petitioner submits terms should be given “ordinary and customary meaning” to those of skill in the art. 37 C.F.R. §42.100(b). Petitioner reserves the opportunity to respond to constructions offered by PO or adopted by the Board. *Nidec Motor*

Petition, 17

- The Board's position in the Institution Decision:

customary meaning' to those of skill in the art.” *Id.* For purposes of this Decision, we conclude that no claim term requires express interpretation to resolve any controversy in this proceeding. *See Vivid Techs., Inc. v. Am. Sci.*

Institution Decision, 7

- IBM's position in the POR:

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.

POR, 26

Sur-Reply, 2-3

LzLabs' Claim Construction Argument Is Untimely And Therefore Improper

- “In an IPR, the petitioner has the burden **from the onset** to show with particularity why the patent it challenges is unpatentable.”
- *Harmonic Inc. v. Avid Tech., Inc.*, 815 F.3d 1356, 1363 (Fed. Cir. 2016) (citing 35 U.S.C. § 312(a)(3))

issues arising from the Supreme Court’s decision in *SAS* (138 S. Ct. at 1358), the Board will permit the petitioner, in its reply brief, to address issues discussed in the institution decision. The patent owner will similarly be allowed to address the institution decision in its sur-reply, if necessary to respond to petitioner’s reply. Petitioner may not submit new evidence or argument in reply that it could have presented earlier, e.g. to make out a prima facie case of unpatentability. A party also may submit rebuttal evidence in support of its reply. *See*

Trial Practice Guide, 73

Sur-Reply, 5-6

Disputed Issues

“Target Machine Condition Code”

- The prior art fails to teach or suggest the claimed “target machine condition code”

LzLabs’ New Proposed Claim Construction

- LzLabs’ new claim construction is untimely and improper
- LzLabs’ new claim construction fails on the merits

LzLabs’ Remaining Arguments

- LzLabs’ remaining arguments fail

LzLabs' Claim Construction Is Circular

discloses a “target machine condition code” of the challenged claims. As the petition explains, Loderer’s target machine condition codes are just like those of the ’289 patent—*i.e.*, encoded values, stored as bits in a register, corresponding to condition codes that would have been generated from executing instructions on the source machine. Patent Owner’s (“PO”) position in this IPR is directly contrary to PO’s

Reply, 1

Sur-Reply, 2-3

Both Experts Agree That “Target Machine Condition Code” Requires No Construction

74. In my opinion, and for purposes of this declaration, the claim terms of the '289 patent do not require construction to address the issues presented. In my analyses of the '289 patent, I have applied the ordinary meaning of the claim language as it would have been understood to a POSA in 2007 in view of the patent specification and its prosecution history.

Ex. 1003 (Jestice Decl.), ¶74

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.

Ex. 2001 (Goodin Decl.), ¶76.

Sur-Reply, 5-7

No Intrinsic Support That rC Is A General-Purpose Register

of a condition code setting.” *Id.*, 4:22-25. The disclosed technique of the ’289 patent results in generating and storing an encoding of the condition code setting in register rC, a general-purpose register on the target machine:

rA, rB—the first and second operand, respectively, of the z/Architecture instruction evaluated into a register;
rT—the result value of the z/Architecture instruction computed into a register;
rX, rY, rZ—temporary registers used to hold intermediate results;
rC—the register that will hold the condition code value at the end of the sequence.

Id., 5:20-29. Indeed, register rC of the Table II embodiment stores the result of the

Reply, 4

Sur-Reply, 8-9

LzLabs' "General-Purpose Register" Argument Is Not Supported By Evidence

tions. First, an instruction (step 100 in FIG. 2) in the target machine's architecture is executed which mimics the instruction present in the source instruction stream in a manner which sets one or more target CPU flag bits and which places a result in a storage location (first location, such as rC above)

Ex. 1001 ('289 Patent), 7:51-55



Mr. Jestice
Petitioner's Expert

Q. Is it your understanding that this section [of MIPS-Kane (Ex. 1009)] describes the benefits of using ***general-purpose registers instead of condition code registers***?

A. ***That's what this article says, yes -- or this book says.***

Ex. 2003 (Jestice Dep. Tr.), 51:13-52:1

Sur-Reply, 8-9

LzLabs Misapprehends IBM's Argument

via the claimed obtaining, generating, and determining steps. However, the challenged claims do not speak to **where** the target machine condition code must be stored on the target machine or otherwise limit the architecture of the target machine, as PO contends. The claims do not require the presence of any specific condition

Reply, 7

- IBM's argument is not contingent on **where** the target machine condition code is stored
- IBM's argument is contingent on whether the target machine provides, e.g., generates, condition codes in the first place
 - Loderer does not satisfy the claimed "target machine condition code"
 - A "condition code" is not a bit pattern in a general-purpose register

Sur-Reply, 9-10, 12

LzLabs' Expert Admitted That A "Bit Pattern" Is Not A "Condition Code"

discloses a "target machine condition code" of the challenged claims. As the petition explains, Loderer's target machine condition codes are just like those of the '289 patent—*i.e.*, encoded values, stored as bits in a register, corresponding to condition codes that would have been generated from executing instructions on the source machine. Patent Owner's ("PO") position in this IPR is directly contrary to PO's

Reply, 1



Mr. Jestice
Petitioner's Expert

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.**

Ex. 2003 (Jestice Dep. Tr.), 86:22-87:2

Surreply, 7-8

Disputed Issues

“Target Machine Condition Code”

- The prior art fails to teach or suggest the claimed “target machine condition code”

LzLabs’ New Proposed Claim Construction

- LzLabs’ new claim construction is untimely and improper
- LzLabs’ new claim construction fails on the merits

LzLabs’ Remaining Arguments

- LzLabs’ remaining arguments fail

LzLabs Mischaracterizes The Specification

- LzLabs mischaracterizes the Specification by taking sentences out of context to argue that “any target architecture” applies to the ‘289 claims

The teachings disclosed in the ‘289 patent are expressly applicable to *any* target architecture. The specification states “the principles set forth herein are applicable to any source architecture and to *any target architecture*.” *Id.*, 1:47-49; *see also id.*, 3:29-32 (“[T]he principals, techniques and methods of *the present invention are not limited to any particular target machine architecture*. The two exemplar architectures discussed herein are merely the most currently ones anticipated to be of the greatest value.”).³

Reply, 5

Sur-Reply, 10

LzLabs Mischaracterizes The Specification

- LzLabs mischaracterizes the Specification by taking sentences out of context to argue that “any target architecture” applies to the ‘289 claims

The teachings disclosed in the ‘289 patent are expressly applicable to *any* target architecture. The specification states “the principles set forth herein are applicable to any source architecture and to *any target architecture*.” *Id.*, 1:47-49; see also *id.*, 3:29-32 (“[T]he principals, techniques and methods of *the present invention are not limited to any particular target machine architecture*. The two exemplar architectures discussed herein are merely the most currently ones anticipated to be of the greatest value.”).³

Reply, 5

instruction emulation. While the systems and methods of the present invention are widely applicable to any emulation method *where condition codes are present*, it is particularly applicable to the emulation of the *z/Architecture*. However, the principles set forth herein are applicable to any source architecture and to any target architecture.

Ex. 1001 (‘289 Patent), 1:44-49

Sur-Reply, 10-11

LzLabs Mischaracterizes The Specification

- LzLabs mischaracterizes the Specification by taking sentences out of context to argue that “any target architecture” applies to the ‘289 claims

The teachings disclosed in the ‘289 patent are expressly applicable to *any* target architecture. The specification states “the principles set forth herein are applicable to any source architecture and to *any target architecture*.” *Id.*, 1:47-49; *see also id.*, 3:29-32 (“[T]he principals, techniques and methods of *the present invention are not limited to any particular target machine architecture*. The two exemplar architectures discussed herein are merely the most currently ones anticipated to be of the greatest value.”).³

Reply, 5

The 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. The sequences of the present inventions are more efficient and generally smaller as opposed to a more straightforward method that requires more flow control. However, it is noted that **the principals, techniques and methods of the present invention are not limited to any particular target machine architecture. The two exemplar architectures discussed herein are merely the most currently ones anticipated to be of the greatest value.**

Ex. 1001 (‘289 Patent), 3:20-34

Sur-Reply, 10-12

LzLabs Ignores The Specification's Full Disclosure

EX1001, 5:39-56, 6:23-25, 6:59-62, 7:3-10, 7:25-49. Moreover, the '289 patent explains that the calculated target machine condition code can be stored in locations other than hardware or condition code elements of the target architecture:

It is noted that while an emulation environment preferably results in the setting of hardware or condition code elements in the target architecture, *the present invention also contemplates the situation in which condition codes are generated and stored in locations other than condition code registers in the target machine.*

EX1001, 1:57-62.

Reply, 10

- Petitioner's quote expressly states "***condition codes are generated***"

Sur-Reply, 13-14

Extrinsic Evidence Is Consistent With Intrinsic Record

meaning of the claim terms”). Yet PO largely ignores the disclosure of the ’289 patent and attempts to use extrinsic evidence to limit “target machine condition code” in a manner that is inconsistent with the intrinsic record and inconsistent with PO’s infringement allegations in parallel litigation between the parties.

Reply, 17

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

POR, iii

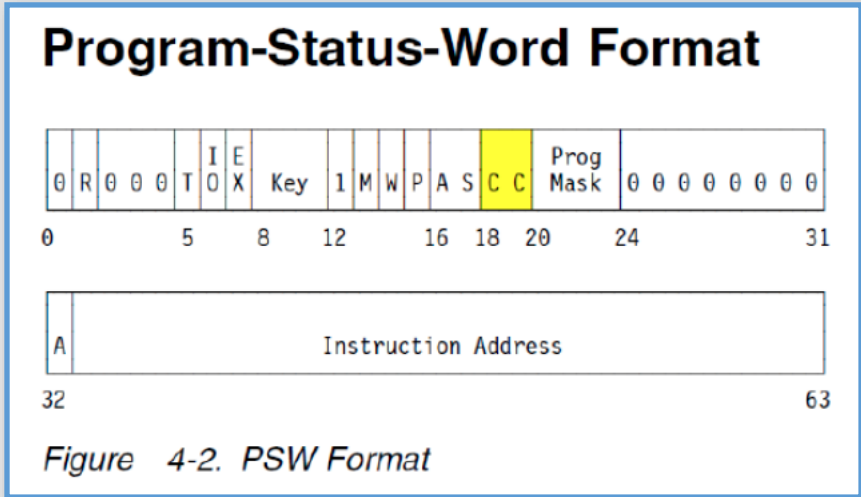
- LzLabs does not dispute the evidence in Exhibits 2004 and 2005
- The POR also relies on LzLabs’ extrinsic evidence (EX1005, EX1008, EX1009, and EX1024)

Sur-Reply, 16-17

LzLabs’ “Encoding” Argument Is Another Red Herring

EX1001, 1:18-23. The challenged claims recite deriving “an encoding for a target machine condition code,” and—like Loderer—the ’289 specification discloses bit patterns stored in a register representing emulated IBM architecture condition codes.

Reply, 23



Ex. 1008 (IBM-PoP), 115

Sur-Reply, 21-22

LzLabs Fails To Show That IBM's IPR Position Is Inconsistent With Its Litigation Positions

EX1028, 4. While PO's preliminary infringement contentions do not foreclose setting hardware or condition code elements at the target machine, PO expressly states it is the "representation of the source machine ISA condition code that is referred to here as the 'target machine condition code.'" *Id.*, 5. PO thus alleged

...

PO's interpretation and mapping of "target machine condition code" for infringement are thus consistent with Petitioner's mapping of "target machine condition code" in Loderer. The POR makes no attempt to square PO's new IPR

Reply, 25

Sur-Reply, 23-24

LzLabs Fails To Show That IBM’s IPR Position Is Inconsistent With Its Litigation Positions

143. To perform this emulation, the SDM first obtains, by the “target machine” with an x86 architecture, a computer instruction that is intended for a “source machine” having a z/Architecture. *Id.* Once it receives this instruction, the SDM generates a sequence of target instructions which together operate to derive an IBM mainframe condition code representation on the target machine, the “target machine condition code,” for a particular IBM mainframe source instruction, as such a condition code may be required to execute a subsequent IBM mainframe source instruction. Performance considerations for a processor, such as an x86 machine using pipelining, dictates that this be done without using branch instructions. Additionally, target instructions would provide information distinguishing between different IBM mainframe condition codes to ensure that the correct condition code is generated. Because IBM mainframe instructions that set the condition code are common in COBOL and PL/I compile programs, if the “branchless” calculations of the ’289 patent were not used, efficiency and performance of related services would noticeably decrease. Thus, because LzLabs claims that there is no such performance degradation, its SDM must make use of these “branchless” calculations when determining the claimed condition code.

Ex. 1027 (Complaint), ¶143

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.

Ex. 2004, 8

Sur-Reply, 22-24

QUESTIONS?

BACKUP SLIDES

IBM's Last Amendment During Prosecution

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.

Ex. 1004 (File History), 74-75

POR, 14-15

Indications Of Condition Codes In A Register Are Not Condition Codes

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 in an AFLAGS register (see FIG. 3). Cited in the Office Action for an

Ex. 1004 (File History), 123

POR, 13-14

LzLabs' "Emulated Condition Code" Argument Is A Red Herring

"Target machine condition code" of the '289 patent thus includes emulated condition code settings on the target machine that would have resulted from execution of the source machine instruction on the source machine.⁴

Reply, 9

- That the "target machine condition code" is an emulated condition code has no bearing on whether the target machine condition code is generated or not

In the principle emulation environment considered in the present description, it is the job of emulation software to accept, as input, strings of source architecture instructions and to generate therefrom strings of instructions that, when run on the target architecture, produce the same results. These results include the setting of various condition codes, such as sign, carry, overflow and various others indicating exceptions

Ex. 1001 ('289 Patent), 1:51-57

Sur-Reply, 12-13

CERTIFICATE OF SERVICE

Pursuant to 37 C.F.R. § 42.6(e), the undersigned certifies that a complete copy of the foregoing was served on counsel of record for the Petitioner by filing this document through P-TACTS 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

Dated: March 19, 2024

Respectfully submitted,

/Theodoros Konstantakopoulos/
Theodoros Konstantakopoulos
(Reg. No. 74,155)
tkonstantakopoulos@desmaraisllp.com
DESMARAIS LLP

Lead Counsel for Patent Owner
International Business Machines Corporation