United States District Court,
W.D. Texas, Austin Division.

VIA TECHNOLOGIES, INC. and,
v.
INTEL CORPORATION.

No. A-01-CA-602-SS

March 6, 2003.

Joy A. Arnold, Birch M. Harms, Bryan J. Vogel, David G. Lindenbaum, Derek M. Kato, Franciscus A. Ladejola-Diaba, Gene W. Lee, Jeffrey D. Blake, John M. Hintz, Khue V. Hoang, Laurence S. Rogers, Robert C. Morgan, Sasha G. Rao, Theresa A. Moehlman, Edward W. Bailey, Fish & Neave, New York, NY, Christa P. Worley, Norman H. Beamer, Fish & Neave, Jennifer A. Ochs, Robert P. Feldman, Rodney G. Strickland, Jr., Wilson, Sonsini, Goodrich & Rosati, Palo Alto, CA, Lin Hughes, Patton G. Lochridge, McGinnis, Lochridge & Kilgore LLP, Austin, TX, Richard L. Rainey, Roderick R. Mckelvie, Fish & Neave, Washington, DC, for Via Technologies, Inc. and Centaur Technology, Inc.

Anthony W. Shaw, Cecil E. Key, Cono A. Carrano, Robert A. King, Dewey Ballantine, LLP, Washington, DC, James J. Elacqua, Andrew N. Thomases, Jeannine Y. Sano, Saxon S. Noh, Dewey Ballantine, LLP, Palo Alto, CA, Kevin S. Kudlac, Weil, Gotshal & Manges LLP, Darryl Adams, Stephen J. Rosenman, Dewey Ballantine, LLP, Steven J. Wingard, Scott, Douglass & McConnico, L.L.P., Austin, TX, for Intel Corporation.

ORDER

SAM SPARKS**, District Judge.**

BE IT REMEMBERED on the *6th* day of March 2003 the Court reviewed the file in the above-styled cause, specifically the Report and Recommendations of the Special Master regarding the patents-in-suit [# 272-29] and the parties' objections thereto [# 288, 291]. Having considered the Report and Recommendations, the parties' objections, the testimony and evidence from the *Markman* hearing, the case file as a whole and the applicable law, the Court enters the following opinion and orders.

Analysis of the Parties' Objections

I. '188 Patent

Intel Corporation ("Intel") did not file any objections to the Special Master's construction of the claims in the '188 patent. The plaintiffs, Via Technologies, Inc. and Centaur Technology, Inc. ("Via") did file objections, which the Court discusses below.

A. buffer/control logic

Via objects to the Special Master's construction of "buffer/control logic" on the grounds that the construction limits the inputs and outputs to the buffer/control logic to those in the preferred embodiment in Figure 4. The Court agrees the Special Master's construction of this term describes what the buffer/control logic does, rather than what it is. This description is redundant because the claim language following the term describes what the buffer/control logic is configured to do. The Court finds the plain language of the term, combined with the description of the term's function in the claim language, is a sufficiently clear explanation of the term for the jury and the Court need not construe the term. Accordingly, the Court sustains in part the objection and removes the Special Master's construction of buffer/control logic.

B. selective clock multiplier

Via objects to the Special Master's recommended construction of "selective clock multiplier," arguing the construction requires the selective clock multiplier to receive inputs from the buffer/control logic and the bus clock signal and clock ratio signals. Via contends these requirements only occur in the preferred embodiment in Figure 4 of the patent. The Court finds the Special Master's construction is consistent with the claim language, which states "a selective clock multiplier, couple to said buffer/control logic, configured to receive the bus clock signal and to generate the core clock signal ..." and "a selective clock multiplier, for receiving a system bus clock signal, and for generating the core clock signal from said system bus clock signal...." '188 Patent, Claims 12; 19.

Additionally, the construction is consistent with Via's counsel's explanation at the hearing that "all of [the claims] that recite a selective clock multiplier indicate that it-as input to it, it will have the output of the buffer/control logic and the bus clock." Markman Transcript, Vol. 1, at 45. As for the clock ratio signals, Via's counsel explained the selective clock multiplier "would still have the input to the processor of these ratio signals, but it just may not necessarily be to that specific piece of circuitry; it may be coming through another route." *Id*. at 46. When the Special Master asked, "So the information is somehow getting to the selective clock multiplier but not directly to it?," Via's counsel answered, "Right." *Id*. In other words, Via conceded the selective clock multiplier does receive the clock ratio signals as input, but argued the signals do not have to come through the circuitry depicted in Figure 4. The Special Master's construction, however, does not require the circuitry shown in Figure 4, but merely requires the clock ratio signals to be inputs into the selective clock multiplier. Accordingly, Via's objection is overruled.

C. test control logic

Via objects to the Special Master's construction of "test control logic" on the grounds that it requires the test signal to come from test equipment (thereby limiting the invention to the preferred embodiment in Figure 4) and requires the test control logic to "re-enable" control of the clock ratio via the control signals. In the Special Master's construction, the test control logic responds to "a signal from test equipment." The claim language limits the test control logic to use "during test of the microprocessor." '188 Patent, Claim 21. To be tested, the microprocessor must receive a signal from some sort of test equipment. Therefore, the Court finds the Special Master's construction does not improperly limit the invention to the preferred embodiment.

Via also contends the "re-enable" language limits the test control logic to situations where the fuse is blown, even though the claim language states "regardless of whether said fuse is blown." '188 Patent, claim 21. The Special Master's construction does not require the fuse to be blown. The word "re-enable" merely emphasizes that the selective clock multiplier ignores any fixed ratio during testing so the clock ratio signals can prescribe the multiple for the clock ratio. The claim language clarifies that the fuse need not be blown.

Via's objection is overruled.

D. fuse

Via objects to the Special Master's construction of the term "fuse" when it appears in the '188, '679 and '735 patents. Via contends the Special Master's construction improperly limits the term to a particular type of fuse. The Special Master's construction of fuse requires an "electrical connection between two points;" in the '188 patent, the claim language states the fuse provides "an electrical signal path." '188 Patent, claim 16. Via points to language in the '735 specification stating "[o]ther fuse structures can be used" (other than those depicted in Figures 3A and 3B) and "other materials (other than polysilicon) can be used to construct the fuses." '735 Patent, 5:65-67. Although the patents allow for varied fuse structures and materials, they require the fuses to provide an electrical connection between two points. Therefore, Via's objection is overruled.

II. '311 Patent

A. floating point register file

Intel objects to the Special Master's construction of "floating point register file" on the grounds that it does not specify the floating point register file must be physically separate from the integer register file. While Intel pressed this argument fervently in its *Markman* briefs and at the hearing, the claim term itself does not require physical separation. The separation between the floating point register file and the integer register file is revealed elsewhere in the claim language, in the requirement that the floating point register file be "coupled to" the integer register file. '311 Patent, claim 1. Because inserting that requirement into the construction of floating point register file would be redundant and simply incorrect, Intel's objection is overruled.

B. translator

Via objects to the proposed construction of "translator" because it states the translator outputs "operations" instead of micro-operations. Via contends the patent specification includes a definition of translator in column 4, lines 37 through 45. The Court holds that passage is not an express definition of a translator but a description of what the translator depicted in Figure 2 does. Additionally, Figure 5 does not require the translator to output a particular type of instruction (i.e.micro-operations). Therefore, "operations" is more appropriate in the construction of translator than "micro-operations," and Via's objection is overruled.

C. instruction, conversion instruction, move instruction

Via objects to the Special Master's construction of "instruction," "move instruction" and "conversion instruction" as not specifying the instructions are translated instructions, or micro-operations. However, the Court finds the claim language clearly indicates the instructions are "provided by a translator" and the definition of instruction need not repeat that requirement. '311 Patent, claims 1, 14. Via's objection is therefore overruled.

III. '**508 and** '748 Patents

A. single logical register file

Intel objects to the Special Master's decision not to construe the term "single logical register file" in the '508

and '748 claim constructions. The Special Master did define the terms "register," "register file," "physical register file" and "logical register file." Given those constructions, the Court finds further construction of "single logical register file" unnecessary and overrules Intel's objection.

B. transition unit

Via objects to the construction of "transition unit," arguing the construction should clarify the transition unit causes transitions from floating point mode to packed data mode and vice versa. The construction states the transition unit "causes the transition between floating point mode and packed data mode," which leaves open the possibility that the transitions could be back and forth.

Via also contends transition unit should be construed as a means-plus-function term under 35 U.S.C. s. 112, para. 6 because, even though the term does not contain the "means for" language, the term "relies on functional terms other than structure or material to describe performance of the claimed function." Micro Chem. Inc. v. Great Plains Chem. Co., 194 F.3d 1250, 1257 (Fed.Cir.1999). Section 112, paragraph 6 of the patent statute provides: "An element in a claim for a combination may be expressed as a means or step for performing a specified function without the recital of structure, material, or acts in support thereof, and such claim shall be construed to cover the corresponding structure, material, or acts described in the specification and equivalents thereof." 35 U.S.C. s. 112, para. 6. However, if an inventor claims his invention under this provision and fails to disclose adequate structure or acts for performing the function, the patent is invalid as indefinite.

Use of the word "means" creates a rebuttable presumption that s. 112, para. 6 applies, whereas failure to use the word "means" creates a presumption that it does not apply. Personalized Media Comm., LLC v. Int'l Trade Comm'n, 161 F.3d 696, 703-04 (Fed.Cir.1999). In deciding whether either presumption has been rebutted, courts focus on "whether the claim as properly construed recited sufficiently definite structure to avoid the ambit of s. 112, para. 6." Personalized Media, 161 F.3d at 704. The Court finds a person of ordinary skill in the art would understand the term "transition unit" to connote sufficient structure. At the hearing, Via's expert testified that one of ordinary skill in the art could figure out the structure of the transition unit in Figure 6A from the flowchart depicted in Figure 9 of the patent, although he had not yet figured it out. *See* Markman Transcript, at Vol. 3, p. 45-46. Therefore, Via has not rebutted the presumption that section 112, paragraph 6 does not apply to this term, and its objection is overruled.

C. transition means

Via objects to the construction of "transition means ..." because the function of the means-plus-function does not specify the term causes two physical register files to logically appear to software executing on the processor as a single logical register file. However, the Court finds that language is not necessary in the description of the term's function because it is already included in the claim language itself.

Via also objects to the exclusion of some of the steps in Figures 8 and 9 in the construction of the structure of the term. However, the patent specification demonstrates not all steps in Figures 8 and 9 will be used in all embodiments of the invention. '748 Patent, at 28:27-41. Accordingly, this objection is overruled.

D. causing said first physical register file and said second physical register file to appear to software as a single logical register file

Via contends the Special Master should have construed this term as step-plus-function under section 112,

paragraph 6. The Special Master did not construe this term at all, although he construed the terms register, register file, physical register file and logical register file.

In method claims where patentees describe a step for performing a certain function or process without describing corresponding acts demonstrating how the function is accomplished, s. 112, para. 6 is implicated. Seal-Flex, Inc. v. Athletic Track & Court Constr., 172 F.3d 836, 849-50 (Fed.Cir.1999, Rader, J., concurring); *O.I. Corp. v. Tekmar Co. Inc.,* 115 F.3d 1376, 1582-83 (Fed.Cir.1997). The Federal Circuit has acknowledged a rebuttable presumption that terms that do not contain the "step for" language will not invoke s. 112, para. 6. Masco Corp. v. United States, 303 F.3d 1316, 1326 (Fed.Cir.2002). The Federal Circuit also indicated its unwillingness to apply s. 112, para. 6 without the "step for" language in *Masco:* "Where the claim drafter has not signaled his intent to invoke s. 112, paragraph 6 by using the 'step[s] for' language, we are unwilling to resort to that provision to constrain the scope of coverage of a claim limitation without a showing that the limitation contains nothing that can be construed as an act. Method claims are commonly drafted, as in this case, by reciting the phrase [steps of] followed by a list of actions comprising the method claimed. An application of s. 112, paragraph 6 in the present circumstances would render the scope of coverage of these method claims uncertain and disrupt patentees' settled expectations regarding the scope of their claims." Masco Corp., 303 F.3d at 1327. Additionally, the simple fact that a claim term ends in "-ing" does not render it a step-plus-function term. *Tekmar,* 115 F.3d at 1583 ("If we were to construe every process claim containing steps described by an "ing" verb, such as passing, heating, reacting, transferring, etc. into a step-plus-function limitation, we would be limiting process claims in a manner never intended by Congress."). In fact, this Court has not found (and Via has not provided) a Federal Circuit case applying s. 112, para. 6 where the "step for" language is absent.

While the Court acknowledges the verb "causing" fails to describe a process as concrete as "heating" or "reacting," it finds Via has not rebutted the presumption against the application of s. 112, para. 6. This method is claimed as any other process claim, and the Federal Circuit has affirmed its unwillingness to apply the step-plus-function limitation to all method claims. The patent specification clarifies the acts through which the process is accomplished. *E.g.,* '748 Patent, 25:3-11. Because the meaning of the term is plain in light of the constructions of "register," "register file," "physical register file" and "logical register file," the Court finds no construction is necessary. Via's objection is overruled.

E. transitioning to said packed data mode; transitioning to said floating point mode

Via objects to the Special Master's construction of these terms on the grounds that he did not construe these terms as step-plus-function terms. Again, Via has not rebutted the presumption that s. 112, para. 6 only applies when the "step for" language is present. The claim language describes a method for a data processing apparatus to execute instructions by (1) receiving an instruction; (2) determining whether it is a floating point or packed data instruction and whether the processor is in floating point or packed data mode; and (3) transitioning to the appropriate mode depending on what type of instruction it received; and (4) executing the instruction on the appropriate set of registers. 748 Patent, claim 62. While these are not physical or tangible acts, they are acts performed by a microprocessor. The specification and figures in the patent describe the acts as part of a flowchart. '748 Patent, 30:44-46, 31:28-30; Figs. 7A & 7C. The Court finds the Special Master's construction, in light of the surrounding claim language and specification and diagrams in the patent, adequately describes the act performed by the claim terms. Via's objection is therefore overruled.

F. stack reference unit; fixed register file unit; non-stack reference unit

Via objects to the Special Master's use of the word "allows" in his construction of the above terms, arguing the construction could allow the claim term to performs other functions in addition to those claimed. The Court finds the word "allows" is appropriate in light of the description in the claim language of the functions these units perform. Moreover, Via's proposed constructions of these terms are confusing, and the Court overrules the objections.

G. execution unit ... to perform floating point operations ...

Via objects to the Special Master's recommended construction ("firmware, microcode and/or circuitry within the processor that performs operations on data") because it does not specify the execution unit operates on both floating point and integer data. Because the claim language demonstrates the execution unit performs both floating point and integer operations, the Court sustains Via's objection and amends the construction accordingly.

IV. '679 Patent

A. functional blocks

Via objects to the Special Master's recommended construction of "functional block" because it states the circuits and microcode are "dedicated to performing a specific function." Via suggests replacing that phrase with the words "for performing a specific function." The Court finds the Special Master's language simply makes Via's description clearer by showing the circuitry and microcode within a functional block perform a specific function.

Via also objects to the statement "[e]ach block may be individually turned on or off," arguing "turned on or off" is ambiguous and the words "enabled/disabled" should be used instead. Via contends "turned on or off" means removing power, instead of disabling the ability of the functional block to perform its intended function. The Court finds "turned on or off" does not necessarily mean power is removed from the functional block; instead, it indicates the block is unable to perform its function. Additionally, the language is clearer for a jury than "enabled/disabled." Moreover, the "enabled/disabled" language still remains in the claim language, and need not appear twice: "A microprocessor, having a plurality of functional blocks, the plurality of functional blocks being individually enabled/disabled...." '679 Patent, claim 1. Via raises the same objection to the construction of the terms "enable/disable signals," "partly determinative" and "processor instruction for writing a configuration command to said feature control register," and the Court overrules the objection to those claim terms as well.

B. fuse array control

Via objects that the Special Master's construction of "fuse array control" limits the invention to the preferred embodiment by requiring the fuse array control to send a signal to the plurality of functional blocks. Via contends the fuse array need not be directly connected to the functional blocks, as is depicted in Figure 3, the preferred embodiment. However, the Special Master's construction does not require a direct connection or even require that the signal be sent directly from the fuse array to the functional blocks; it merely requires the signal to originate from the fuse array and arrive at the functional blocks. The recommended construction allows for intermediate logic. Via raises the same objection to the construction of "enable/disable signals," and the objection is overruled as to that term as well.

Intel objects to the Special Master's failure to specify the fuse array control is "distinct" from the rest of the

microprocessor. The Court finds neither the claim language nor the diagrams and specification in the patent include such a requirement and overrules the objection.

C. feature control register

Intel objects to the construction of "feature control register" as a register that "holds at least one bit indicative of whether a functional block" should be turned on or off. Intel contends the register must hold at least two bits and must turn on or off more than one functional block. Intel points to the language in claim 1 regarding "the plurality of functional blocks being individually enabled/disabled." '679 Patent, claim 1. However, the claims specify the functional blocks are treated individually. Even if this "plurality" language in claim 1, upon which claim 12 is dependent, were a requirement, it is explicitly stated in claim 1 and need not be reiterated in the construction of "feature control register" in claim 12. Intel's objection is therefore overruled.

D. processor instruction for writing a configuration command to said feature control register

Via objects to the Special Master's construction of this term because it does not clearly indicate the processor instruction need not address all functional blocks. The construction states the processor instruction identifies "which functional blocks are turned on and which are turned off." It does not even hint that all functional blocks must be enabled or disabled. Accordingly, Via's objection is overruled.

E. logically merging

Intel objects that the construction of "logically merging" must be limited to exclusive-or operations, because that is the only example of "logically merging" given in the patent. However, the patent specification explicitly states the exclusive-or operation is only one embodiment of the invention. ' 679 Patent, 5:15-26 ("In one embodiment, the state of the fuse array 422 that was read into the TEMP register 426 is complemented, and then XOR'ed with the default value stored in the FCR register 428."). Via should not be limited to an example described in the patent. The exclusive-or operation is included in the construction, which simply states "logical operations" that involve combining values. The "merging" requirement is reflected in the "combining" language in the construction. Accordingly, the Court overrules Intel's objection.

V. '043 Patent

Via objects to the Special Master's failure to find this patent indefinite. The Special Master submitted his Report and Recommendation regarding the '043 summary judgment on March 5, 2003, and the parties have not yet submitted objections. The Court will discuss the application of section 112, paragraph 6 to this patent and the '423 patent when it enters its order accepting or rejecting the Special Master's Report and Recommendation.

Intel objects that gate 47 in Figure 5 is not included in the structure for the term "system for generating memory requests...." Upon reviewing the post- *Markman* briefs and charts, the Court notes Intel did not include gate 47 in its proposed structure, although Via did include it. Nonetheless, the Court sustains the objection, finding gate 47 should be included in the construction and amending it accordingly.

VI. '423 Patent

Via objects to the Special Master's listing of "possible structure" for the means-plus-function terms that are

the subject of Via's summary judgment motion. The Court will change this language after it rules on the summary judgment motion, upon receipt of the objections to the Report and Recommendation.

A. transition microcode program

Via contends the construction of the term "transition microcode program" should include the requirement that the program "unconditionally transitions." The Special Master construed the term "transition microcode program" only, leaving the "unconditionally transitioning" language unconstrued in the claim language. The Court finds the presence of the unconditional language in the claim language is sufficient, and the construction of "transition microcode program" need not reiterate it. Accordingly, Via's objection is overruled.

B. means for halting

Via objects that the function of "means for halting" is construed as "resetting the microprocessor" instead of "halting the microprocessor." The Court finds this construction was a typographical error and corrects it accordingly.

C. reset means

Via objects to the inclusion of microcode program 26 in Figure 2 of the patent as possible structure for "reset means...." The Court agrees no microcode should be included as structure and rejects the Report and Recommendation of the Special Master on this point. The patent specification states the reset function is "hardware oriented," and the prosecution history demonstrates Intel represented to the patent offices that the RESET pin utilizes hardware, while the INIT pin utilizes microcode. *See* '423 Patent, 5:51-55; Sept. 3 Hughes Affidavit, Ex. F at 7-8, 94-95, 108 & 122. Intel went out of its way to limit the reset function to hardware in order to get its application granted, and it must be bound by those limitations in claim construction. Therefore, Via's objection is sustained.

In accordance with the foregoing:

IT IS ORDERED that the Report and Recommendations of the Special Master regarding the patents-in-suit [# 272-29] are ACCEPTED in part and REJECTED in part, as discussed above;

IT IS FURTHER ORDERED that the attached construction of the contested patent claims will be incorporated into any jury instructions given in the above-styled cause and will be applied by the Court in ruling on the issues raised in summary judgment motions;

IT IS FINALLY ORDERED that the parties SHALL FILE a chart listing the agreed-upon terms and the agreed construction of those terms by March 14, 2003, to be incorporated into the attached chart.

UNITED STATES PATENT NO. 6.161.188

Actual Claim Language	Court's Claim Construction
CLAIM 1	
A microprocessor having a configurable core-to-bus clock ratio, the configurable core-to-bus clock ratio determining a first frequency of a core clock signal within the microprocessor, the core clock signal being	**A fuse** is a circuit element that provides an electrical connection between two

derived from a bus clock signal operating at a second frequency that is provided to the microprocessor from an external source, the microprocessor comprising:	points, but which may be irreversibly destroyed thereby electrically disconnecting the two points.
a **fuse,** configured to provide a **permanent state** that prescribes the configurable core-to-bus clock ratio;	A **permanent state** is a state that is retained upon removal of power to the microprocessor.
buffer/control logic, coupled to said fuse, configured to accept said permanent state and to direct the microprocessor to either set the configurable core-to-bus clock ratio according to the states of a plurality of clock ratio signals or to **ignore** said states and to set the configurable clock ratio to a **fixed** multiple of the second frequency.	**Buffer/control logic:** No construction necessary.
	Ignore and **fixed:** No construction necessary.
CLAIM 6	
The microprocessor as recited in claim 5, wherein, if said **fuse** is in said blown state, the second frequency is determined to be the fixed multiple of the first frequency, without regard to said states of said plurality of clock ratio signals. [Claim 5. The microprocessor as recited in claim 2, wherein, if said fuse is in said intact state, the second frequency of the core clock signal is determined to be a first multiple of said first frequency, said multiple being prescribed by a plurality of clock ratio signals.] [Claim 2. The microprocessor as recited in claim 1, wherein said **permanent state** comprises and intact state or a blown state.]	A **fuse** is a circuit element that provides an electrical connection between two points, but which may be irreversibly destroyed thereby electrically disconnecting the two points.
	A **permanent state** is a state that is retained upon removal of power to the microprocessor.
CLAIM 9	
An apparatus in a microprocessor for determining a core clock frequency of a core clock signal, the core clock signal being derived from a bus clock signal that is provided from without the microprocessor, the apparatus comprising: **ratio determination logic,** configured to provide a **fixed/variable clock ratio signal;** and **buffer/control logic,** coupled to said ratio determination logic, configured to direct the microprocessor to set a core-to-bus clock ratio based upon said fixed/variable clock ratio signal and a plurality of clock ratio signals,	**Ratio determination logic** is a logic device whose state can be permanently established, even when power is removed from the microprocessor. **A fixed/variable clock ratio signal** is an electrical signal that indicates that a variable core-to-bus clock ratio should be used, or is permanently set to indicate that a fixed core-to-bus clock ratio should be used.
wherein, if said fixed/variable clock ratio signal indicates a **fixed** core-to-bus clock ratio, said buffer control logic directs the microprocessor **to disregard** said plurality of clock ratio signals.	

CLAIM 10	
The apparatus as recited in claim 9, wherein said **ratio determination logic** comprises: a logic device, having a default state or an altered state, said states persisting when power is removed from the microprocessor; wherein, when said default state is provided by said logic device, said plurality of clock ratio signals prescribes a variable core-to-bus clock ratio for the microprocessor.	**Ratio determination logic** is a logic device whose state can be permanently established, even when power is removed from the microprocessor.
CLAIM 12	
The apparatus as recited in claim 9, further comprising: a **selective clock multiplier,** coupled to said buffer/control logic. configured to receive the bus clock signal and to generate the core clock signal,	A **selective clock multiplier** is circuitry that takes the output of the buffer/control clock signal, and clock ratio signals as inputs and generates a processor clock signal as an output.
wherein a first frequency of the core clock signal is a multiple of a second frequency of the system bus clock signal.	
CLAIM 15	
The apparatus as recited in claim 12, wherein, if said **ratio determination logic** directs the microprocessor to vary said core-to-bus clock ratio, said **selective clock multiplier** multiplies said second frequency of the bus clock signal by a factor prescribed by said plurality of clock ratio signals to set said first frequency of the core clock signal.	**Ratio determination logic** is a logic device whose state can be permanently established, even when power is removed from the microprocessor.
	A **selective clock multiplier** is circuitry that takes the output of the buffer/control clock signal, and clock ratio signals as inputs and generates a processor clock signal as an output.
CLAIM 16	
A microprocessor having a core-to-bus clock ratio, the core-to-bus clock ratio determining a first frequency of a core clock signal within the microprocessor, the core clock signal being derived from a bus clock signal provided from an external source, the microprocessor comprising:	**Ratio determination logic** is a logic device whose state can be permanently established, even when power is removed from the microprocessor.
ratio determination logic, configured to provide a **fixed/variable clock ratio signal,** said **ratio determination logic** comprising:	A **fixed/variable clock ratio signal** is an electrical signal that indicates that a variable core-to-bus clock ratio should be used, or is permanently set to indicate that a fixed core-to-bus clock ratio should be used.

a **fuse,** configured to provide an electrical signal path for said fixed/variable clock ratio signal;	A **fuse** is a circuit element that provides an electrical connection between two points, but which may be irreversibly destroyed thereby electrically disconnecting the two points.
buffer/control logic, coupled to **said ratio determination logic,** configured to convert a state of said electrical signal path into a voltage level, said voltage level being in a default state if said fuse is intact and in an altered state otherwise; and	**Ratio determination logic** is a logic device whose state can be permanently established, even when power is removed from the microprocessor.
a selective clock multiplier, coupled to said buffer/control logic, configured to receive the bus clock signal and said voltage level, for generating the core clock signal, wherein the first frequency of the core clock signal is a multiple of a second frequency of the bus clock signal, and wherein, when said voltage level is in said altered state, said selective clock multiplier is **unresponsive to** clock ratio signals, said clock ratio signals prescribing said multiple when said voltage level is in said default state.	**A selective clock multiplier** is circuitry that takes the output of the buffer/control clock signal, and clock ratio signals as inputs and generates a processor clock signal as an output.
CLAIM 19	
An apparatus in a microprocessor for generating a core clock signal having a configurable frequency or a **fixed** frequency, the apparatus comprising:	**Fixed:** No construction necessary.
a selective clock multiplier, for receiving a system bus clock signal, and for generating the core clock signal from said system bus clock signal, wherein a first frequency of the core clock signal is a multiple of a second frequency of said system bus clock signal; A **selective clock multiplier** is circuitry that takes the output of the buffer/control clock signal, and clock ratio signals as inputs and generates a processor clock signal as an output.	
buffer/control logic, coupled to said selective clock multiplier, for enabling/disabling a plurality of clock ratio signals to prescribe said multiple during normal operation of the microprocessor; and a **fuse,** coupled to said buffer/control logic, capable of being blown during fabrication of the microprocessor, wherein blowing said fuse causes said buffer/control logic **to disable** said plurality of clock ratio signals and **to fix** said multiple to a predetermined value.	A **fuse** is a circuit element that provides an electrical connection between two points, but which may be irreversibly destroyed thereby electrically disconnecting the two points.
CLAIM 21	
The apparatus as recited in claim 19, further comprising:	**Test control logic** is circuitry that, in response to a signal from test equipment, provides a signal that directs the selective clock multiplier to override the fixed core-to-bus clock ratio and re-enable control of the clock ratio via the control signals.

test control logic, coupled to said selective clock multiplier, **for enabling said plurality of clock ratio signals to prescribe said multiple during test of the microprocessor, regardless of whether said fuse is blown.**

UNITED STATES PATENT NO. 6,253,311

Actual Claim Language	Court's Claim Construction
CLAIM 1	
A microprocessor comprising: an **integer register file** configured to store a plurality of integers;	**A register** is a storage area in a microprocessor configured to temporarily store data. A **register file** is a set of registers along with read and write circuitry.
	An **integer register file configured to store a plurality of registers** is a register file which is set up to store data in integer format.
a **floating point register file,**	A **floating point register file ... configured to store a plurality of floating point numbers, said floating point register file also configured to store data in integer format** is a register file which is set up to store data in floating point format and also to store data in integer format.
coupled to said integer register file, configured to store a plurality of floating point numbers, said floating point register file also configured to store data in integer format; and	**Coupled to** means connected either directly or indirectly.
a first **conversion instruction,**	An **instruction** is a programming statement that indicates an operation for a processor to perform.
	A **conversion instruction** is a programming statement that indicates an operation for a processor to change a floating point number into an integer.
provided **by a translator,** for converting a first one of said plurality of floating point numbers within said floating point register file into a first integer, and for **temporarily storing** said first integer within said floating point register file.	A **translator** is hardware that receives an instruction as an input and as a result outputs one or more operations for a functional unit to perform.
	Temporarily storing: No construction necessary.
CLAIM 3	
The microprocessor as recited in claim 2 wherein each of said plurality of integer **registers** store 16 or 32-bit integer data.	**A register** is a storage area in a microprocessor configured to temporarily store data.
CLAIM 10	
The microprocessor as recited in claim 5 wherein at least one of said floating point **registers** is **configured to store said first integer** in either	A **register** is a storage area in a microprocessor configured to temporarily store data.

16, 32 or 64-bit format.	
CLAIM 14	
The microprocessor as recited in claim 1 wherein said translator provides a first **move instruction** for moving said first integer from said floating point register file to said integer register file .	A **move instruction** is a programming statement that indicates an operation for a processor to move data between the floating point register file and the integer register file.
CLAIM 18	
The microprocessor as recited in claim 1 wherein said **translator provides a** second **move instruction** for moving a second integer from said integer **register file** to said floating point **register file.**	A **register file** is a set of registers along with read and write circuitry.
	A **move instruction** is a programming statement that indicates an operation for a processor to move data between the floating point register file and the integer register file.
CLAIM 19	
The microprocessor as recited in claim 18 wherein said second integer is stored into said floating point **register file** in integer format.	A **register file** is a set of registers along with read and write circuitry.
CLAIM 21	
The microprocessor as recited in claim 1 wherein said first integer is transferred from said floating point **register file** to said integer **register file,** without first being stored in memory.	A **register file** is a set of registers along with read and write circuitry.

UNITED STATES PATENT NO. 5,701,508

Actual Claim Language	**Court's Claim Construction**
CLAIM 1	
In a data processing apparatus, a method for executing instructions comprising the steps of:	**An instruction type** is a set of instructions with a common data format, register addressing format or other attribute assigned to a group of instructions based on an intrinsic feature of the instruction.
executing a first set of instructions of a first **instruction type**	
on the contents of a **single logical register file,** wherein said single logical register file is operated as a flat register file while executing said first set of instructions; and	A **register** is a storage area in a microprocessor configured to temporarily store data.
Executing a first instruction of a second instruction type also on the contents of said single logical register file, wherein said single logical	A **register file** is a set of registers along with read and

register file is operated as a stack referenced register file while executing said first instruction.	write circuitry. A **physical register file** is a register file that is physically implemented as circuitry in the microprocessor.
	A **logical register file** is a register file that is seen from the user's/programmer's/operating system's perspective.
CLAIM 2	
The method of claim 1 further comprising the step of:	
Altering a plurality of tags corresponding to said single logical register file to a non-empty state sometime between starting said step of executing said first set of instructions and completing said step of executing said first instruction of said second instruction type, and wherein said plurality of tags identify whether registers in said single logical register file are empty or non-empty.	**Altering a plurality of tags:** No construction necessary. A **register** is a storage area in a microprocessor configured to temporarily store data. A **register file** is a set of registers along with read and write circuitry. A **physical register file** is a register file that is physically implemented as circuitry in the microprocessor. A **logical register file** is a register file that is seen from the user's/programmer's/operating system's perspective.
	An **instruction type** is a set of instructions with a common data format, register addressing format or other attribute assigned to a group of instructions based on an intrinsic feature of the instruction.
CLAIM 3	
The method of claim 2, wherein said step of **altering** is performed in response to either attempting to execute said first instruction or executing the first of	**Altering:** No construction necessary.

said first set of instructions .	
CLAIM 4	
The method of claim 2, wherein said step of **altering** is performed in between said step of executing said first set of instructions and executing said first instruction of said second **instruction type.**	An **instruction type** is a set of instructions with a common data format, register addressing format or other attribute assigned to a group of instructions based on an intrinsic feature of the instruction.
CLAIM 5	
The method of claim 2, wherein said step of **altering** is performed in response to attempting to execute the first of said first set of instructions.	
CLAIM 6	
The method of claim 1 further comprising the step of:	**An instruction type** is a set of instructions with a common data format, register addressing format or other attribute assigned to a group of instructions based on an intrinsic feature of the instruction.
altering a top of stack indication to an initialization value sometime between starting said step of executing said first set of instructions and completing said step of executing said first instruction of said second **instruction type,** wherein said top of stack indication identifies one **register** in said **single logical register file** as a current top of stack register.	A **register** is a storage area in a microprocessor configured to temporarily store data. A **register file** is a set of registers along with read and write circuitry. A **physical register file** is a register file that is physically implemented as circuitry in the microprocessor.
	A **logical register file** is a register file that is seen from the user's/programmer's/operating system's perspective.
CLAIM 7	
The method of claim 1 further comprising the step of:	**A register** is a storage area in a microprocessor configured to temporarily store data.
writing, in a sign and exponent field of each **register** in **said single logical**	A **register file** is a set of

register file that is written to during the step of executing said first set of instructions, a value indicating either not a number or infinity, sometime between starting said step of executing said first set of instructions and starting said step of executing said first instruction of said second **instruction type.**

registers along with read and write circuitry.

A **physical register file** is a register file that is physically implemented as circuitry in the microprocessor.

A **logical register file** is a register file that is seen from the user's/programmer's/operating system's perspective.

An **instruction type** is a set of instructions with a common data format, register addressing format or other attribute assigned to a group of instructions based on an intrinsic feature of the instruction.

CLAIM 8

The method of claim 1, wherein each tag in said set of tags corresponds to a different **register** in said **single logical register file** and identifies whether said corresponding register is empty or non-empty.

A **register** is a storage area in a microprocessor configured to temporarily store data.

A **register file** is a set of registers along with read and write circuitry.

A **physical register file** is a register file that is physically implemented as circuitry in the microprocessor.

A **logical register file** is a register file that is seen from the user's/programmer's/operating system's perspective.

CLAIM 9

The method of claim 1, wherein said step of executing said first instruction of said second **instruction type** further comprises the steps of:

An instruction type is a set of instructions with a common data format, register addressing format or other attribute assigned to a group of instructions based on an

	intrinsic feature of the instruction.
copying data contained in said **single logical register file** into a memory.	A **register** is a storage area in a microprocessor configured to temporarily store data. A **register file** is a set of registers along with read and write circuitry. A **physical register file** is a register file that is physically implemented as circuitry in the microprocessor.
	A **logical register file** is a register file that is seen from the user's/programmer's/operating system's perspective.
CLAIM 10	
The method of claim 2, wherein said step of **altering** is performed in response to executing each instruction in said first set of instructions.	**Altering:** No construction necessary.
CLAIM 11	
The method of claim 1, wherein said step of executing said first set of instructions further comprises the steps of performing **packed operations.**	**Packed operations** are operations on packed data, a data format in which the bits used to represent a single value are broken into a number of fixed sized data elements, each of which represents a separate value.
CLAIM 12	
The method of claim 1, wherein said step of executing said first set of instructions further comprises the steps of performing **packed integer operations.**	**Packed operations** are operations on packed data, a data format in which the bits used to represent a single value are broken into a number of fixed sized data elements, each of which represents a separate value.
CLAIM 13	
The method of claim 1, wherein said step of executing said first set of instructions further comprises the steps of performing **packed floating point operations.**	**Packed operations** are operations on packed data, a data format in which the bits used to represent a single value are broken into a number of fixed sized data

	elements, each of which represents a separate value.
CLAIM 14	
The method of claim 1, wherein said second **instruction type** are instructions that cause floating point operations to be performed.	An **instruction type** is a set of instructions with a common data format, register addressing format or other attribute assigned to a group of instructions based on an intrinsic feature of the instruction.
CLAIM 15	
The method of claim 1, wherein said second **instruction type** are instructions that cause scalar operations to be performed.	An **instruction type** is a set of instructions with a common data format, register addressing format or other attribute assigned to a group of instructions based on an intrinsic feature of the instruction.
CLAIM 17	
In a data processing apparatus, a method for executing instructions comprising the steps of:	A **register** is a storage area in a microprocessor configured to temporarily store data.
Executing a set of packed instructions and a set of floating point instructions on the contents of a **single logical register file**	A **register file** is a set of registers along with read and write circuitry.
	A **physical register file** is a register file that is physically implemented as circuitry in the microprocessor.
	A **logical register file** is a register file that is seen from the user's/programmer's/operating system's perspective.
that is **at least partially aliased,**	**Aliased** means that two or more different types of instructions appear to software to operate on a single logical register file.
Wherein said set of packed data instructions is executed prior to said set of floating point instructions, wherein a plurality of tags correspond to said single logical register file; and	**Partially alias** means that during a transition, only those registers implementing a single logical register file that

	contain useful data are copied from one physical register file to another.
Altering at least those of said plurality of tags corresponding to aliased registers in said single logical register file to a non-empty state sometime between attempting to execute the first of said set of packed data instructions and completing execution of the first of said set of floating point instructions, and wherein said plurality of tags identify whether registers in said single logical register file are empty or non-empty.	No construction necessary.
CLAIM 18	
The method of claim 17, wherein said step of executing comprises the steps of: executing said set of packed data instructions on said **single logical register file** as a fixed **register file;** and executing said set of floating point instructions on said single logical register file as a stack referenced register file.	A **register** is a storage area in a microprocessor configured to temporarily store data. A **register file** is a set of registers along with read and write circuitry. A **physical register file** is a register file that is physically implemented as circuitry in the microprocessor.
	A **logical register file** is a register file that is seen from the user's/programmer's/operating system's perspective.
CLAIM 19	
The method of claim 17, wherein said step of executing comprises the steps of:	**Packed operations** are operations on packed data, a data format in which the bits used to represent a single value are broken into a number of fixed sized data elements, each of which represents a separate value.
executing said set of packed data instructions to perform **packed integer operations;** executing said set of floating point instructions to perform scalar floating point operations.	
CLAIM 20	
The method of claim 17, wherein said step of executing comprises the steps of: executing said set of packed data instructions to perform **packed floating point operations;** executing said set of floating point instructions to perform scalar floating point operations.	**Packed operations** are operations on packed data, a data format in which the bits used to represent a single value are broken into a number of fixed sized data

	elements, each of which represents a separate value.
CLAIM 21	
In a data processing apparatus, a method for implementing partial context switching when executing scalar and packed data instructions comprising the steps of:	
receiving an instruction belonging to a first routine that is either one of said scalar or said packed data instructions;	
determining that a **single logical register file** for executing both said scalar and packed data instructions is unavailable due to a partial context switch; and	A **register** is a storage area in a microprocessor configured to temporarily store data. A **register file** is a set of registers along with read and write circuitry. A **physical register file** is a register file that is physically implemented as circuitry in the microprocessor.
	A **logical register file** is a register file that is seen from the user's/programmer's/operating system's perspective.
if said single logical register file is unavailable, then performing the steps of: Interrupting execution of said first routine; and executing a second routine to copy the contents of said single logical register file into a memory;	
Otherwise, executing said instruction on said single logical register file.	
CLAIM 22	
The method of claim 21 further comprising the steps of:	**A scalar emulation indication** is an indicator of floating point unit availability.
Determining if a **scalar emulation indication** is in an emulation state;	
if said scalar emulation indication is in said emulation state, then performing the steps of: Interrupting execution of said first routine; if said instruction is one of said scalar instructions, executing said second routine; and	
Otherwise, said instruction is one of said packed data instructions and executing a third routine rather than said second routine.	
CLAIM 23	
The method of claim 21, wherein said step of executing said instruction further includes the steps of:	A **register** is a storage area in a microprocessor configured to temporarily store data.

If said instruction is one of said packed data instructions, then performing the steps of: determining if said instruction causes a packed data item to be written to said **single logical register file;**

If said instruction causes said packed data item to be written to said single logical register file, then performing steps of:

writing said packed data item in a mantissa field of a **register** in said single logical register file; and

writing a value representing not a number or infinity in a sign field and an exponent field of said register.

A **register file** is a set of registers along with read and write circuitry.

A **physical register file** is a register file that is physically implemented as circuitry in the microprocessor.

A **logical register file** is a register file that is seen from the user's/programmer's/operating system's perspective.

CLAIM 24

The method of claim 21, wherein said step of executing said instruction further includes the steps of:

If said instruction is a **transition instruction of said packed data instructions,**

Then **altering** each tag in **a set of tags** to an empty state, wherein each tag in said set of tags corresponds to a different register in said single logical register file and identifies whether said corresponding register is empty or non-empty if said instruction is not said transition instruction but is one of said packed data instructions, then altering said set of tags to a non-empty state.

Transition instruction of said packed data instructions means an instruction used to indicate the end of a block of one or more packed data instructions.

A **register** is a storage area in a microprocessor configured to temporarily store data.

A **register file** is a set of registers along with read and write circuitry.

A **physical register file** is a register file that is physically implemented as circuitry in the microprocessor.

A **logical register file** is a register file that is seen from the user's/programmer's/operating system's perspective.

CLAIM 26

The method of claim 21, wherein said step of executing said instruction further includes the steps of:

A register is a storage area in a microprocessor configured to temporarily

	store data.
if said instruction if one of said packed data instructions, then **altering a set of tags** to a non-empty state if one of said scalar instructions was executed more recently than one of said packed data instructions, **wherein each tag in said set of tags corresponds to a different register** in said **single logical register file** and identifies whether said corresponding register is empty or non-empty; and	A **register file** is a set of registers along with read and write circuitry. A **physical register file** is a register file that is physically implemented as circuitry in the microprocessor.
	A **logical register file** is a register file that is seen from the user's/programmer's/operating system's perspective.
if said instruction is a **transition instruction of said packed data instructions,** then altering said set of tags to a empty state .	**Transition instruction of said packed data instructions** means an instruction used to indicate the end of a block of one or more packed data instructions.
CLAIM 30	
The method of claim 21, wherein said scalar instructions are for performing scalar floating point operations and said packed data instructions are for performing **packed integer operations.**	**Packed operations** are operations on packed data, a data format in which the bits used to represent a single value are broken into a number of fixed sized data elements, each of which represents a separate value.
CLAIM 31	
The method of claim 21, wherein said scalar instructions are for performing scalar floating point operations and said packed data instructions are for performing **packed floating point operations.**	**Packed operations** are operations on packed data, a data format in which the bits used to represent a single value are broken into a number of fixed sized data elements, each of which represents a separate value.
CLAIM 32	
In a data processing apparatus, a method for executing scalar and packed data instructions comprising the steps of: Receiving an instruction that is either a packed data instruction or a scalar instruction; **Determining that execution of scalar instructions should be emulated** and/or that a **single**	**A register** is a storage area in a microprocessor configured to temporarily store data.

logical register file for executing both said scalar and packed data instructions is unavailable due to a partial context switch; if said instruction is said scalar instruction, then executing a first routine; and if said instruction is said packed data instruction, then performing steps of: if said single logical register file is unavailable, then executing said first routine, and if execution of said scalar instructions should be emulated, then executing a second routine.

A **register file** is a set of registers along with read and write circuitry.

A **physical register file** is a register file that is physically implemented as circuitry in the microprocessor.

A **logical register file** is a register file that is seen from the user's/programmer's/operating system's perspective.

	Determining that execution of scalar instructions should be emulated: No construction necessary.
CLAIM 33	
The method of claim 32, wherein said scalar instructions are for performing scalar floating point operations and said packed data instructions are for performing **packed integer operations.**	**Packed operations** are operations on packed data, a data format in which the bits used to represent a single value are broken into a number of fixed sized data elements, each of which represents a separate value.
CLAIM 34	
The method of claim 32, wherein said scalar instructions are for performing scalar floating point operations and said packed data instructions are for performing **packed floating point operations.**	**Packed operations** are operations on packed data, a data format in which the bits used to represent a single value are broken into a number of fixed sized data elements, each of which represents a separate value.
CLAIM 35	
In a data processing apparatus, a method for executing packed data instructions comprising the steps of:	**A register** is a storage area in a microprocessor configured to temporarily store data.

receiving a packed data instruction that causes a packed data item to be written to **what at least logically appears to software as a register in a logical register file that is also used for saving floating point data;** writing said packed data item in a mantissa field of said logical register file; and writing a value representing not a number or infinity in a sign field and an exponent field of said logical register.

A **register file** is a set of registers along with read and write circuitry.

A **physical register file** is a register file that is physically implemented as circuitry in the microprocessor.

A **logical register file** is a register file that is seen from the user's/programmer's/operating system's perspective.

UNITED STATES PATENT NO. 5,835,748

Actual Claim Language	Court's Claim Construction
CLAIM 1	
A processor comprising:	**A register** is a storage area in a microprocessor configured to temporarily store data.
a first **physical register file** for executing scalar instructions;	A **register file** is a set of registers along with read and write circuitry.
a second physical register file for executing packed data instructions;	A **physical register file** is a register file that is physically implemented as circuitry in the microprocessor.
	A **logical register file** is a register file that is seen from the user's/programmer's/operating system's perspective.
a **transition unit**	A **transition unit** is firmware, microcode and/or circuitry that causes the transition between floating point mode and packed data mode.
Configured to cause said first physical register file and said second physical register file to logically appear to software executing on said processor as a **single logical register file;**	A **register** is a storage area in a microprocessor configured to temporarily store data. A **register file** is a set of registers along with read and write circuitry. A **physical register file** is a register file that is physically implemented as circuitry in the microprocessor.
	A **logical register file** is a register file that is seen from the user's/programmer's/operating system's perspective.
an internal write-back cache coupled to the control unit;	
a selective re-initialization microcode program including microcode instructions to selectively re-initialize registers	

selected to place the microprocessor in its initial mode of operation while maintaining the validity of the contents of the internal write-back cache;	
transition means for unconditionally transitioning from the second mode to the initial mode while maintaining the validity of the contents of the internal write-back cache, said transition means including	**Transition means for unconditionally transitioning from the second mode to the initial mode while maintaining the validity of the contents of the internal write-back cache** is a means-plus-function term: *Function:* Unconditionally transitioning
	Possible Structure: Figure 2, INIT pin 14, Control Unit 16, "program for converting from second mode to initial mode".
an external pin coupled to the control unit to assert a high priority interrupt, and	**Coupled to** means connected directly or indirectly via hardware.
means, responsive to said high priority interrupt, **for halting operation of the microprocessor** and executing said selective re-initialization microcode; and	**Means ... for halting operation of the microprocessor** is a means-plus-function term: *Function:* Halting the microprocessor.
	Possible Structure: Circuitry within control unit 16.
reset means including a reset pin coupled to the internal write-back cache and the plurality of registers for resetting the microprocessor by resetting the contents of said registers and invalidating the contents of the internal write-back cache.	**Coupled to** means connected directly or indirectly via hardware. **Reset means** ... is a means-plus-function term: *Function:* Resetting the microprocessor.
	Possible Structure: Figure 2, RESET pin, and control unit 16.
CLAIM 9	
The apparatus of claim 7 wherein the **reset pin** is **coupled to** the control unit, and said microcode memory further comprises an initialization microcode program, responsive to assertion of the reset pin, for initializing the microprocessor to the initial mode.	A **reset pin** is a pin that is connected via hardware to a plurality of registers and the cache. **Coupled to** means connected directly or indirectly via hardware.

Actual Claim Language	Court's Claim Construction
a **stack reference unit,**	**Stack reference unit** means firmware, microcode and/or circuitry that allows the registers to be addressed relative to a top of stack

	value.
coupled to said first register file, configured to operate said first physical register file as a stack, said stack reference unit including a set of tags, each tags of said set of tags corresponding to a different register in said first physical register file and identifying whether said corresponding register is in either a empty state or a non-empty state; and	
an **fixed register file unit,**	A **fixed register file unit** is firmware, microcode and/or circuitry that allows the registers to be addressed as a flat file.
coupled to said second physical register file, configured to operate said second physical register file as a fixed register file.	
CLAIM 4	
The processor of claim 1, wherein: said transition unit is configured to cause each tag in said set of tags to be altered to said non-empty state sometime in a first interval of time between the start of executing a set of packed data instructions and the start of executing a set of scalar instructions if a set of **transition instructions** is not executed sometime in a second interval of time after the execution of said set of packed data instructions and before the execution of said set of scalar instructions.	**Transition instructions** are one or more instructions used to indicate the end of a block of one or more packed data instructions.
CLAIM 6	
The processor of claim 1, wherein: each **register** in said first **physical register file** corresponds to a different **register** in said second **physical register file;** and said **transition unit** is also configured to store, sometime between the start of executing a set of packed data instructions and the start of executing a set of scalar instructions, a value indicating either not a number or infinity in a sign and exponent field of each register in said first physical register file whose corresponding register in said second physical register file was written to during the execution of said set of packed data instructions.	**A register** is a storage area in a microprocessor configured to temporarily store data. A **register file** is a set of registers along with read and write circuitry. A **physical register file** is a register file that is physically implemented as circuitry in the microprocessor. A **logical register file** is a register file that is seen from the user's/programmer's/operating system's perspective.
	A **transition unit** is firmware, microcode and/or circuitry that causes the

	transition between floating point mode and packed data mode.
CLAIM 7	
The processor of claim 1, wherein said first and second **physical register files** each contain n **registers,** and said **single logical register file** contains n registers.	A **register** is a storage area in a microprocessor configured to temporarily store data. A **register file** is a set of registers along with read and write circuitry. A **physical register file** is a register file that is physically implemented as circuitry in the microprocessor.
	A **logical register file** is a register file that is seen from the user's/programmer's/operating system's perspective.
CLAIM 11	
The processor of claim 1, wherein said packed data instructions cause said processor to perform **packed floating point operations.**	**Packed floating point operations:** No construction necessary.
CLAIM 12	
The processor of claim 1, wherein said packed data instructions cause said processor to perform **packed integer operations.**	**Packed integer operations:** No construction necessary.
CLAIM 26	
A processor comprising: a first set of physical **registers;**	A **register** is a storage area in a microprocessor configured to temporarily store data.
a second set of physical **registers;**	
an **execution unit** coupled to said first set of physical registers **to perform floating point operations on data stored in said first set of physical registers, and** coupled to said second set of physical registers **to perform integer operations on data stored in said second set of physical registers;**	**Execution unit** ... means firmware, microcode and/or circuitry within the processor that performs floating point and integer operations on data.
a **transition unit to at least partially alias** said first set of physical registers onto said second set of physical registers;	A **transition unit** is firmware, microcode and/or circuitry that causes the transition between floating point mode and packed data mode.

	Aliased means that two or more different types of instructions appear to software to operate on a single logical register file.
	Partially alias means that during a transition, only those registers implementing a single logical register file that contain useful data are copied from one physical register file to another.
a **stack reference unit** coupled to said first set of physical registers, said stack reference unit including a first storage area having stored therein a top of stack indication identifying one register in said first set of physical registers; and	**Stack reference unit** means firmware, microcode and/or circuitry that allows the registers to be addressed relative to a top of stack value.
a **non-stack reference unit** coupled to said second set of physical registers.	A **non-stack reference unit** is firmware, microcode and/or circuitry that allows the registers to be addressed as a flat file.
CLAIM 27	
The processor of claim 26, wherein said **transition unit** is microcode.	**A transition unit** is firmware, microcode and/or circuitry that causes the transition between floating point mode and packed data mode.
CLAIM 28	
The processor of claim 26, wherein said first set of physical registers and said second set of physical registers **logically appear as a single set of logical registers to software executing on said processor.**	A **register** is a storage area in a microprocessor configured to temporarily store data. A **register file** is a set of registers along with read and write circuitry. A **physical register file** is a register file that is physically implemented as circuitry in the microprocessor.
	A **logical register file** is a register file that is seen from the user's/programmer's/operating

	system's perspective.
CLAIM 29	
The processor of claim 26, further comprising:	
a **second storage area, coupled to said execution unit, having stored therein a mode indication** identifying either a floating point mode or a integer mode, said transition unit altering said mode indication to identify said floating point mode and copying data from said second set of physical registers into said first set of physical registers in response to receiving one of a set of floating point instructions when said mode indication identifies said integer mode, said transition unit altering said mode indication to identify said integer mode and copying data from said first set of physical registers into said second set of physical registers in response to receiving one of a set of integer instructions when said mode indication identifies said floating point mode.	**Second storage area ...:** No construction necessary.
CLAIM 30	
The processor of claim 29, further comprising:	A **register** is a storage area in a microprocessor configured to temporarily store data.
said **second storage area** also having stored therein a set of dirty indications in one of a dirty state and a clean state, each dirty indication in said set of dirty indications corresponding to a different **register** in said second set of physical registers;	A **transition unit** is firmware, microcode and/or circuitry that causes the transition between floating point mode and packed data mode.
said execution unit, in response to writing data to a selected register in said second set of physical registers, also altering the dirty indication corresponding to said selected register to said dirty state, and said **transition unit,** in response to receiving one of a set of floating point instructions while said mode indication identifies said integer mode, also writing a value indicating not a number or infinity in a sign and exponent field of each register in said first set of physical registers that corresponds to a register in said second set of physical registers whose corresponding dirty indication is in said dirty state.	
CLAIM 34	
The processor in claim 26, further comprising:	**Second storage area ...:** No construction necessary.
a **second storage area having stored therein a mode indication** identifying either a floating point mode or a integer mode, and a speculative indication identifying either a speculative state or a non-speculative state;	
said **transition unit** altering said mode indication to identify said integer mode, copying data from said first set of physical **registers** into said second set of physical registers, and altering said speculative indication to identify said speculative state in response to receiving one of a set of integer instructions when said mode indication identifies said floating point mode;	A **register** is a storage area in a microprocessor configured to temporarily store data.

	A **transition unit** is firmware, microcode and/or circuitry that causes the transition between floating point mode and packed data mode.
said **execution unit** coupled to receive said mode indication and said speculative indication, said execution unit altering said speculative indication to identify said nonspeculative state in response to executing one of said set of integer instructions;	**Execution unit** means firmware, microcode and/or circuitry within the processor that performs floating point and integer operations on data.
said **transition unit** altering said mode indication to identify said floating point mode in response to receiving one of a set of floating point instructions when said mode indication identifies said integer mode; and said **transition unit** altering said mode indication to identify said floating point mode in response to receiving one of a set of floating point instructions when said mode indication identifies said integer mode; and said transition unit also copying data from said second set of physical **registers** into said first set of physical registers in response to receiving one of said set of floating point instructions when said mode indication identifies said integer mode and when said speculative indication identifies said non-speculative state.	A **transition unit** is firmware, microcode and/or circuitry that causes the transition between floating point mode and packed data mode.
	A **register** is a storage area in a microprocessor configured to temporarily store data.
CLAIM 38	
The processor of claim 26, wherein said integer operations are **packed integer operations.**	**Packed integer operations:** No construction necessary.
CLAIM 40	
A processor comprising:	A **register** is a storage area in a microprocessor configured to temporarily store data.
a first plurality of physical **registers** for executing scalar instructions;	
a second plurality of physical registers for executing packed data instructions; and	
a **transition means coupled to said first and second plurality of physical registers for causing said first and second plurality of physical registers to appear to software executing on said processor as a single logical register file,** and for causing a value indicating not a number or infinity to be written in a sign and exponent field of said first plurality of physical **registers** sometime in an interval of time between the start of executing a set of packed data instructions and the start of executing a set of said scalar instructions.	**Transition means ...** is a means-plus-function term: *Function:* Causing the transition between floating

	transition between floating point mode and packed data mode *Structure:* Transition Unit 600, Mode Indication 675, steps 804, 812, 908 and 914.
	A **register** is a storage area in a microprocessor configured to temporarily store data.
CLAIM 41	
The processor of claim 40, wherein said transition means **at least partially aliases** said second plurality of physical registers on said first plurality of physical registers.	**Aliased** means that two or more different types of instructions appear to software to operate on a single logical register file. **Partially alias** means that during a transition, only those registers implementing a single logical register file that contain useful data are copied from one physical register file to another.
	A **register** is a storage area in a microprocessor configured to temporarily store data.
CLAIM 44	
The processor of claim 40, further comprising:	**Means for operating said first plurality of physical registers as a stack** is a means-plus-function term:
a first **means for operating said first plurality of physical registers as a stack;** and	The **function** is operating the register file as a stack (i.e., addressing relative to a top of stack value).
	The **structure** is the circuitry 630 shown in figure 6B.
a second **means for operating said second plurality of physical registers as a fixed register file.**	**Means for operating said second plurality of physical registers as a fixed register file** is a means-plus-function term:
	The **function** is to operate as a fixed (or directly addressed) register file. The **structure** is the packed data nonstack reference unit 660 of figure 6A, described in the

	specification at col. 24:65-col. 25:2 of the '748 patent.
CLAIM 48	
The processor of claim 40, wherein said packed data instructions cause said processor to perform **packed floating point operations.**	**Packed floating point operations:** No construction necessary.
CLAIM 58	
In a data processing apparatus, a method for executing instructions comprising the steps of:	A **register** is a storage area in a microprocessor configured to temporarily store data.
executing a set of scalar instructions on a first physical register file, operating said first physical register file as a stack;	A **register file** is a set of registers along with read and write circuitry.
executing a set of packed data instructions on a second physical register file; and **causing said first physical register file and said second physical register file to appear to software as a single logical register file.**	A **physical register file** is a register file that is physically implemented as circuitry in the microprocessor.
A **logical register file** is a register file that is seen from the user's/programmer's/operating system's perspective.	
CLAIM 61	
The method of claim 58, wherein said step of executing said set of scalar instructions further comprised the steps of:	**A register** is a storage area in a microprocessor configured to temporarily store data.
Determining if said **single logical register file** is unavailable due to a partial context switch;	A **register file** is a set of registers along with read and write circuitry.
if said single logical register file is unavailable, then performing the steps of:	A **physical register file** is a register file that is physically implemented as circuitry in the microprocessor.
executing a routine to store in a memory data stored in said logical register file prior to executing said set of scalar instructions.	A **logical register file** is a register file that is seen from the user's/programmer's/operating system's perspective.
CLAIM 62	
In a data processing apparatus, a method for executing instructions comprising the steps of:	**Determining that said first instruction is either a floating point instruction or a packed data instruction:** No construction necessary.
receiving a first instruction;	**Transitioning to said floating point mode** is changing to the mode in

	which floating point instructions execute.
Determining that said first instruction is either a floating point instruction or a packed data instruction; Determining if a processor containing a first and second set of physical registers is in either a floating point mode or a packed data mode;	
if said first instruction is said floating point instruction, then **transitioning to said floating point mode** if said processor is in said packed data mode, and executing said floating point instruction using said first set of physical registers; otherwise,	
Transitioning to said packed data mode if said processor is in said floating point mode, and executing said packed data instruction using said second set of physical registers that is **at least partially aliased** on said first set of physical registers such that said first set of physical registers and said second set of physical registers logically appears to software as a **single logical register file.**	**Transitioning to said packed data mode** is switching to the mode in which packed data instructions execute. **Aliased** means that two or more different types of instructions appear to software to operate on a single logical register file. **Partially alias** means that during a transition, only those registers implementing a single logical register file that contain useful data are copied from one physical register file to another. A **register** is a storage area in a microprocessor configured to temporarily store data. A **register file** is a set of registers along with read and write circuitry. A **physical register file** is a register file that is physically implemented as circuitry in the microprocessor.
	A **logical register file** is a register file that is seen from the user's/programmer's/operating system's perspective.
CLAIM 63	
The method of claim 62, wherein said step of executing said floating point	A **register** is a storage area in

instruction using said first set of physical **registers** and said step of executing said packed data instruction using said second set of physical registers aliased on said first set of physical registers both further comprise the steps of:	a microprocessor configured to temporarily store data.
determining if said **single logical register file** is unavailable due to a partial context switch; if said single logical register file is unavailable, then performing the steps of: interrupting execution of said first instruction; executing a second routine to store in a memory data stored in said logical register file; **restarting execution** of said first instruction.	A **register** is a storage area in a microprocessor configured to temporarily store data. A **register file** is a set of registers along with read and write circuitry. A **physical register file** is a register file that is physically implemented as circuitry in the microprocessor. A **logical register file** is a register file that is seen from the user's/programmer's/operating system's perspective. **Restarting execution:** No construction necessary.
CLAIM 65	
The method of claim 62, wherein said step of **transitioning to said floating point mode** includes the step of copying data stored in said second set of physical **registers** into said first set of physical registers, and wherein said step of **transitioning to said packed data mode** includes the step of copying data stored in said first set of physical registers into said second set of physical registers.	**Transitioning to said floating point mode** is changing to the mode in which floating point instructions execute. A **register** is a storage area in a microprocessor configured to temporarily store data. **Transitioning to said packed data mode** is switching to the mode in which packed data instructions execute.
CLAIM 66	
The method of claim 62 wherein said step of transitioning to said floating point mode is performed by microcode and **execution is resumed without executing any non-microcode instructions.**	**Execution is resumed without executing any non-microcode instructions:** No construction necessary.
CLAIM 67	
The method of claim 62 wherein said step of **transitioning to said packed data mode** is performed by microcode and **execution is resumed without executing any non-microcode instructions.**	**Transitioning to said packed data mode** is switching to the mode in

	which packed data instructions execute.
	Execution is resumed without executing any non-microcode instructions: No construction necessary.
CLAIM 68	
The method of claim 62, wherein said step of **transitioning to said floating point mode** further includes the step of:	**Transitioning to said floating point mode** is changing to the mode in which floating point instructions execute.
setting a top of stack indication to an initialization value, said data processing apparatus operating said first set of physical **registers** as a stack and said top of stack indication identifying which of said first set of physical **registers** is currently on top of said stack.	A **register** is a storage area in a microprocessor configured to temporarily store data.
CLAIM 69	
The method of claim 62 wherein: said step of executing said packed data instruction further includes the steps of: Determining if said packed data instruction is a **transition instruction;** if said packed data instruction is said transition instruction, then altering a last instruction indication to indicate the last packed data instruction executed was said transition instruction;	**Transition instruction** is one or more instructions used to indicate the end of a block of one or more packed data instructions.
otherwise, then altering said last instruction indication to indicate the last packed data instruction executed was not said transition instruction; and wherein said step of **transitioning to said floating point mode** further includes the step of:	**Transitioning to said floating point mode** is changing to the mode in which floating point instructions execute.
determining whether said last instruction indication indicates the last packed data instruction executed was the transition instruction;	
if said last instruction indication indicates the last packed data instruction executed was the transition instruction, then altering each of a set of tags to an empty state, each register in said first set of physical registers corresponding to a different one of said set of tags; otherwise, altering each of said set of tags to a non-empty state.	A **register** is a storage area in a microprocessor configured to temporarily store data.
CLAIM 70	
The method of claim 62, wherein: said step of **transitioning to said packed data mode** further includes the step of:	**Transitioning to said packed data mode** is switching to the mode in which packed data instructions execute.
altering each dirty indication in a set of dirty indications to indicate a clean state, each dirty indication in said set of dirty indications corresponding to a different one of said second set of physical	**registers;** and A **register** is a storage area in a microprocessor configured to temporarily store data.
wherein said step of executing said packed data instruction further includes the steps of:	

determining if execution of said packed data instruction causes data to be written to one or more of said second set of physical registers; and	
if said packed data instruction causes said processor to write to one or more of said second set of physical registers, then altering, to a dirty state, those of said set of dirty indications that correspond to those registers in said second set of physical registers to which data is written to;	
wherein said step of **transitioning to said floating point mode** further includes the step of:	**Transitioning to said floating point mode** is changing to the mode in which floating point instructions execute.
identifying a subset of said second set of physical **registers,** said subset including those of said second set of physical registers whose corresponding dirty indication is in said dirty state as dirty registers; and altering a sign and an exponent field of each register in said subset to indicate not a number of infinity.	A **register** is a storage area in a microprocessor configured to temporarily store data.
CLAIM 71	
The method of claim 62, wherein said first set of physical **registers** is operated as a stack and said second set of physical registers is operated as a fixed **register file.**	A **register** is a storage area in a microprocessor configured to temporarily store data.
	A **register file** is a set of registers along with read and write circuitry.
CLAIM 76	
The apparatus of claim 1, wherein said **transition microcode program** selectively re-initializes registers including a CRO register, a EFLAGS register, an instruction pointer register, segment registers, interrupt descriptor table registers, and a debug control register.	**Transition microcode program** means a firmware and/or microcode that causes the transition of the processor from the second mode of operation to the initial mode of operation.
CLAIM 5	
The apparatus of claim 1 wherein the **reset pin** is **coupled to** the control unit, and said microcode memory further comprises an initialization microcode program, responsive to assertion of the reset pin, for initializing the microprocessor to the initial mode.	A reset **pin** is a pin that is connected via hardware to a plurality of registers and the cache.
	Coupled to means connected directly or indirectly via hardware.
CLAIM 7	
In a microprocessor having at least two modes of operation including an initial mode and a second mode, said initial mode being the mode to which the microprocessor is initialized upon start-up, an apparatus for transitioning from the second mode to the initial mode including reinitializing the contents of selected registers, said apparatus comprising:	A **control unit** is circuitry that receives and executes microcode instructions and receives an interrupt signal.

a microcode memory means for storing microcode; **a control unit**	
coupled to said microcode memory means for executing microcode stored in said microcode memory means;	**Coupled to** means connected directly or indirectly via hardware.
a plurality of registers coupled to the control unit; said registers **having an initial state that corresponds to the initial mode;**	**Coupled to** means connected directly or indirectly via hardware.
A processor comprising:	**A register** is a storage area in a microprocessor configured to temporarily store data.
a first physical register file for executing scalar instructions;	A **register file** is a set of registers along with read and write circuitry.
a second physical register file for executing packed data instructions, wherein each register in said first physical register file corresponds to a different register in said second physical register file; and	A **physical register file** is a register file that is physically implemented as circuitry in the microprocessor.
a **transition unit** configured to cause said first physical register file and said second physical register file to logically appear to software executing on said processor as a **single logical register file,** and to store, sometime between the start of executing a set of packed data instructions and the start of executing a set of scalar instructions, a value indicating either not a number or infinity in a sign and exponent field of each register in said first physical register file whose corresponding register in said second physical register file was written to during the execution of said set of packed data instructions.	A **logical register file** is a register file that is seen from the user's/programmer's/operating system's perspective.
CLAIM 77	
The processor of claim 76, further comprising:	**Stack reference unit** means firmware, microcode and/or circuitry that allows the registers to be addressed relative to a top of stack value.
a **stack reference unit,** coupled to said first physical register file, configured to operate said first physical register file as a stack; and	
an **fixed register file unit,** coupled to said second physical register file, configured to operate said second physical register file as a fixed register file.	A **fixed register file unit** is firmware, microcode and/or circuitry that allows the registers to be addressed as a flat file.
CLAIM 87	
The processor of claim 76, wherein said packed data instructions cause said processor to perform **packed floating point operations.**	**Packed floating point operations:** No construction necessary.
CLAIM 88	
A processor comprising: a first **physical register file;**	**A register** is a storage area

	in a microprocessor configured to temporarily store data. A **register file** is a set of registers along with read and write circuitry. A **physical register file** is a register file that is physically implemented as circuitry in the microprocessor.
	A **logical register file** is a register file that is seen from the user's/programmer's/operating system's perspective.
a **stack reference unit,** coupled to said first physical register file, configured to operate said first physical register file as a stack;	**Stack reference unit** means firmware, microcode and/or circuitry that allows the registers to be addressed relative to a top of stack value.
a second physical register file; a **fixed register file unit,** coupled to said second physical register file, configured to operate said second physical register file as a fixed register file;	A **fixed register file unit** is firmware, microcode and/or circuitry that allows the registers to be addressed as a flat file.
a **transition unit** configured to cause said first and second physical register file to logically appear to software executing on said processor as a **single logical register file,** said processor configured to store a plurality of data types in said single logical register file, said plurality of data types including scalar floating point and packed integer data.	

UNITED STATES PATENT NO. 5,889,679

Actual Claim Language	Court's Claim Construction
CLAIM 1	
A microprocessor, having a plurality of **functional blocks,** the plurality of functional blocks being **individually enabled/disabled,** the microprocessor comprising:	**Functional blocks** are a collection of circuits and possibly microcode in a microprocessor, dedicated to performing a specific function. Each block may be individually turned on or off.
	Individually enabled/disabled: No construction necessary.
a **fuse array control,** coupled to the plurality of functional blocks, said fuse array control selectively signaling the plurality of functional blocks to be enabled/disabled,	A **fuse array control** is a collection of circuitry and possibly microcode, including a fuse array, that sends a signal, based on the

	fuses and inputs from the control unit, to each functional block that turns the functional block on or off.
said fuse array control having a **fuse array;** and	A fuse is a circuit element that provides an electrical connection between two points, but which may be irreversibly destroyed thereby electrically disconnecting the two points.
a **control unit,** coupled to said fuse array control, for reading said fuse array, and indicating to said fuse array control which of the plurality of functional blocks should be enabled/disabled.	A **fuse array** is a group of fuses.
	A **control unit** is circuitry, and possibly microcode, that issues control signals.
CLAIM 4	
The microprocessor, as recited in claim 1, wherein **said fuse array control** further comprises	A **fuse array control** is a collection of circuitry and possibly microcode, including a fuse array, that sends a signal, based on the fuses and inputs from the control unit, to each functional block that turns the functional block on or off.
a plurality of **enable/disable signals.**	**Enable/Disable Signals** are signals sent to the functional blocks that indicate whether the functional block should be turned on or off.
CLAIM 6	
The microprocessor, as recited in claim 4, wherein said fuse array control selectively signals the plurality of functional blocks to be enabled/disabled via said plurality of **enable/disable signals.**	**Enable/Disable Signals** are signals sent to the functional blocks that indicate whether the functional block should be turned on or off.
CLAIM 11	
The microprocessor, as recited in Claim 10, wherein said open/closed states of said ones of said plurality of fuses is **partly determinative** of which of the functional blocks should be enabled/disabled.	**Partly determinative** means the state of the fuses in the fuse array does not by itself determine whether a functional block is turned on or off.
CLAIM 12	
The microprocessor, as recited in claim 1, wherein said fuse array control further comprises **a feature control register,** coupled to said fuse array, into which at least a portion of the state of said fuse array is stored.	A **feature control register** is a register which holds at least one bit indicative of whether a functional block should be turned on or off.
	A **fuse** is a circuit element that provides an electrical connection between two points, but which may be irreversibly destroyed thereby electrically disconnecting the two points.
	A **fuse array** is a group of fuses.

CLAIM 13	
The microprocessor, as recited in claim 12, wherein the state of said **fuse array** is stored into said **feature control register** upon power up of the microprocessor.	A **feature control register** is a register which holds at least one bit indicative of whether a functional block should be turned on or off. A **fuse** is a circuit element that provides an electrical connection between two points, but which may be irreversibly destroyed thereby electrically disconnecting the two points.
	A **fuse array** is a group of fuses.
CLAIM 14	
The microprocessor, as recited in claim 13, wherein contents of said **feature control register** are indicative of which of the functional blocks should be enabled/disabled.	A **feature control register** is a register which holds at least one bit indicative of whether a functional block should be turned on or off.
CLAIM 16	
The microprocessor as recited in claim 12, wherein contents of said **feature control register** are read by said **control unit.**	**A feature control register** is a register which holds at least one bit indicative of whether a functional block should be turned on or off.
	A **control unit** is circuitry, and possibly microcode, that issues control signals.
CLAIM 17	
The microprocessor, as recited in claim 16, wherein said **control unit** overwrites said contents of said **feature control register.**	A **control unit** is circuitry, and possibly microcode, that issues control signals.
	A **feature control register** is a register which holds at least one bit indicative of whether a functional block should be turned on or off.
CLAIM 18	
The microprocessor, as recited in claim 17, wherein said **control unit** overwrites said contents of said **feature control register** to override enabling/disabling of ones of the functional blocks specified by the state of said **fuse array.**	A **control unit** is circuitry, and possibly microcode, that issues control signals. A **feature control register** is a register which holds at least one bit indicative of whether a functional block should be turned on or off. A **fuse** is a circuit element that provides an electrical connection between two points, but which may be irreversibly destroyed thereby electrically disconnecting the two points.

	A **fuse array** is a group of fuses.
CLAIM 19	
The microprocessor, as recited in claim 18, wherein if said **control unit** does not overwrite said contents of said **feature control register,** the plurality of functional blocks are enabled/disabled based on the state of said **fuse array.**	**A control unit** is circuitry, and possibly microcode, that issues control signals. A **feature control register** is a register which holds at least one bit indicative of whether a functional block should be turned on or off. A **fuse** is a circuit element that provides an electrical connection between two points, but which may be irreversibly destroyed thereby electrically disconnecting the two points.
	A **fuse array** is a group of fuses.
CLAIM 20	
The microprocessor, as recited in claim 19, wherein if said **control unit** does overwrite said contents of said **feature control register,** the plurality of functional blocks are enabled/disabled based on program instructions executed by the microprocessor.	A **control unit** is circuitry, and possibly microcode, that issues control signals.
	A **feature control register** is a register which holds at least one bit indicative of whether a functional block should be turned on or off.
CLAIM 21	
The microprocessor, as recited in claim 20, wherein said program instructions comprise microcode stored within said **control unit.**	A **control unit** is circuitry, and possibly microcode, that issues control signals.
CLAIM 23	
The microprocessor, as recited in claim 22, wherein said BIOS instructions perform a write to a machine specific register (MSR), which is said **feature control register.**	A **feature control register** is a register which holds at least one bit indicative of whether a functional block should be turned on or off.
CLAIM 24	
The microprocessor, as recited in claim 23, wherein said write to said MSR may be **partially or wholly blocked** by said **control unit.**	**Partially or wholly blocked:** No construction necessary.
	A **control unit** is circuitry, and possibly microcode, that issues control signals.
CLAIM 27	
A microprocessor, having a plurality of **functional blocks** that are **individually enabled/disabled,** the microprocessor comprising:	**Individually enabled/disabled:** No construction necessary.
	Functional blocks are a collection of

	circuits and possibly microcode in a microprocessor, dedicated to performing a specific function. Each block may be individually turned on or off.
a **fuse array,** fabricated on the die of the microprocessor, the fuse array comprising a plurality of **fuses** that may be individually blown during manufacturing;	A **fuse** is a circuit element that provides an electrical connection between two points, but which may be irreversibly destroyed thereby electrically disconnecting the two points.
	A **fuse array** is a group of fuses.
feature control register, coupled to said fuse array, said **feature control register** for selectively enabling/disabling ones of said plurality of functional blocks; and	A **feature control register** is a register which holds at least one bit indicative of whether a functional block should be turned on or off.
	Functional blocks are a collection of circuits and possibly microcode in a microprocessor, dedicated to performing a specific function. Each block may be individually turned on or off.
a **control unit,** coupled to said feature control register, for reading the state (closed or open) of said plurality of fuses, and for storing into said feature control register a value indicative of which of said plurality of functional blocks are to be enabled/disabled.	A **control unit** is circuitry, and possibly microcode, that issues control signals.
CLAIM 29	
The microprocessor, as recited in claim 27, wherein the plurality of functional blocks are individually enabled/disabled via **signal** lines that couple the plurality of functional blocks to said feature control register.	**Signal lines** are wires that carry signals.
	Enable/disable signal lines are wires that carry signals indicating whether a functional block is to be turned on or off.
CLAIM 30	
The microprocessor, as recited in claim 27, wherein said **fuse array** is coupled to said **control unit** to allow said control unit to read the state (closed or open) of each of said plurality of fuses.	**A fuse** is a circuit element that provides an electrical connection between two points, but which may be irreversibly destroyed thereby electrically disconnecting the two points. A **fuse array** is a group of fuses.
	A **control unit** is circuitry, and possibly microcode, that issues control signals.
CLAIM 35	
The microprocessor, as recited in claim 27, wherein said feature control register comprises **enable/disable signal lines,** coupled to each of the plurality of functional blocks, to selectively enable/disable ones of said plurality of	**Signal** lines are wires that carry signals.

functional blocks.

	Enable/disable signal lines are wires that carry signals indicating whether a functional block is to be turned on or off.
CLAIM 38	
The microprocessor, as recited in claim 27, wherein said **control unit** further comprises a control ROM that stores a default configuration for the plurality of functional blocks on the microprocessor.	A **control unit** is circuitry, and possibly microcode, that issues control signals.
CLAIM 40	
The microprocessor, as recited in claim 27, wherein the microprocessor further comprises a **processor instruction for writing a configuration command to said feature control register.**	A **processor instruction for writing a configuration command to said feature control register** is a "processor instruction" that writes values into the feature control register to specify which functional blocks are turned on and which are turned off.
CLAIM 41	
The microprocessor, as recited in claim 40, wherein said processor instruction overwrites said **feature control register** to selectively enable/disable particular ones of said plurality of functional blocks.	A **feature control register** is a register which holds at least one bit indicative of whether a functional block should be turned on or off.
CLAIM 42	
The microprocessor, as recited in claim 41, wherein said plurality of functional blocks are selectively enabled/disabled by either a default configuration, said state of said fuse **array,** or execution of said processor instruction.	**A fuse** is a circuit element that provides an electrical connection between two points, but which may be irreversibly destroyed thereby electrically disconnecting the two points.
	A fuse array is a group of fuses.
CLAIM 43	
The microprocessor, as recited in claim 40, wherein said control unit selectively **blocks portions** of said overwrites by said processor instruction to prevent certain ones of said plurality of functional blocks from being enabled/disabled by said processor instruction.	**Blocks portions:** No construction necessary.
CLAIM 45	
A method for selectively enabling/disabling **functional blocks** on a microprocessor, the microprocessor having a plurality of fuses, ones of which 4are blown during manufacturing, the method comprising the steps of:	**Functional blocks** are a collection of circuits and possibly microcode in a microprocessor, dedicated to performing a specific function. Each block may be individually turned on or off.
reading the state of the plurality of **fuses;**	A **fuse** is a circuit element that provides an electrical connection between two points, but which may be irreversibly destroyed thereby electrically disconnecting the two points.
determining from said step of reading which of the functional	

blocks are to be enabled/disabled;	
logically merging results from said step of determining with a predetermined configuration for the functional blocks; and enabling/disabling the functional blocks according to the result from said step of logically merging.	**Logically** merging means performing logical operations by combining one or more bits of a plurality of values.
CLAIM 46	
The method for selectively enabling/disabling functional blocks on a microprocessor, as recited in claim 45, wherein the method further comprises the step of: executing an instruction to overwrite a portion of the results from said step of **logically merging** to selectively enable/disable the functional blocks after the ones of the plurality of fuses are blown.	**Logically merging** means performing logical operations by combining one or more bits of a plurality of values.
CLAIM 47	
The method for selectively enabling/disabling functional blocks on a microprocessor, as recited in claim 46, wherein the method further comprises the step of: **blocking a portion** of the overwrite of said step of executing to prevent certain ones of the functional blocks from being enabled/disabled by said step of executing.	**Blocking a portion:** No construction necessary.

UNITED STATES PATENT NO. 6,385,735

Actual Claim Language	Court's Claim Construction
CLAIM 5	
A frequency **selecting** circuit comprising:	**Selecting:** No construction necessary.
one or more frequency selecting fusible elements which can be programmed to select a processor clock frequency;	A **processor clock frequency** is a clock rate for the processor expressed as a specific frequency or as a ratio of a base frequency, such as the frequency of an external clock.
	One or more frequency selecting fusible elements which can be programmed to select a processor clock frequency means one or more fuses that can be blown to select a clock rate for the processor expressed as a specific frequency, or as a ratio of a frequency.
frequency selection logic coupled to **the fusible elements** and receiving as an input a signal identifying a **selected** processor clock frequency, said **frequency selection logic selecting the clock frequency identified by the fusible elements if one or more of the fuses are programmed, and otherwise selecting the clock frequency identified by the received signal.**	A **fuse** is a circuit element that provides an electrical connection between two points, but which may be irreversibly destroyed thereby electrically disconnecting the two points. A **processor clock frequency** is a clock rate for the processor expressed as a specific frequency or as a ratio of a base frequency, such as the frequency of an external

	clock.
	Frequency selection logic selecting the clock frequency identified by the fusible elements if one or more of the fuses are programmed, and otherwise selecting the clock frequency identified by the received signal is circuitry that selects a processor clock frequency identified by the fuses if one or more of the fuses are blown, and otherwise selects the processor clock frequency identified by the received signals.
CLAIM 11	
A method of selecting a **processor clock frequency** comprising the steps of: **selectively programming fusible elements to encode a** first **processor clock frequency;** generating a signal selecting a second processor clock frequency;	**A processor clock frequency** is a clock rate for the processor expressed as a specific frequency or as a ratio of a base frequency, such as the frequency of an external clock.
	Selectively programming means choosing which, if any, of the fusible elements to blow. generating a signal selecting a second processor clock frequency;
selecting the first processor clock frequency if at least one of the fusible elements has been programmed, otherwise selecting the second processor clock frequency.	**Selecting the first processor clock frequency if at least one of the fusible elements has been programmed, otherwise selecting the second processor clock frequency** means selecting the first processor clock frequency if at least one of the fuses has been blown, otherwise selecting the second processor clock frequency.

UNITED STATES PATENT NO. 5,201,043

Actual Claim Language	Court's Claim Construction
CLAIM 8	
A microprocessor device for executing instructions, said device having privilege levels 0-3 within which said instructions execute, and having a **system for generating memory requests with alignment checking of said memory requests,** said device comprising: a flag register writable by an instruction executed at privilege level 3, said flag register including an alignment control bit for enabling generation of an alignment fault;	**System for generating memory requests with alignment checking of said memory requests** is a means-plus-function term: **Function:** (1) Generating memory requests-generating a signal on a bus connected to memory, which signal contains a memory address, control information (i.e. whether it is a read or write request and how many bits are desired), and if it is a write

	request, the data to be written; (2) Alignment checking-detecting misaligned data references (i.e. the data address is not a multiple of its length); (3) Receiving an enable signal. The enable signal is present when the alignment control bit and the masking bit are set, as specified in the claim; and (4) Generating an alignment fault if a misaligned memory request occurs and the enable signal is present-generating a signal indicating a fault if the memory request is misaligned and the enable signal is present. **Corresponding Structure:** Fig. **4** (segmentation unit 14, control unit 19); Fig. 5 (gates 46, 47, 48, 49, 63, 65 and 67-72).
a control register writable by an instruction executed at privilege level 0, but not writable by an instruction executed at privilege level 3, and control register including a masking bit; and logic circuitry coupled to receive the contents of both said flag register and said control register, wherein said logic circuitry generates an enable signal only if both said alignment control bit and said masking bit are set, said enable signal being coupled to said system for generating memory requests such that **said system generates an alignment fault if a misaligned memory request occurs and said enable signal is present.**	**A control register writable by an instruction executed at privilege level 0, but not writable by an instruction executed at privilege level 3:** No construction necessary.
CLAIM 9	
The microprocessor device according to claim 8 wherein said alignment control and masking bits are set when their value is 1.	

UNITED STATES PATENT NO. 5,555,423

Actual Claim Language	Court's Claim Construction
CLAIM 1	
In a microprocessor having at least two modes of operation including an initial mode and a second mode, said initial mode being the mode to which the microprocessor is initialized upon start-up, an apparatus for transitioning from the second mode to the initial mode, said apparatus comprising: a microcode memory means for storing microcode; **a control unit**	**A control unit** is circuitry that receives and executes microcode instructions and receives an interrupt signal.
coupled to the microcode memory means for receiving and executing microcode that is stored in said microcode memory means and that	**Coupled to** means connected directly or indirectly via hardware.

controls the microprocessor, said control unit also for receiving interrupt signals;	
a plurality of registers, coupled to the control unit, **for storing register data;**	A **plurality of registers:** No construction necessary.
an internal write-back cache, coupled to the control unit, for storing cache data; a **transition microcode program for unconditionally transitioning** from the second mode to the initial mode while maintaining the validity of the contents of the internal write-back cache **including means for re-initializing registers selected to place the microprocessor in its initial mode of operation;**	**Transition microcode program** means a firmware and/or microcode that causes the transition of the processor from the second mode of operation to the initial mode of operation. **Means for re-initializing registers selected to place the microprocessor in its initial mode of operation** is a means-plus-function term: *Function:* Re-initializing registers selected to place the microprocessor in its initial mode of operation.
	Possible Structure: Figure 2. "Program for concerting from second mode to initial mode"; Col. 4, ll. 35-44; Col. 6, ll. 13-15 and 35-44.
an external electrical pin connected to the microprocessor and coupled to the control unit but not to the internal write-back cache and the plurality of registers, that, when asserted, asserts a high priority interrupt to the control unit, and	
the control unit includes **means for halting the microprocessor** and unconditionally executing the transition microcode program when said interrupt is recognized; and	**Means for halting the microprocessor** is a means-plus-function term: *Function:* Halting the microprocessor.
	Possible Structure: Circuitry within control unit 16.
a reset pin coupled to the internal write-back cache and the plurality of registers for resetting the microprocessor by directly resetting the contents of said plurality of registers and invalidating the contents of the internal write-back cache.	A **reset pin** is a pin that is connected via hardware to a plurality of registers and the cache. **Directly resetting** means using hardware to reset the contents of the plurality of registers and invalidate the contents of the cache.
	Coupled to means connected directly or indirectly via hardware.

CLAIM 2	
The apparatus of claim 1, wherein the microprocessor further comprises floating point registers, and the transition microcode program further comprises **means for maintaining the contents of said floating point registers.**	**Means for maintaining the contents of said floating point registers** is a means-plus-function term: *Function:* Maintaining the contents of the floating point registers.
	Possible Structure: Figure 2, "program for converting from second mode to initial mode"; Col. 4, ll. 35-44; Col. 6, ll. 13-15 and 33-44.
CLAIM 3	
[missing text]	[missing text]