

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
Patent 8,713,289

PETITIONER'S REPLY

TABLE OF CONTENTS

I.	Unrebutted Contentions	2
II.	Condition Codes in the '289 Patent	2
III.	“Target Machine Condition code” in the Intrinsic Record	5
A.	“Target Machine Condition Code” in the Claims.....	5
B.	“Target Machine Condition Code” in the Specification	7
1.	Emulated Condition Code Settings are “Target Machine Condition Codes”	7
2.	“Target Machine Condition Codes” May Be Stored in a Variety of Locations	10
C.	“Target Machine Condition Code” in the Prosecution History	11
IV.	Undisputed teachings of Lorderer	13
V.	PO’s Attempt to Limit the Target Machine Architecture Is Based on Extrinsic Evidence and Inconsistent With PO’s Litigation Positions, the Claim Language, and the Intrinsic Record.....	17
A.	Summary of PO’s Argument Based on Extrinsic Evidence	17
B.	PO’s Argument is Inconsistent with Its Litigation Positions.....	24
C.	PO’s Argument is Inconsistent with the Claim Language and the Intrinsic Record.....	25
VI.	Conclusion.....	28

LIST OF EXHIBITS

1001	U.S. Patent No. 8,713,289 to Copeland et al (“the ’289 patent”)
1002	Ian Jestice CV
1003	Expert Declaration of Ian Jestice
1004	File History of U.S. Patent No. 8,713,289 to Copeland
1005	Translation of EP0843256A2 to Loderer and certification of translation
1006	EP0843256A2 to Loderer
1007	U.S. Patent 6,049,864 to Liu
1008	Enterprise Systems Architecture/390 Principles of Operation (“PoO”), SA22-7201-08, Ninth edition (June 2003)
1009	Kane, G., MIPS RISC Architecture, © 1988 (“Kane”)
1010	U.S. Patent Pub. No. 2006/0036790A1 to Peterson et al.
1011	U.S. Patent No. 5,889,980
1012	Umeno, H., “New Method for Dispatching Waiting Logical Processors in Virtual 7Machine System,” IEEE 2005
1013	Scheduling Order (Dkt. 26) <i>International Business Machines Corp. v. LzLabs GmbH and Texas Wormhole, LLC</i> , Case No. 6:22-cv-00299-ADA (W.D. Tex.)
1014	Cragon, H., “Computer Architecture and Implementation,” ©2000, Cambridge University Press, ISBN 0-521-65168-9
1015	Enterprise Systems Architecture/390 Principles of Operation, SA22-7201-06, Seventh Edition (July 1999)
1016	IBM System/370 Principles of Operation, SA22-7085-07, First Edition (March 1983)

1017	Declaration of Jim Mullins, Ph.D
1018	Information Disclosure Statement, U.S. Patent Application No. 09/608750 (June 29, 2005)
1019	Information Disclosure Statement, U.S. Patent Application No. 09/0924755 (July 1, 2005)
1020	U.S. Patent No. 5,661,674
1021	U.S. Patent No. 5,576,555
1022	Plambeck, K. et al., “Development of attributes of z/Architecture,” IBM J. RES. & DEV. VOL. 46 NO. 4/5, (July/September 2002)
1023	U.S. Patent No. 5,481,688
1024	Hennessey, J. et al., “Hardware/Software Tradeoffs for Increased Performance,” ACM SIGARCH Computer Architecture News, Vol. 10, Issue 2, March 1982 (c) 1982
1025	IBM System/360 Principles of Operation, Form A22-6821-0
1026	IBM z/Architecture Principles of Operation, SA22-78320-02, Third Edition (June 2003)
1027	IBM’s Original Complaint in Civil Action No. 6:22-cv-00299 (W.D. Tex.), filed March 21, 2022
1028	Excerpt of IBM’s Preliminary Infringement Contentions (’289 Patent) in Civil Action No. 6:22-cv-00299 (W.D. Tex.), served June 23, 2022

Patent Owner's Response ("POR") raises a single dispute: whether Loderer 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 position in parallel litigation between the parties, in which IBM alleges a "target machine condition code" is "an IBM mainframe condition code representation on the target machine." PO's new IPR position is based on the flawed premise that the alleged invention of the '289 patent is limited to target architectures with certain kinds of dedicated support for condition codes, such as a "PSW" (for IBM/390) or a "condition code register" (for PowerPC). The POR and PO's expert¹ simply ignore PO's litigation contentions and express disclosures of the '289 patent that directly undercut PO's position, relying instead on extrinsic evidence for this unsupported target architecture limitation. The Board should find all challenged claims unpatentable for the reasons set forth in the petition.

¹ The substance of PO's expert declaration is nearly identical to the POR. *Compare* EX2001, *with* Paper 12.

I. UNREBUTTED CONTENTIONS

The sole dispute between the parties is whether Loderer discloses a “target machine condition code” recited in the challenged claims. PO does not refute that Loderer discloses or suggests every other element of claims 1, 4, 6, 10-11, and 16 as enumerated in Ground 1 of the petition. Paper 1, 18-48. Similarly, PO does not dispute that a POSA would have been motivated to combine Loderer with PoO and Kane and would have had a reasonable expectation of success doing so, or that such a combination discloses or suggests every other element of claims 1-4, 6, 10-12, and 16-17 as explained in Ground 2 of the petition. Paper 1, 48-76. PO has therefore waived any such arguments. *See* Paper 8, 9 (“Patent Owner is cautioned that any arguments not raised in the response may be deemed waived.”).

II. CONDITION CODES IN THE '289 PATENT

As explained in the petition, the '289 patent relates to a system that emulates the generation of condition codes for instructions from a computer having a first architecture on a computer having a different architecture, without using branch instructions. Paper 1, 3-4.

Modern data processing systems generate “condition codes,” which the specification describes broadly as bits of data that indicate the outcome status of instruction execution:

In virtually all modern data processing systems, the execution of

various operations such as arithmetic operations, logical operations and even data transfer operations, may result in the generation of several *bits of data to indicate the outcome status of instruction execution.*

These bits are typically referred to as condition codes.

EX1001, 1:18-23²; *see also* Paper 7 (Institution Decision), 2 (“The ’289 patent explains that executing various operations may result in the generation of condition codes, which are bits of data indicating the outcome status of instruction execution.” (citing EX1001, 1:18-23)).

The ’289 patent explains it is sometimes desirable to emulate instructions designed for one computer architecture on another system with a different architecture, a process generally referred to as emulation. EX1001, 1:36-39. It is the job of emulation software to accept, as input, strings of source architecture instructions and to generate strings of instructions that, when run on the target architecture, produce the same results. *Id.*, 1:50-54. These results include producing condition code settings to reflect the outcome status of the instructions that are then stored in the target architecture. *Id.*, 1:8-11, 1:54-62.

The alleged invention of the ’289 patent relates to the handling of condition code settings in the context of emulation. *Id.*, 1:42-44. For example, the specification explains that “conventional approaches to the emulation of a compare instruction”

² Emphasis and annotations herein added.

relies on a series of if/then statements, also referred to as branch instructions, to determine and set the condition code on the target machine. *Id.*, 2:7-10. Such branch instructions are undesirable for reasons enumerated in the specification. *Id.*, 2:14-20, 2:26-32.

To avoid branch instructions, the '289 patent generates sequences of code on the target computer to for “emulating condition code settings” on a target machine by directly calculating and encoding “target machine condition codes.” *Id.*, 5:31-32; *see also id.*, 2:35-37, 5:17-7:18, 7:25-49. “The technique used herein to derive the sequences is to implement very short one or two instruction sequence fragments that set a bit or bits in a result or temporary register to distinguish each possible outcome 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 com-
puted 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 subtraction of instruction [1], rC is an operand in instructions [4] and [5], and rC

holds the condition code value at the end of the sequence. *Id.*, 6:23-25, 6:59-62, 7:3-8.

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

III. “TARGET MACHINE CONDITION CODE” IN THE INTRINSIC RECORD

A. “Target Machine Condition Code” in the Claims

The challenged claims recite a method for generating a sequence of target machine instructions to derive an encoding for a “target machine condition code”:

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

³ The POR and PO’s expert simply ignore these statements from the specification, presumably because they directly undermine PO’s attempt limit the target machine architecture.

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.

The claim explicitly states that the “target machine condition code” is calculated for a computer instruction obtained from the source machine and that the

code is calculated through a sequence of steps. The “generating” step generates target machine instructions that derive an encoding for a “target machine condition code for the [source machine] computer instruction” by directly calculating the target machine condition code. “Directly calculating” is defined by the claim to include first determining an intermediate value of the target machine condition code, and possibly modifying that value based on distinguishing information. The claim also requires this direct calculation is performed without using branch instructions.

Thus, the challenged claims state that the “target machine condition code” is the result of these steps that recite *how* to calculate a target machine condition code 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 code hardware or elements on the target machine and do not limit the storage location of a “target machine condition code.”

B. “Target Machine Condition Code” in the Specification

1. Emulated Condition Code Settings are “Target Machine Condition Codes”

As detailed in the petition, the specification provides an example of emulating the IBM z/Architecture “Compare Logical” instruction in Table II, which compares

the contents of two register operands, rA and rB, and illustrates the steps of the challenged claims:

TABLE II

[1]	sub_set_carry	rC, rA, rB
[2]	set_bit_on_zero	rX, rC
[3]	set_bit_on_not_zero	rC, rC
[4]	add_to_carry_immed	rC, rC, 0
[5]	sub	rC, rC, rX

Instruction	rA > rB			rA = rB			rA < rB		
	Register Contents			Register Contents			Register Contents		
	rC	rX	Carry	rC	rX	Carry	rC	rX	Carry
[1]	rC > 0	N/A	1	rC = 0	N/A	1	rC < 0	N/A	0
[2]	rC > 0	0	1	rC = 0	1	1	rC < 0	0	0
[3]	1	0	1	0	1	1	1	0	0
[4]	2	0	N/A	1	1	N/A	1	0	N/A
[5]	2	0	N/A	0	1	N/A	1	0	N/A

EX1001, 5:41-56; *id.*, 5:20-22, 5:29-37; Paper 1, 5-7. Instructions [1]-[5] perform arithmetic and logical operations, storing bits representing values in general-purpose registers rC and rX. *See, e.g.* EX1001, 5:20-27-28. At the end of the sequence of instructions, register rC holds the result (*i.e.*, target machine condition code) for the Compare Logical instruction—*e.g.*, rC=2 if rA>rB; rC=1 if rA<rB; and rC=0 if rA=rB. *Id.*, 7:9-10.

This “sequence for emulating condition code settings” “emulate[s] the generation of condition code settings in target architectures” while avoiding branch instructions, *i.e.*, it generates bits of data representing a target machine condition code, without using branch instructions, then stores those bits in register rC. *See* EX1001, 2:35-37, 5:31-32.

The sequence of instructions of the embodiments of columns 5-7 of the '289 specification set condition code values in register rC of the target machine that would have been generated from executing the disclosed instructions on the z/Architecture source machine. *See* EX1001, 5:34-36 (“In particular, the subject source instruction is the z/Architecture Instruction called the Compare Logical operation.”), 5:40-55 (disclosing target machine condition codes of 0, 1, and 2 for “Compare Logical” instruction); EX1026, 78-79, 219-220 (z/Architecture condition codes for “Compare Logical”: 0=operands equal; 1= first operand low; 2=first operand high); *see also* EX1001, 7:19-21 (“As another example . . . the Add Logical (32 bit) and Add Logical (64 bit) instructions are considered below.”), 7:30-48 (disclosing target machine condition codes of 0, 1, 2, and 3 for “Add Logical” instruction); EX1026, 41 (z/Architecture condition codes for “Add Logical”: 0=result zero; no carry; 1=result not zero, no carry; 2=result zero, carry; 3=result not zero, carry).

“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

⁴ Though the '289 patent identifies PowerPC and Intel as exemplary target machine architectures (EX1001, 3:24-26), neither the patent nor the POR identifies condition

2. “Target Machine Condition Codes” May Be Stored in a Variety of Locations

Though “[i]n some cases it is possible and efficient to manipulate the PowerPC [target machine] condition code register itself to derive the z/Architecture [source machine] condition code settings,” the ’289 patent is expressly not limited to such embodiments. *See* EX1001, 1:57-62, 3:29-34. Indeed, in the embodiment described above, the target machine condition code is stored in general-purpose register rC, which is the same register used to store the result of an arithmetic operation in instruction [1] and as an operand in instructions [4] and [5]. *See* 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.

codes used by PowerPC or Intel processors for any instructions (including “Compare Logical” or “Add Logical”).

C. “Target Machine Condition Code” in the Prosecution History

Consistent with Petitioner’s above explanation, PO identified during prosecution the value calculated and stored as bits in general-purpose register rC discussed in Table II as the “target machine condition code”:

Thus, according to an aspect of applicants’ recited invention, a target machine condition code is calculated without using branch instructions. . . . To illustrate, refer to the tables presented in paragraphs [0025] and [0031] in applicants’ specification. *The target machine condition code, as a result of the sequence of instructions, will be held in register rC.* An intermediate condition code value for that target machine condition code is determined (e.g. by performing the first four instructions in these examples) and placed in rC. That intermediate code value is a provisional value for the target machine condition code . . . Based on performing the final instruction, in these examples, *the target machine condition code is determined and stored in rC*, which may or may not be a modified version of the intermediate condition code value that was determined and stored in rC prior to performance of the final instruction.

EX1004, 83-84; *see also id.*, 212 (“The examples provided in the specification for the present invention describe how to derive an encoding of 0, 1 or 2 (with binary bit patterns of 00, 01, 10, respectively”).

Thus, according to PO during prosecution, a “target machine condition code” is bits of data representing a value that is calculated by a sequence of target machine

instructions that may be stored in a general-purpose register, such as register rC. And as discussed above, the target machine condition codes disclosed in the specification are emulated source machine condition code settings (*i.e.*, condition codes that would have been generated from executing the source machine instruction on the source machine). *See* EX1001, 7:8-10 (target machine condition codes of 0, 1, and 2 for “Compare Logical”), 7:30-48 (target machine condition codes of 0, 1, 2, and 3 for “Add Logical”); EX1026, 78-79, 219-220 (z/Architecture condition codes of 0, 1, and 2 for “Compare Logical”), 41 (z/Architecture condition codes of 0, 1, 2, and 3 for “Add Logical”). PO’s identification of the value stored in register rC as the target machine condition code during prosecution reflects the specification’s broad description of condition codes as “bits of data to indicate the outcome status of instruction execution” that can be “stored in locations other than condition code registers in the target machine.” EX1001, 1:18-23, 1:57-62.⁵

⁵ PO suggests “target machine condition code” was added to the challenged claims late in prosecution to overcome the prior art and somehow this supports PO’s effort to limit the target machine architecture. *See* POR, 11-16, 33. PO, however, overlooks that the term “target machine condition codes” was part of the original claims (EX1004, 405) and that the amendments to independent claim 9 the POR discusses

IV. UNDISPUTED TEACHINGS OF LORDERER

As detailed in the petition, Loderer is directed to the same problem as the '289 patent: generating condition code settings during emulation without branches. EX1005, 1, col. 1 ¶1; Petition, 10-13.

PO acknowledges Loderer discloses a method for simulating a condition code when porting hardware-specific program code from a source hardware to a target hardware. *See* POR, 16-17. Program code PCU is ported from a source hardware M1 to a target hardware M2. *Id.*, 16-17, 20.

(POR, 13-15) did not introduce the term “target machine condition code” into the claims. Rather, these amendments merely clarified that “the condition code” referenced in the latter part of the claim refers to the same “target machine condition code” already recited in the pending claim. EX1004, 74-75. Moreover, the response to the fourth Office Action PO cites as “clarify[ing] the requirement that the claimed ‘condition code’ is a ‘target machine condition code’” (POR, 14) is the same response in which PO identified the value stored in general-purpose register rC from the specification as the “target machine condition code.” EX1004, 83-84.

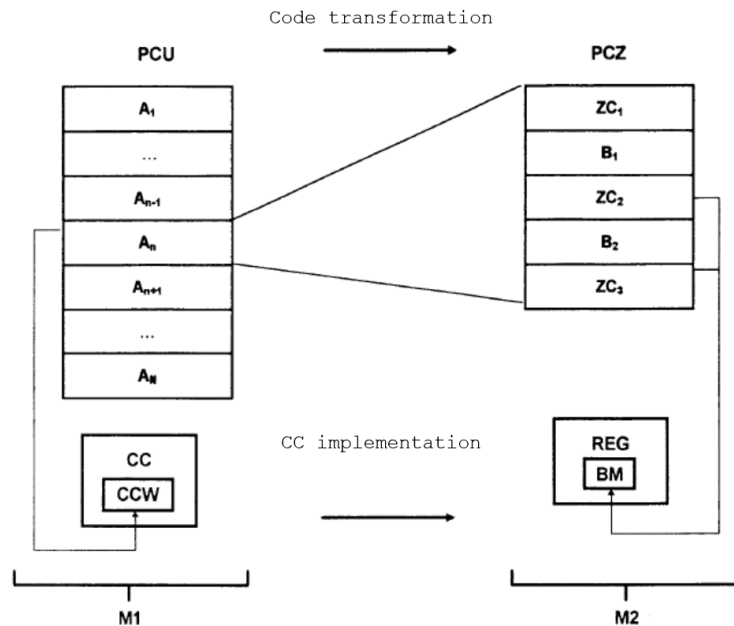


FIG. 1

EX1005, Fig. 1; *id.*, 1, col. 4 ¶7. Instructions $A_1 \dots A_N$ from the source machine with architecture M1 are emulated on the second machine with architecture M2, where each instruction A_n is replaced with a sequence of instructions for architecture M2. *Id.*, col. 4 ¶¶7-8; POR, 20-21. Loderer explains that “[f]or instructions potentially influencing the value CCW of the condition code CC [on the source machine], additional program code (additional code) is generated for the simulation of the condition code CC on the target hardware M2.” EX1005, col. 4 ¶8; POR, 21.

For example, instruction A_n from the source machine is replaced on the target machine with a “command sequence” consisting “of two commands B_1 and B_2 required for the semantic representation of the instruction A_n and an additional code consisting of three subsections ZC_1 , ZC_2 and ZC_3 .” EX1005, col. 4 ¶8.

Loderer uses the setting of condition codes for IBM/390 comparison instructions to illustrate the problem and solution. Paper 1, 18-22. Fig. 2 illustrates condition code settings for the COMPARE instruction:

Comparison result	CCW	BM
OP1 equal to OP2	0	1000
OP1 < OP2	1	0100
OP1 > OP2	2	0010
Overflow	3	0001

FIG. 2

EX1005, Fig. 2. The IBM/390 comparison operation compares two signed operands (OP1 and OP2) and sets a condition code value (column CCW above) to zero, one, or two, depending on whether OP1 is equal to, less than, or greater than OP2. *Id.*, col. 5 ¶¶1-3, Fig. 2. Loderer encodes these condition code values in a bit pattern (BM) stored in a register on the target machine, with BM=1000 for CCW=0, BM=0100 for CCW=1, and BM=0010 for CCW=2. *Id.*

Like the '289 patent, Loderer's sequence of instructions generates target machine condition codes corresponding to condition codes as if the instruction had been executed on the source machine—*i.e.*, BM on the target machine representing the IBM/390 condition code CCW in Figure 2 of Loderer. Paper 1, 18-22; EX1005, 1 (“[A]dditional 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 (An)”); *id.*, 4 (“The bit pattern BM resulting in this step is uniquely associated with the value CCW of the condition code CC occurring after execution of the original comparison instruction on the source hardware M1 and can be evaluated by subsequent instructions in the program code PCZ”); *see also* EX1008, 339 (IBM/390 condition codes for “Compare”: 0=operands equal; 1= first operand low; 2=first operand high, which correspond to CCW in Loderer).

Loderer’s Fig. 4 illustrates a target machine (PCZ) command sequence that calculates encoding for the condition code without branch instructions on the target machine M2. EX1005, Fig. 4, col. 5 ¶4-col. 6 ¶2.

	Command sequence in PCZ	a = 0	a > 0	a < 0
ZC ₁	Set BM to '0100'	BM = 0100	BM = 0100	BM = 0100
B1	If a = 0, set t ₁ = 1, otherwise t ₁ = 0	t ₁ = 1	t ₁ = 0	t ₁ = 0
ZC ₂	BM << t ₁	BM = 1000	BM = 0100	BM = 0100
B2	If a > 0, set t ₂ = 1, otherwise t ₂ = 0	t ₂ = 0	t ₂ = 1	t ₂ = 0
ZC ₃	BM >> t ₂	BM = 1000	BM = 0010	BM = 0100

FIG. 4

Id., Fig. 4. Instructions ZC₁ and ZC₂ set provisional values in BM, which are subject to change based on distinguishing information (the values of t₁ and t₂) set by the commands that follow. EX1005, Fig. 4, 6 ¶4-5. Depending on the values of t₁ and t₂, the intermediate value of BM may be shifted left or right to generate the bit pattern corresponding with the value CCW of the condition code on the source machine.

EX1005, 4. As the petition explains, the bits stored in Loderer's register are an encoding of target machine condition codes because they represent the condition codes that would result from executing the source machine instruction on the IBM/390 computer, the source machine. Paper 1, 29; *see also id.*, 18-22; *see also* POR, 19 (quoting EX1005, 2).

V. PO'S ATTEMPT TO LIMIT THE TARGET MACHINE ARCHITECTURE IS BASED ON EXTRINSIC EVIDENCE AND INCONSISTENT WITH PO'S LITIGATION POSITIONS, THE CLAIM LANGUAGE, AND THE INTRINSIC RECORD

A. Summary of PO's Argument Based on Extrinsic Evidence

PO contends all claim terms should be given their plain and ordinary meaning. *See* POR, 26 ("IBM adopts the Board's application of the plain and ordinary 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.

At its core, the parties' dispute appears to be whether a "target machine condition code" includes bits of data stored in a register at the target machine that represent the outcome status of executing the source machine instruction on the

source machine. Despite sidestepping claim construction entirely in the POR⁶, PO apparently interprets “target machine condition code” as limiting the architecture of the target machine to an architecture that provides certain dedicated support for condition codes, such as a program-status word (“PSW”) in the IBM/390 or a dedicated “condition code register” in other systems. *See* POR, 5-6, 16-23. According to PO, a condition code value that is directly calculated without branch instructions but stored in a “condition register” or general-purpose register at the target machine cannot, by definition, be a “target machine condition code” of the challenged claims. *See* POR, 5-6 (PO arguing that MIPS uses only “condition registers,” not “condition codes”); *id.*, 18 (“The MIPS architecture [of Loderer] provides no condition codes”); *id.*, 27-28 (PO arguing that “no hardware support of a condition code is provided” in the MIPS architecture); *id.*, 32 (“Loderer stores the bit pattern in a general-purpose register.”); *id.*, 33 (“That Loderer teaches a bit pattern stored in a general-purpose register—not a ‘target machine condition code’—is fatal to LzLabs’ Petition.”).

⁶ Claim construction arguments not presented in the POR should be deemed waived. *See* Paper 8, 9 (“Patent Owner is cautioned that any arguments not raised in the response may be deemed waived.”).

The POR begins with a “Technology Background” section based exclusively on extrinsic evidence. *See* POR, 2-6. In this section, PO ignores the specification entirely, including its broad description of “condition codes,” and defines condition codes as “**special** bits set by ALU operations.” POR, 6; *see id.*, 2-3 (“Condition codes are ‘**special** bits’ that represent hardware state information used in computer architectures to hold information about the results of various hardware operations”). The ’289 specification does not describe condition codes as “special bits,” and neither the POR nor PO’s expert explains what would make bits of data “special” to qualify as a condition code.⁷ Nor does the POR or PO’s expert explain why the bits stored in general purpose register rC of the disclosed embodiments are special

⁷ To the extent PO contends CPU flag bits and bits stored in a PSW or a “condition code register” are “special,” but bits stored in other kinds of storage (e.g., a general-purpose register or “condition register”) are not, PO’s argument is directly contrary to the teachings of the ’289 patent, which as explained above include storing “target machine condition codes” in general-purpose register rC. Unlike the challenged claims, other claims of the ’289 patent require the presence of “target central processing unit (CPU) flag bits.” EX1001, Claims 9, 15, and 20.

enough to be condition codes, but the bit patterns BM stored in Loderer's register are not.⁸

PO describes a "condition register" as an alternative approach to condition codes:

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.

⁸ Remarkably, the POR and PO's expert declaration omit any discussion of the preferred embodiments of the '289 patent or the disclosed examples of a "target machine condition code."

POR, 6.⁹ PO identifies Digital Equipment Corporation's (DEC) Alpha architecture and the MIPS architecture as “[e]xemplary architectures that implement ‘condition registers.’” *Id.*

Based on this extrinsic evidence, PO reasons that “target machine condition code” in the challenged claims excludes target architectures that use “condition registers” and requires the target architecture to have certain kinds of specialized or dedicated elements for storing bits of data representing the outcome status of instruction execution:

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.

⁹ To the extent PO contends “target machine condition codes” must be updated as a “side-effect” of ALU operations, PO’s contention is meritless because the challenged claims require direct calculation of the target machine condition code. *See* POR, 6 (“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.” (citing EX1024, 4)).

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.

POR, 10.

PO contends Loderer does not disclose a “target machine condition code” because “the target architecture contemplated in Loderer is a MIPS architecture, which, as shown in Figure B.16 of EX2004, implements a ‘condition register’ instead of ‘condition codes.’” POR, 18-19; *see also id.*, 19 (“Loderer lacks a target architecture condition code”¹⁰), *id.*, 17 (“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.*’”). PO distinguishes Loderer’s

¹⁰ The parties dispute whether Loderer discloses a “target *machine* condition code.” Neither the ’289 specification nor the challenged claims use the term “target *architecture* condition code” recited in the POR.

bit pattern BM because “Loderer stores the bit pattern in a general-purpose register.”

¹¹ POR, 32; *see also id.*, 29 (“Loderer teaches that ‘a memory area or a separate register is provided on the target hardware.’”).

¹¹ The POR suggests Loderer’s BM is distinguishable from a “condition code” because it is a “bit pattern” simulating a condition code. *See* POR, 19 (“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.’”); *id.*, 22 (“Mr. Jestice admitted, Loderer’s bit pattern is not a condition code”); *id.*, 32 (“Loderer discloses a source machine condition code, which is distinguished from Loderer’s target machine bit pattern.”). PO offers no explanation why Loderer’s BM does not satisfy the ’289 patent’s broad description of condition codes as “several bits of data to indicate the outcome status of instruction execution.” 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. *See* EX1004, 212 (“The examples provided in the specification for the present invention describe how to derive an encoding of 0, 1 or 2 (with binary bit patterns of 00, 01, 10, respectively) ...”).

B. PO's Argument is Inconsistent with Its Litigation Positions

PO's argument ignores PO's interpretation of "target machine condition code" in the parallel litigation between the parties. For example, PO's March 2022 complaint against Petitioner alleges the accused "target machine condition code" is **a representation of an IBM mainframe condition code** (*i.e.*, a representation of the source machine condition code) generated **on a target machine**:

Once it receives this [source machine] instruction, the SDM [accused product] 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.

EX1027, ¶143; *see also id.*, ¶144 ("For example, the procedure for generating a condition code without branches involves successive updates **to an IBM mainframe condition code representation on the target machine** based on other values such as sign bits and the like ('distinguishing information')."). PO's preliminary infringement contentions likewise confirm PO's understanding that "target machine condition code" includes emulated source machine condition codes on the target machine:

For example, SDM [accused product] generates sequence of target instructions which together operate to derive **an IBM mainframe**

condition code (CC), the “target machine condition code,” for a particular IBM mainframe source instruction, as such a CC may be required to execute a subsequent IBM mainframe source instruction.

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 infringement of the ’289 patent based on generating a representation of IBM mainframe condition codes on the target machine—precisely what the ’289 patent discloses as “target machine condition codes,” as explained above.

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 position with its earlier infringement contentions in parallel litigation.

C. PO’s Argument is Inconsistent with the Claim Language and the Intrinsic Record

In addition to being contrary to PO’s litigation position, PO’s argument is unsupported by the claim language and the intrinsic record. As explained above, the challenged claims state the target machine condition code is directly calculated by generating and executing a sequence of instructions to produce the target machine condition code. EX1001, 9:34-47. The challenged claims do not limit the

architecture of the target machine or require storing the code in a special type of storage. Similarly, the POR and PO's expert do not attempt to square PO's narrow interpretation of "target machine condition code" with the disclosed embodiments, which—like Loderer—store bits of data representing an emulated source machine condition code in a general-purpose register (e.g., register rC).

Both parties contend "target machine condition code" should be given its plain and ordinary meaning, which requires reading the term in the context of the specification and prosecution history. *Thorner v. Sony Computer Entm't Am. LLC*, 669 F.3d 1362, 1365 (Fed. Cir. 2012). But the POR and PO's expert do not address the term in the context of the preferred embodiments or address the portions of the intrinsic record that undercut PO's position, instead relying on a table from an 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 architecture and/or the target machine condition code storage location based on

extrinsic evidence that is inconsistent with the intrinsic record and inconsistent with PO's litigation positions.

As noted above, the '289 patent discloses a broad interpretation of condition codes: “[i]n virtually all modern data processing systems, the execution of various operations . . . may result in the generation of several bits of data to indicate the outcome status of instruction execution. *These bits are typically referred to as condition codes.*” EX1001, 1:18-23 (emphasis added). The '289 patent also explicitly states that the disclosed principles for generating the target machine condition code apply to “any target architecture” (*id.*, 1:47-49) and are “not limited to any particular target machine architecture” (*id.*, 3:29-32).¹² Thus, the specification supports Petitioner's mapping of Loderer's target machine condition codes and directly undercuts PO's effort to limit the architecture of the target machine.

¹² The POR and PO's expert conveniently omit this disclosure, suggesting the '289 patent is limited to exemplary target architectures identified in the patent. *See, e.g.*, POR, 10-11 (quoting the exemplary “PowerPC architecture and the Intel IA32 architectures [] employed to emulate the z/Architecture” as support for PO's attempt to exclude Loderer's target machine architecture, while simply ignoring the following three sentences of the specification).

Similarly, the challenged claims do not limit the target machine condition code storage location, and the specification contemplates that condition codes can be generated and stored in locations other than condition code registers:

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. Moreover, PO's attempt to distinguish "target machine condition code" from condition code values stored in a condition register or general-purpose register is inconsistent with the specification because the disclosed embodiments of the '289 patent store the target machine condition code in general-purpose register rC. EX1001, 5:20-29. And PO identified the value calculated and stored in general-purpose register rC as the "target machine condition code" during prosecution. EX1004, 83-84.

VI. Conclusion

Loderer addresses the same problem and discloses the same solution as the '289 patent, including directly calculating condition code settings on the target machine without branch instructions. Petitioner respectfully requests that the challenged claims be cancelled under 35 U.S.C. §318(b).

Respectfully submitted,

December 22, 2023

Date

/s/ *Joseph D. Gray*

Joseph D. Gray (Reg. 61,803)
Lead Counsel for Petitioner

CERTIFICATE OF COMPLIANCE

Pursuant to 37 C.F.R. § 42.24(d), the undersigned certifies that the foregoing Petitioner's Reply, exclusive of the exempted portions as provided in 37 C.F.R. § 42.24(c), contains no more than 5,531 words and therefore complies with the type-volume limitations of 37 C.F.R. § 42.24(c). The word count was calculated by starting with Microsoft Word's total document word count and subtracting the words for the Table of Contents, the Exhibit List, the Certificate of Compliance, and the Certificate of Service.

This paper complies with the general format requirements of 37 C.F.R. 42.6(a) and has been prepared using Microsoft Word in 14-point Times New Roman.

December 22, 2022

Date

/s/ Joseph D. Gray

Joseph D. Gray (Reg. 61,803)
Tecuan Flores (Reg. 77,668)

Slayden Grubert Beard PLLC
401 Congress Avenue, Suite 1650
Austin, Texas 78701

Christopher V. Ryan (Reg. No. 54,759)
RyanIPLaw, PC
1508 Bay Hill Drive
Austin, TX 78701-4078

Counsel for Petitioner

CERTIFICATE OF SERVICE

In accordance with 37 C.F.R. §§ 42.6(e), the undersigned certifies that on the 22nd day of December 2023, a complete and entire copy of **PETITIONER’S REPLY** and all supporting exhibits were served on the following via Electronic Mail:

tkonstantakopoulos@desmaraisllp.com
yha@desmaraisllp.com
jwilcox@desmaraisllp.com
ibmlzlabsservice@desmaraisllp.com

Respectfully submitted,

December 22, 2023
Date

/s/ Joseph D. Gray

Joseph D. Gray (Reg. 61,803)
Tecuan Flores (Reg. 77,668)

Slayden Grubert Beard PLLC
401 Congress Avenue, Suite 1650
Austin, Texas 78701

Christopher V. Ryan (Reg. No. 54,759)
RyanIPLaw, PC
1508 Bay Hill Drive
Austin, TX 78701-4078

Counsel for Petitioner