

UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

LZLABS GMBH,
Petitioner,

v.

INTERNATIONAL BUSINESS MACHINES CORPORATION,
Patent Owner.

IPR2022-01571
Patent 7,434,209 B2

Before KEN B. BARRETT, MICHELLE N. WORMMEESTER, and
ARTHUR M. PESLAK, *Administrative Patent Judges*.

WORMMEESTER, *Administrative Patent Judge*.

DECISION
Denying Institution of *Inter Partes* Review
35 U.S.C. § 314

I. INTRODUCTION

LzLabs GmbH (“Petitioner”) filed a Petition (Paper 1, “Pet.”) requesting *inter partes* review of claims 1, 2, 37, 39, 40, 75, 77, 78, and 113 of U.S. Patent No. 7,434,209 B2 (Ex. 1001, “the ’209 patent”). International Business Machines Corporation (“Patent Owner”) filed a Preliminary Response (Paper 6, “Prelim. Resp.”). We have authority under 35 U.S.C. § 314 and 37 C.F.R. § 42.4(a). Pursuant to 35 U.S.C. § 314(a), an *inter partes* review may not be instituted “unless . . . there is a reasonable likelihood that the petitioner would prevail with respect to at least 1 of the claims challenged in the petition.” For the reasons that follow, we decline to institute an *inter partes* review of the challenged claims.

II. BACKGROUND

A. *Related Proceedings*

The parties identify one related federal district court case: *International Business Machines Corp. v. LzLabs GmbH*, No. 6:22-cv-00299-ADA (W.D. Tex.). Pet. 1; Paper 3, 1 (Mandatory Notices for Patent Owner).

B. *The ’209 Patent*

The ’209 patent describes program code conversion methods for translating program code that includes function calls. Ex. 1001, 1:8–12. To expedite program code conversion, these methods use “[a] native binding technique . . . for inserting calls to native functions during translation of subject code to target code, such that function calls in the subject program to

subject code functions are replaced in target code with calls to native equivalents of the same functions.” *Id.* at 1:51–59.

Figure 1 of the ’209 patent, reproduced below, illustrates an apparatus for implementing a program code conversion method according to the patent. Ex. 1001, 2:12–13, 2:32–33.

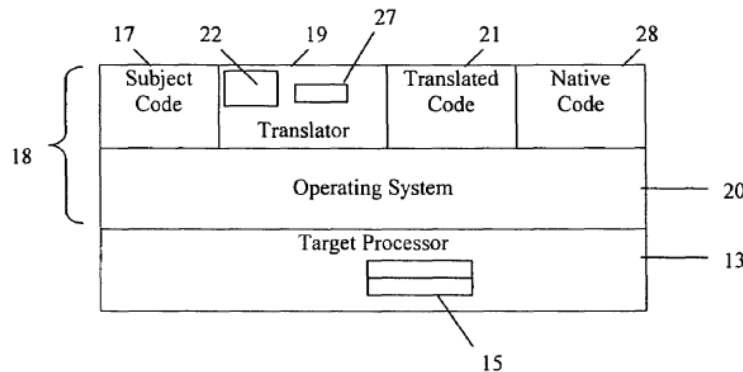


FIG. 1

Figure 1 shows in particular an apparatus with target processor 13 and memory 18. *Id.* at 2:33–36. Target processor 13 includes target registers 15. *Id.* Memory 18 includes software components, including subject code 17, operating system 20, translator 19, translated code (or target code) 21, global register store 27, native binding mechanism 22, and native code 28. *Id.* at 2:36–40. Translated code 21 is the translation of subject code 17 generated by translator 19. *Id.* at 2:47–52. Translator 19 and translated code 21 run in conjunction with operating system 20 running on target processor 13. *Id.* Native code 28 is code compiled directly for target processor 13; it is not generated by translator 19. *Id.* at 2:45–46, 2:66–3:2.

In operation, translator 19 translates subject program 17 at runtime to execute directly on the target architecture and transforms subject operating system calls made by subject program 17 so that they work properly when passed to target operating system 20. Ex. 1001, 3:8–13. To illustrate the

operational flow of an embodiment of translator 19, Figure 3 of the '209 patent is reproduced below. *Id.* at 5:3–4, 5:19–27.

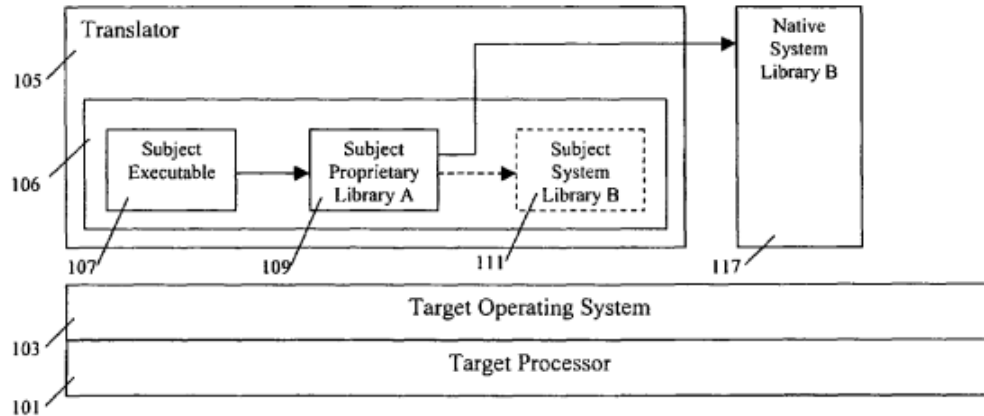


FIG. 3

Figure 3 shows translator 105 running as an executable compiled for the native architecture, which includes native operating system 103 and native processor 101. *Id.* at 5:7–12. Translator 105 translates subject program 106, which is in subject code 17. *Id.* at 5:4–6. Subject program 106 includes subject executable 107 and subject libraries, such as proprietary libraries 109 and system libraries 111. *Id.* at 5:21–24. Translator 105 uses native binding to replace subject program 106 calls to functions in subject system library 111 with calls to functions in native system library 117. *Id.* at 5:24–27. Native binding can be implemented when the subject program's flow of control enters a section of subject code 17, such as a subject library, for which a native version of the subject code exists; instead of translating subject code 17 (e.g., subject system library 111), the translator executes equivalent native code 28 (e.g., native system library 117). *Id.* at 5:40–45; *see also id.* at 1:63–67. The translator uses a mapping function to correlate subject code 17 functions with their native code 28 counterparts in order to

know which subject functions to bind and to which native functions to bind them. *See id.* at 5:63–6:2.

When making function calls to native code functions, the translator needs to comply with the calling conventions of the target architecture.

Ex. 1001, 7:13–14. Figure 6 of the '209 patent, reproduced below, is a flow diagram illustrating native function invocation. *Id.* at 2:26–28.

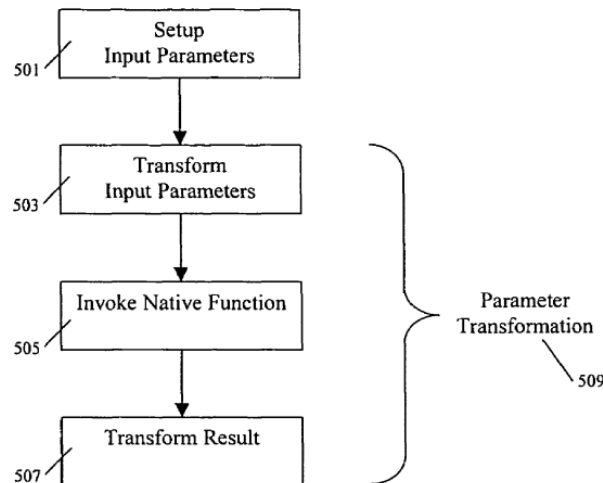


FIG. 6

Figure 6 shows steps that the translator performs to invoke a native function. *Id.* at 7:34–35. In step 501, target code 21 calculates the values of the function call parameters. *Id.* at 7:38–40. In step 503, the translator reformats the parameter values, organizing their target code 21 representations into the format and location that the native function code expects. *Id.* at 7:40–43. In step 505, target code 21 makes a function call to the native function, where the reformatted parameters are ordered according to the requirements of the function prototype (which specifies the order and type of parameters) to comply with the native calling conventions. *Id.* at 7:43–54. In step 507, the translator transforms any return value of the function. *Id.* at 7:54–55. If the function returns a value in a format

consistent with the native architecture, the translator converts it into the representation used by target code 21. *Id.* at 7:55–58.

C. Illustrative Claim

Petitioner challenges claims 1, 2, 37, 39, 40, 75, 77, 78, and 113 of the '209 patent. Claims 1, 39, and 77 are independent. Claim 1 is illustrative of the claims under challenge:

1. A method of translating subject program code executable by a subject processor into target program code executable by a target processor, said method comprising:

dividing the subject program code into a plurality of subject program code units;

translating one or more of the subject program code units into one or more target program code units; and

executing the one or more target program code units on the target processor;

wherein the translating step includes identifying a subject function in the subject program code having a corresponding native function of native code, wherein the native code is code executable by the target processor, and identifying the native function of the native code which corresponds to the identified subject function; and

wherein the executing step includes executing the native function on the target processor instead of executing a translated version of the identified subject function, including transforming zero or more function parameters from a target code representation to a native code representation, invoking the native function with the transformed zero or more function parameters according to a prototype of the native function, and transforming zero or more return values of the invoked native function from a native code representation to a target code representation.

D. Asserted Ground of Unpatentability

Petitioner challenges claims 1, 2, 37, 39, 40, 75, 77, 78, and 113 of the '209 patent on the following sole ground. Pet. 2, 23–85.

Claims Challenged	35 U.S.C. § ¹	References/Basis
1, 2, 37, 39, 40, 75, 77, 78, 113	103	Chernoff, ² FX!32, ³ Richter ⁴

In support of its arguments, Petitioner relies on a Declaration of Ian Jestice (Exhibit 1003).

Patent Owner disputes Petitioner's challenges (Prelim. Resp. 29–47) and relies on a Declaration of Martin C. Rinard, Ph.D. (Exhibit 2001) in support of its arguments.

III. DISCUSSION

A. Claim Construction

In an *inter partes* review proceeding, we construe a claim of a patent “using the same claim construction standard that would be used to construe the claim in a civil action under 35 U.S.C. 282(b).” *See* 37 C.F.R. § 42.100(b). That standard involves construing claims in accordance with the ordinary and customary meaning of such claims as would have been

¹ The Leahy-Smith America Invents Act (“AIA”), Pub. L. No. 112-29, 125 Stat. 284 (2011), amended 35 U.S.C. § 103, effective March 16, 2013. Because the application from which the '209 patent issued was filed before this date, the pre-AIA version of § 103 applies.

² Chernoff et al., U.S. Patent No. 6,000,028, issued Dec. 7, 1999 (Ex. 1005).

³ Anton Chernoff et al., *FX!32 A Profile-Directed Binary Translator*, IEEE Micro, Mar.–Apr. 1998, at 56–64 (Ex. 1007).

⁴ Jeffrey Richter, *ADVANCED WINDOWS NT THE DEVELOPER'S GUIDE TO THE WIN32™ APPLICATION PROGRAMMING INTERFACE* 26, 70, 188, 294–95, 341, 355, 418, 471, 481–82, 489–91, 496–97, 635–36 (Dean Holms et al. eds., 1994) (Ex. 1010).

understood by one of ordinary skill in the art and the prosecution history pertaining to the patent. *See id.*; *Phillips v. AWH Corp.*, 415 F.3d 1303, 1312–14 (Fed. Cir. 2005) (en banc).

Patent Owner proposes constructions for the claim terms “native code” and “target code.” Prelim. Resp. 15–17. Petitioner does not propose any claim constructions. Pet. 22. 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. & Eng’g, Inc.*, 200 F.3d 795, 803 (Fed. Cir. 1999).

B. Obviousness over Chernoff, FX!32, and Richter

Petitioner asserts that claims 1, 2, 37, 39, 40, 75, 77, 78, and 113 of the ’209 patent would have been obvious over Chernoff, FX!32, and Richter. Pet. 23–85. Patent Owner challenges certain aspects of Petitioner’s analysis. Prelim. Resp. 29–47. For the reasons explained below, we determine that Petitioner has not demonstrated a reasonable likelihood of prevailing on its asserted ground.

1. Overview of Asserted Prior Art

We start with an overview of the asserted prior art, Chernoff, FX!32, and Richter.

a. Chernoff

Chernoff is a U.S. patent titled “Means and Apparatus for Maintaining Condition Codes in an Unevaluated State.” Ex. 1005, code (54). The abstract of Chernoff describes the subject matter as follows:

A computer system for executing a binary image conversion system which converts instructions from a[n] instruction set of a first, non[-]native computer system to a second, different, native computer system, includes a[] run-time system which in response to a non-native image of an application program written for a non-native instruction set provides a[] native instruction or a native instruction routine. The run-time system collects profile data in response to execution of the native instructions to determine execution characteristics of the non-native instruction. Thereafter, the non-native instructions and the profile statistics are fed to a binary translator operating in a background mode and which is responsive to the profile data generated by the run-time system to form a translated native image. The run-time system and the binary translator are under the control of a server process. The non-native image is executed in two different enviro[n]ments with first portion executed as an interpreted image and remaining portions as a translated image. The run-time system includes an interpreter which is capable of handling condition codes corresponding to the non-native architec[t]u[r]e.

Id. at code (57).

b. FX!32

FX!32 is an article that describes software designed to “provide fast and transparent execution of x86 Win32 applications on Alpha systems.” Ex. 1007, 56. According to FX!32, “[b]efore the introduction of this software, two common techniques for running an application on a different architecture than the one for which it was originally compiled were emulation and binary translation.” *Id.* FX!32 explains that its “software combines these technologies to provide both fast and transparent execution.” *Id.* Specifically, the software includes a runtime environment providing transparent execution, a binary translator (background optimizer) providing

high performance, and a server coordinating them. *Id.* FX!32 further explains,

The first time an x86 application runs, all of the application is emulated. Together with transparently running the application, the emulator generates an execution profile describing the application's execution history. The profile shows which parts of the application are heavily used (for each user) and which parts are unimportant or rarely used. While the first run may be slow, it "primes the pump" for additional processing. Later, after the application exits, the profile data directs the background optimizer to generate native Alpha code to replace all the frequently executed procedures. The next time the application runs, native Alpha code is used and the application executes much faster. This process repeats whenever a sufficiently enlarged profile shows that it is warranted.

Id. FX!32 states that "[t]hree significant innovations of [its software] include transparent operation, interface to native APIs, and, most importantly, profile-directed binary translation." *Id.*

c. Richter

Richter is a book, which describes itself as a "developer's guide to the Win32 application programming interface." Ex. 1010, 1 (title) (capitalization omitted).

2. Analysis

Turning to the claims, we address independent claims 1, 39, and 77 together. Claim 1 is directed to a "method of translating subject program code executable by a subject processor into target program code executable by a target processor." Claim 39 is directed to a "computer-readable storage medium having software resident thereon in the form of computer-readable

code executable by a computer having a target processor.” Claim 77 is directed to a “computer apparatus.” Claims 39 and 77 recite similar limitations as claim 1. Petitioner relies primarily on its discussion of claim 1 for claims 39 and 77. Pet. 84–85. Accordingly, we focus on the language of claim 1 in particular. Our analysis of claim 1 applies to claims 39 and 77.

Claim 1 recites “identifying a subject function in the subject program code having a corresponding native function of native code” and “identifying the native function of the native code which corresponds to the identified subject function.” For these limitations, Petitioner points to the x86 code described in Chernoff and FX!32 as “subject program code,” and the Alpha code described in Chernoff and FX!32 as “native code.” Pet. 31, 43. According to Petitioner, Chernoff and FX!32 teach that

native functions and corresponding native code are identified in two ways: (1) jacketing external functions: i.e., identifying external functions, such as Windows OLE or Windows API library calls in the subject program, that have corresponding native (Alpha) functions and corresponding native (Alpha) code; and (2) background translation: i.e., identifying native code that corresponds to subject functions and was compiled for the target (Alpha) machine using the background translation process.

Id. at 42. We discuss the two ways in turn.

a. Jacketing External Functions

With respect to jacketing external functions, Petitioner contends that “FX!32 and Chernoff disclose function jacketing that identifies function calls to external libraries in the x86 (subject) code that have native counterparts on the Alpha (target) machine and the corresponding native code.” Pet. 43; *see also id.* at 43 (citing Ex. 1007, 57–58); *id.* at 43–44

(citing Ex. 1005, 38:10–15, 47:20–48:20, Figs. 31, 38, 39). Regarding Chernoff’s teachings in particular, Petitioner asserts that Chernoff’s “non-native loader identifies calls to x86 functions (subject functions) that have corresponding native functions and native code that is jacketed.” *Id.* at 44. To illustrate, Petitioner provides an annotated version of Figure 38 of Chernoff, reproduced below. *Id.*

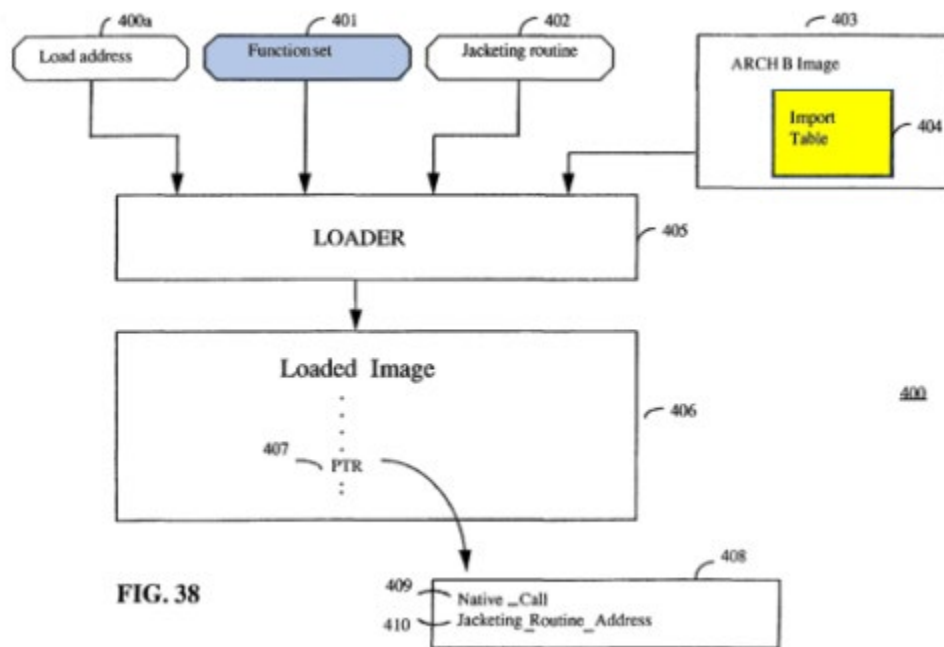


Figure 38 of Chernoff, as annotated by Petitioner, is a diagram of “a system for load time processing to support interception of functions which take a pointer to an object as a parameter.” *See* Ex. 1007, 5:42–44. According to Petitioner, Figure 38 shows in particular “an x86 program (subject program) being loaded.” Pet. 31. Petitioner directs us to where Chernoff teaches “[t]he code image 403 is for example a non-native code image developed for an Architecture B, and including an import table 404,” which “includes a list of functions or routines which are invoked from the image 403, but which

are not implemented within the image 403.” Ex. 1005, 47:34–39 (cited by Pet. 31, 44). Referring to annotated Figure 38, Petitioner asserts that “Import Table 404 (yellow) indicates the external function calls in the non-native x86 program,” and that “[f]unction set 401 (blue) indicates which x86 functions have corresponding replacement native calls.” Pet. 44 (citing Ex. 1005, 47:36–39, 44–58). Petitioner further asserts that “[t]he loader identifies the function calls of the subject program having a corresponding native function and inserts calls to the Native_Call and jacketing routines into the non-native image (408).” *Id.* (citing Ex. 1005, 47:36–39, 47:48–58). Additionally, Petitioner asserts that “[w]hen the program reaches these calls, the jacketing routine running on Alpha machine replaces the interface to the corresponding native function and invokes the native code.” *Id.* (citing Ex. 1005, 48:4–20, Figs. 31, 39). Petitioner relies on the declaration testimony of Mr. Jestice. *Id.* at 42–43, 45 (citing Ex. 1003 ¶¶ 91–95, 129–33).

Patent Owner responds that “the claimed ‘subject function’ is ‘in the subject program code,’” but “Petitioner maps functions in the prior art that are—according to its own statements—**not part of** what Petitioner contends is the subject program code in the prior art.” Prelim. Resp. 32. As support, Patent Owner points to Petitioner’s assertion that “FX!32 and Chernoff disclose function jacketing that identifies function calls to external libraries in the x86 (subject) code that have native counterparts on the Alpha (target) machine and the corresponding native code.” Pet. 43 (cited by Prelim. Resp. 33). According to Patent Owner, Petitioner “fails to meet its burden” for institution. *Id.* at 35.

We agree with Patent Owner. Claim 1 recites “a subject function in the subject program code.” As discussed above, Petitioner identifies the x86 code described in FX!32 and Chernoff as the recited subject program code. Pet. 31, 43. For example, Petitioner identifies Chernoff’s image 403 as subject program code. *Id.* Petitioner further identifies the x86 functions described in Chernoff as meeting the recited subject function. *Id.* at 44. Chernoff teaches that these “functions or routines . . . are invoked from the image 403, but . . . are not implemented within the image 403.” Ex. 1005, 47:34–39 (cited by Pet. 44). On the record before us, we are not persuaded that Petitioner has shown sufficiently how functions that are *not implemented within* image 403 disclose the recited “subject function *in* the subject program code” (emphasis added) of claim 1.

We note that Petitioner also identifies a Windows API (e.g., x86 Win32 API) as described in FX!32 as the recited subject function. Pet. 42–43. As Patent Owner points out, however, FX!32 teaches that “Win32 applications make calls to routines that are not part of the application, specifically the Win32 API,” where “these are x86 applications.” Ex. 1007, 58 (cited by Prelim. Resp. 33). Moreover, Petitioner directs us to where FX!32 teaches that “in Windows NT most system APIs are part of the operating system.” Pet. 43 (quoting Ex. 1007, 58). Petitioner’s declarant Mr. Jestice testifies that “calls to operating system library functions are examples of functions that would be identified in an image, but not implemented within the image.” Ex. 1003 ¶ 113 (cited by Pet. 32). As with the functions and routines in Chernoff, we are not persuaded that Petitioner has shown sufficiently how the Win32 API in FX!32, which is *not part of*

the x86 application, discloses the recited “subject function *in* the subject program code” (emphasis added) of claim 1.

b. Background Translation

With respect to background translation, Petitioner asserts that “Chernoff describes background system (34) employs a binary image transformer to generate native code, based on profile statistics, and stores the native code in logical disk drive storage 17c’.” Pet. 45 (citing Ex. 1005, 7:57–8:6). To illustrate, Petitioner provides an annotated version of Figure 2 of Chernoff, which is reproduced below. *Id.*

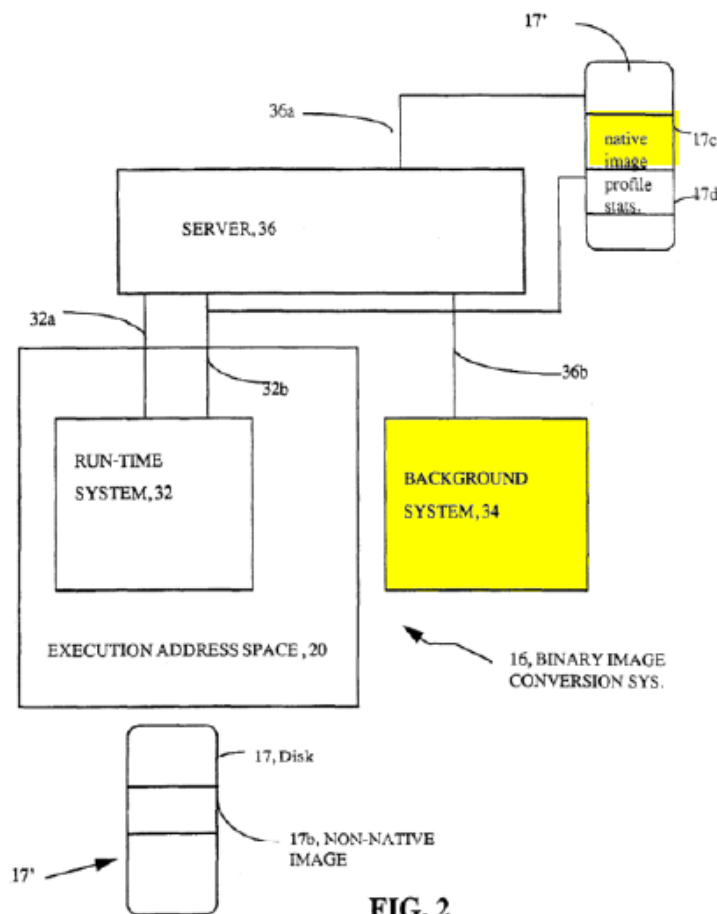


Figure 2 of Chernoff, as annotated by Petitioner, is a diagram of dual stage instruction conversion system 16 including run-time system 32 and background system 34. *See* Ex. 1005, 4:15–17, 9:14–17, 10:21–25.

Petitioner directs us to where Chernoff teaches,

Prior to performing a conversion by the run time system 32, the run-time system 32 interrogates the server process 36 via a path 32a to determine from the server process whether there is a native image corresponding to the routine of the application program stored in segment 17b whose execution has just been requested by a user. . . . If there is code in existence for the particular instruction reached in the application program, due to a prior execution in the run-time system and subsequent conversion by a background system, the run-time system 32 will request and execute the native code.

Pet. 46–47 (quoting Ex. 1005, 9:60–10:5). According to Petitioner, “[t]he native Alpha code generated by Chernoff’s background process is generated external to the interpreter and optimized for the Alpha (target) machine,” and, “[t]hus, it is ‘native code’ of the ’209 patent.” *Id.* at 47. Here, the interpreter refers to Chernoff’s interpreter 44, which is a part of run-time system 32. Ex. 1005, 11:65–67, Fig. 3. Petitioner asserts that Chernoff’s “run-time interpreter [is] similar to the ‘translator’ in the ’209 patent.”

Pet. 40.

We note that Chernoff teaches implementing its background system 34 as a binary translator. Ex. 1005, 10:42–45. Petitioner asserts that FX!32 also teaches a binary translator, and points to FX!32 for additional details regarding the generation, identification, and execution of native code for native functions. Pet. 46–47, 49 (citing Ex. 1007, 56, 58–60).

In response, Patent Owner contends that the native Alpha code generated by the binary translator in Chernoff and FX!32 does not disclose

the recited native code of claim 1. Prelim. Resp. 44–47. As support, Patent Owner asserts that “the claimed ‘native code’ is ‘code compiled directly for the target architecture,’” and that it “is not the result of any translation.” Prelim. Resp. 45; *see also* Ex. 1001, 2:60–63 (“‘[N]ative code’ 28[] . . . is compiled directly for the target architecture”), 2:66–3:2 (“Native code 28 is generated external to the translator 19, meaning that the translator 19 does not generate the native [c]ode 28.”). Patent Owner points to Petitioner’s assertion that Chernoff teaches that its background system 34 uses a “**binary image transformer to generate native code**.” Prelim. Resp. 45 (quoting Pet. 45) (emphasis added by Patent Owner). Patent Owner also points to FX!32’s teaching that its binary translator uses an execution profile “**to translate executed x86 code into native Alpha code**.” *Id.* (quoting Ex. 1007, 60) (emphasis added by Patent Owner). According to Patent Owner, “the ‘native Alpha code’ in FX!32 and Chernoff is code **originally compiled for the x86 architecture** that is translated into Alpha code,” and “without translation, the code could not execute in the Alpha architecture.” *Id.*; *see also id.* at 46–47. Patent Owner relies on the declaration testimony of Dr. Rinard. *Id.* at 45–47 (citing Ex. 2001 ¶¶ 108–122).

We agree with Patent Owner. Chernoff teaches that its background system 34 “convert[s] portions of the non-native image into a native image corresponding to the operation and function of those portions of the non-native application program.” Ex. 1005, 8:3–7; Pet. 45 (citing Ex. 1005, 7:57–8:6). In other words, the native Alpha code generated by Chernoff’s background system 34 is a translated version of the non-native x86 code. The ’209 patent teaches that its “translator 19 does not generate the native [c]ode 28.” Ex. 1001, 2:66–3:2. As discussed above, Petitioner contends

that Chernoff's run-time interpreter 44 corresponds to the '209 patent's translator 19, and that the native Alpha code generated by Chernoff's background system 34 therefore is native code because it "is generated external to the interpreter." Pet. 40, 47. Claim 1, however, additionally requires "executing the native function on the target processor *instead of executing a translated version* of the identified subject function" (emphasis added). For this requirement, Petitioner asserts that "Chernoff and FX!32 each disclose . . . *code for native functions created by the background translator are executed*, instead of executing translated versions of those functions." Pet. 53 (emphasis added). On the record before us, we are not persuaded that Petitioner has shown sufficiently how native Alpha code generated by Chernoff's background system 34 discloses the recited claim limitation "native code," which claim 1 itself distinguishes from "a translated version" of the subject (non-native) code.

3. Summary

In view of the foregoing, we determine that Petitioner has not demonstrated a reasonable likelihood of showing that independent claims 1, 39, and 77 would have been obvious over Chernoff, FX!32, and Richter. Claims 2, 37, 40, 75, 78, and 113 depend from claims 1, 39, or 77. Petitioner's contentions regarding these dependent claims rely on the same analysis for the limitations of claim 1 discussed above. *See* Pet. 76–85. Accordingly, we also determine that Petitioner has not demonstrated a reasonable likelihood of showing that dependent claims 2, 37, 40, 75, 78, and 113 would have been obvious over Chernoff, FX!32, and Richter.

IV. CONCLUSION

For the foregoing reasons, Petitioner has not demonstrated on the record before us a reasonable likelihood that it would prevail in showing the unpatentability of any of the challenged claims of the '209 patent.

V. ORDER

For the reasons given, it is

ORDERED that the Petition is *denied* and no trial is instituted.

PETITIONER:

Joseph D. Gray
Chris V. Ryan
Tecuan Flores
Slayden Grubert Beard PLLC
jgray@sgbfirm.com
chris.ryan@ryaniplaw.com
tflores@sgbfirm.com

PATENT OWNER:

Theodoros Konstantakopoulos
Yung-Hoon Ha
DESMARAIS LLP
tkonstantakopoulos@desmaraisllp.com
yha@desmaraisllp.com