United States District Court,
N.D. New York.

CORNELL UNIVERSITY, a non-profit New York corporation, and Cornell Research Foundation, Inc., a nonprofit New York corporation,
Plaintiffs/counterclaim defendants.
v.
HEWLETT-PACKARD COMPANY, a Delaware corporation,
Defendant/counterclaimant.

March 26, 2004.

Background: Owner of patent for instruction-issuing mechanism for computer processors having multiple functional units sued competitor for infringement. Parties sought claim construction.

Holdings: The District Court, Mordue, J., held that:
(1) term "instruction" was not limited to specific format used in preferred embodiment;
(2) "concurrencies" were plurality of instructions that were ready to be executed at same time because they were dependency free;
(3) "dispatch stack" was enriched instruction buffer having additional fields holding values that tracked any dependencies associated with various instruction formats, and accompanying logic; and
(4) term "data dependencies" included both essential and false dependencies.

Claims construed.

See also 2003 WL 1524492.

4,807,115. Construed.

Sidley, Austin, Brown & Wood, LLP, Edward G. Poplawski, of counsel, Denise L. McKenzie, of counsel, Los Angeles, CA, Counsel for Plaintiffs/counterclaim defendants.

Gray, Cary, Ware & Freidenrich, LLP, Stewart M. Brown, of counsel, John E. Giust, of counsel, Licia E. Vaughn, of counsel, Barry K. Shelton, of counsel, San Diego, CA, Hiscock & Barclay, LLP, Robert A. Barrer, of counsel, Syracuse, NY, Counsel for Defendant/counterclaimant.

MEMORANDUM-DECISION AND ORDER

MORDUE, **District Judge.**

INTRODUCTION

In this action, FN1 plaintiffs Cornell University and Cornell Research Foundation, Inc. (collectively referred to as "plaintiff") claim that defendant Hewlett-Packard Company has infringed United States Patent 4,807,115 ("115 patent"). Defendant interposed counterclaims asserting that the patent is invalid. United States Magistrate Judge David E. Peebles issued an order on August 8, 2002, noting that "[i]t is envisioned that the parties will work together toward the preparation for a *Markman* hearing [pursuant to Markman v. Westview Instruments, Inc., 52 F.3d 967 (Fed.Cir.1995), *aff'd* 517 U.S. 370 388-89, 116 S.Ct. 1384, 134 L.Ed.2d 577 (1996) (' *Markman*')], to address claim construction in connection with the '115 patent." The order permitted the parties to proceed with any discovery necessary to prepare for the claim construction hearing, and stayed for a limited period all other discovery on issues not relating to claim construction.

FN1. The action was commenced December 27, 2001; the First Amended Complaint was filed September 6, 2002.

On defendant's motion, the Court held a *Markman* claim construction hearing on April 29 and 30, and May 1, 2003. Based on the record, the Court interprets the disputed claims as set forth herein.

THE INVENTION

Background

In setting forth the following simplified description of the invention-based primarily on the testimony of plaintiff's expert, James E. Smith, Ph.D., and defendant's expert, Michael J. Flynn, Ph.D.-the Court merely intends to provide background for the discussion of the issues. This section does not constitute any part of the Court's findings, which are set forth in subsequent sections herein. FN2

FN2. It is well established that a court need only construe claim elements or limitations that are in dispute between the parties. *See* Pall Corp. v. Hemasure, Inc., 181 F.3d 1305, 1308-09 (Fed.Cir.1999).

The patent characterizes the invention as an "instruction issuing mechanism for processors with multiple functional units." "Instructions" may be generally defined as expressions which specify operations a computer is to perform. Instructions are issued, executed and otherwise handled by the computer's "processor." A processor typically includes (1) an instruction unit, which stores and issues instructions fetched from a memory, and (2) an execution unit which performs the operations specified by the instruction. The instruction unit typically comprises an instruction fetch unit, an instruction decode unit and an instruction issue unit. The instruction issue unit has an instruction buffer (also known as instruction stack, issue buffer or instruction issue buffer), which temporarily holds the instructions that the instruction unit issues to the execution unit. Within the instruction buffer, the instructions are held in "cells" or "rows," one for each instruction.

Each instruction is placed in a certain format, or sequence of fields, each field corresponding to a separate part of the instruction. There is no universal instruction format or set of fields. A typical format comprises fields for "operation codes," which tell the processor what functions to perform; "source operands," which tell the processor the location of the data on which the functions are to be performed; and "destinations," which tell the processor where to deliver the results. Each row (or cell) in the instruction buffer is made up of a number of "registers," or containers, with a register to correspond to each of the separate parts, or fields, of the instruction. Thus, when an instruction is held in a row, the information in each field of the instruction is held in the corresponding register within that row.

When the instruction unit issues the instruction, the execution unit executes the particular operation designated by the operation code. The operation is performed in a functional unit, such as an adder or multiplier, in the execution unit. The execution unit may have multiple functional units.

Early processors issued and executed only one instruction at a time. Therefore, during the time required to complete that instruction ("clock cycle" or "machine cycle"), only one functional unit would be in use. This resulted in inefficient use of functional units in processors equipped with multiple functional units. One challenge in designing processors that issue multiple instructions using multiple functional units during a single clock cycle is that some instructions "depend" on others. For example, an instruction to divide a number by the sum of X and Y can not be executed until after the completion of the instruction to add X and Y; thus, the former instruction is "dependent" or "data dependent" on the latter.

The 115 patent teaches an instruction issuing mechanism capable of detecting and issuing those instructions which are not dependent and therefore can be performed during the same clock cycle without conflict or error. Such instructions are also referred to as "concurrencies" or "concurrently executable" instructions.

Dependencies

There are two primary types of dependencies in the context of instruction issuance for processors with multiple functional units. This section sets forth a simplified discussion of these two types of dependencies; a detailed explanation is found in the Declaration of James E. Smith, Ph.D., plaintiff's expert, Docket No. 68, paragraphs 29, 31-42.

The first type of dependencies, known as "essential," "true," "read after write" ("RAW"), or "(alpha)" [alpha] dependencies, occur where the result of one operation is needed for the performance of another. An example is set forth above: an instruction to divide a number by the sum of X and Y can not be executed until after the completion of the instruction to add X and Y. Such dependencies are inherent in the nature of the instructions and cannot be eliminated.

The second type of dependencies, known as "non-essential," "false" or "(beta)" [beta] dependencies, are not inherent in the nature of the instructions themselves. Rather, they are the result of "register conflicts" arising when a processor, which has a limited number of registers, reuses registers and overwrites the values stored therein. There are two types of false dependencies: "write after read" ("WAR") and "write after write" ("WAW"). The patent covers both types. FN3 As an example, a "write after read" ("WAR") type of false dependency occurs because an operand stored in a register cannot be overwritten until after it has been used to execute the instruction for which it has been fetched and stored. To illustrate, assume that X, an operand upon which Instruction A is to operate, is stored in register F0. Assume further that before Instruction A executes, Instruction B executes and writes its result, Y, to register F0, thus overwriting operand X. Then, when Instruction A executes, it will operate upon value Y instead of X, causing error.

FN3. That the patent covers-but is not limited to-both types of false dependencies (*i.e.,* WAR and WAW) is clear from the wording of the patent (in particular, see col. 7, ll. 60-64) and the testimony of both experts. It is not necessary to distinguish them for purposes of this decision.

As Dr. Smith explains, dependencies in instructions arise from inter-relationships among instructions that prevent one instruction from being issued until after another instruction is executed. Such dependencies may be discerned by looking at the actual sequence of the instructions; accordingly, the number and types of dependencies necessarily vary according to the format of the instructions.

The problem and the inventor's solution

In general terms, the problem addressed by the patent was the inability of early processors with multiple functional units to issue multiple instructions per clock cycle. In addition, the issuance of instructions in the order in which they entered the instruction buffer delayed the issuance of instructions which were dependency free but which were not "next in line." Prior approaches to the problem of multiple-instruction issuance and the related problem of out-of-order issuance are described in articles published as early as 1967. See, for example, R.M. Tomasulo, *An Efficient Algorithm for Exploiting Multiple Arithmetic Units,* IBM Journal, Jan. 1967; D.W. Anderson, F.J. Sparacio and R.M. Tomasulo, *The IBM System/360 Model 91: Machine Philosophy and Instruction Handling,* IBM Journal, Jan. 1967; Garold S. Tjaden and Michael J. Flynn, *Detection and Parallel Execution of Independent Instructions,* IEEE Transactions on Computers, Oct. 1970; and Edward M. Riseman and Caxton C. Foster, *The Inhibition of Potential Parallelism by Conditional Jumps,* IEEE Transactions on Computers, Dec. 1972.

The inventor's solution, as stated in the 115 patent, is to define an instruction issuing mechanism which detects concurrencies, that is, instructions without dependencies, and thus is able to issue multiple non-sequential instructions within a given clock cycle. FN4 This is achieved by "enriching" the instruction buffer, so that, in addition to fields for the instructions (*i.e.,* the operation codes, source operands and destinations), the instruction buffer also has an additional field or fields to keep track of the dependencies, if any, associated with each source or destination field of the instruction. Each such dependency detection field in a given row occupies a register in that row and has corresponding detection logic. This enrichment enables the instruction buffer to detect instructions which are free of dependencies and thus ready for execution. The patent refers to the enriched instruction buffer as a "dispatch stack."

FN4. For a detailed description of the invention, the Court refers the reader to Dr. Smith's Declaration, Docket No. 68, paragraphs 55-86, which clearly and correctly describes the invention.

The preferred embodiment also describes a "precedence count memory" ("PCM"), which contains an entry for each register and counts the number of times that instructions in the dispatch stack designate each register as a source and/or a destination; this facilitates the detection of dependency free instructions. The invention also includes a "reservation circuit" ("RC") which issues instructions having no dependencies to the appropriate functional unit.

The patent

The 115 patent was originally filed October 7, 1983, and issued February 21, 1989. The inventor, Hwa C. Torng, assigned it to plaintiff Cornell Research Foundation, Inc., a subsidiary of plaintiff Cornell University. The invention, an "instruction issuing mechanism for processors with multiple functional units," is described in the "Abstract" as follows:

An instruction issuing mechanism for boosting throughput of processors with multiple functional units. A Dispatch Stack (DS) and a Precedence Count Memory (PCM) are employed which allow multiple instructions to be issued per machine cycle. Additionally, instructions do not have to be issued according to their order in the instruction stream, so that non-sequential instruction issuance occurs. In this system, multiple instruction issuance and non-sequential instruction issuance policies enhance the throughput of processors with multiple functional units.

The "Background" section states that the invention relates to "an instruction issuing mechanism capable of detection of concurrencies in an instruction stream and issuing multiple instructions within a given machine cycle." It further states as follows:

The emergence of VLSI technology has stimulated research into the use of execution structures employed by processors having multiple functional units. Such high performance processors are generally partitioned to two sections, an instruction unit (IU) and an execution unit (EU) .... The IU and EU communicate with each other, with the IU fetching instructions from a memory and formulating and decoding those instructions. The IU is also employed to fetch operands if necessary. Additionally, the IU sends arithmetic/logic commands, that is, the decoded instructions together with a requisite operand to the EU. This invention relates specifically to an instruction issuing mechanism enhancing the throughput, that is, the number of instructions executed per unit of time of the EU.

With respect to prior art, the "Background" section concludes:

[In the prior art,] the IU loads sequences of instructions into an instruction stack from which instructions are issued to and then executed by the functional units. Within prior art systems employing multiple functional units, at most one instruction is issued from the instruction stack during every machine cycle. As a result, the instructions execution rate of such an EU structure cannot be greater than the inverse of the machine cycle time ... [references omitted].

In the section headed "Summary of the Invention," the patent states:

Given the deficiencies in prior art computer systems employing multiple functional units, it is an object of the present invention to define an instruction issuing mechanism which is capable of detecting concurrencies in an instruction stream and issuing multiple instructions within a given machine cycle and which may be extended to modify the instruction stream.

It is a further object of the invention to define an arithmetic engine implemented in a VLSI environment that substantially enhances the throughput of such a processor.

Yet another object of this invention is to formulate and define an instruction issuing mechanism for arithmetic engines utilizing multiple functional units to achieve high instruction execution rates.

A further object of this invention is to define a dispatch stack component of the instruction issuing mechanism operating in a FIFO mode and detecting instructions that can be issued at each machine cycle.

Still another object of this invention is to define a precedent count memory component of the instruction issuing mechanismto assign alpha- and beta-values for each instruction being loaded into the dispatch stack and to assign general purpose registers to operands to enhance possible execution concurrencies.

These and other objects of the present invention are achieved by an instruction issuing mechanism which detects concurrencies and issues multiple instructions within a given machine cycle.

The 115 patent describes a specific embodiment of the invention, referred to as the "preferred embodiment." The preferred embodiment of the claimed invention includes (1) a dispatch stack ("DS"), *i.e.,* an "enriched" instruction buffer, which cooperates with a precedence count memory ("PCM") to detect "concurrently executable (*i.e.,* dependency free) instructions"; and (2) a reservation circuit which then issues these multiple, possibly non-sequential, instructions to the execution unit within a single clock cycle. In the preferred embodiment, the dispatch stack is "enriched" in that it (1) has additional fields and accompanying logic for resolving dependencies in instructions; and (2) stores instructions whose dependency values are "initialized" (*i.e.,* initially assigned by the PCM). The dispatch stack and PCM eliminate the time-consuming comparison of all the instructions that would otherwise be necessary to ascertain whether the instructions contained dependencies. The preferred embodiment of the present invention is described as operating on a sample set of instructions that have a common instruction format (*i.e.,* OP, S1, S2, D) and contain both essential and false dependencies.

The patent claims

The 115 patent has two categories of patent claims: (1) instruction issuing system, or means-plus-function claims (*i.e.,* claims 1-5, and 14); and (2) method of issuing instructions, or method claims (*i.e.,* claims 6-13 and 15-19). Each category of claims also has both independent claims (*i.e.,* means-plus-function claims 1 and 14 and method claims 6 and 15) and dependent claims which incorporate by reference a previous claim and specify a further limitation to the invention. *See* 35 U.S.C. s. 112(4). FN5

FN5. Section 112(4) provides:
[A] claim in dependent form shall contain a reference to a claim previously set forth and then specify a further limitation of the subject matter claimed. A claim in dependent form shall be construed to incorporate by reference all the limitations of the claim to which it refers.

The full text of the 115 patent's 19 claims is as follows:

I claim:

1. An instruction issuing system for a processor including an execution unit having multiple functional units comprising:

an instruction issuing unit receiving instructions from a memory, operating on instructions and forwarding instructions to said execution unit, said instruction issuing unit including means for detecting the existence of concurrencies in said instructions received from said memory; and

said instruction issuing unit further including means for issuing multiple instructions and non-sequential instructions to said execution unit within a single processor cycle when a concurrency is detected by said means for detecting the existence of concurrencies in said instructions.

2. The instruction issuing system of claim 1 wherein said means for detecting the existence of concurrencies comprises a dispatch stack receiving instructions from said memory and operating in a first-in first-out manner, said dispatch stack receiving instructions having instruction fields of OP, S1, S2, D, where:

OP is the arithmetic/logic operation to be performed,

S1 specifies a register which provides the first of two or the only operand called for,

S2 specifies a register yielding the second operand, and,

D specifies a register receiving the result of the arithmetic/logic operation.

3. The instruction issuing system of claim 2, wherein said means for detecting the existence of concurrencies in said instruction issuing unit further comprises a precedent count memory, said precedent count memory providing fields of a first value ((alpha)) [alpha] to instruction fields S1 and S2 indicative of the number of times a register S1(S2) is used as destination register in preceding, uncompleted instructions and, a second value ((beta)) [beta] to register field D indicative of the number of times that register D is designated as a source register in preceding, uncompleted instructions.

4. The instruction issuing system of claim 2 wherein said means for detecting the existence of concurrencies includes a precedent count memory for providing values to each instruction loaded into said dispatch stack indicative of the number of times a register for a particular field is designated as a source register in preceding uncompleted instructions.

5. The instruction issuing system of claim 3 wherein said means for detecting the existence of concurrencies determines an issue index (I^{2}) for each instruction in said dispatch stack wherein I^{2}+(alpha) (S1)+ (alpha) (S2)+ (beta) (D) such that when an instruction having I^{2}=0 is encountered by said means for detecting the existence of concurrencies, said means for issuing multiple instructions reserves an available functional unit and issues said instruction to it.

6. A method of issuing instructions for a processor having multiple functional units comprising the steps of:

reading in and storing instructions from an instruction stream into a dispatch stack, said instructions having an instruction format of OP, S1, S2, D, where:

OP is the arithmetic/logic operation to be performed;

S1 is the register which provides the first of two or the only operand called for;

S2 is the register yielding the second operand; and

D is the register receiving the result of the arithmetic/logic operation;

detecting the existence of concurrencies in instructions stored in said dispatch stack and;

issuing multiple instructions and non-sequential instructions within a given processor cycle when the existence of concurrencies is detected.

7. The method of claim 6 wherein said step of detecting further comprises the steps of:

determining the number of times that individual registers in said processor are used as destination registers in preceding, uncompleted instructions, determining the number of times the individual registers in said processor are used as source registers in preceding uncompleted instructions, and providing an indication of the determination in said instruction format for each instruction in said dispatch stack.

8. The method of claim 7 wherein said step of issuing multiple instructions further comprises the step of immediately issuing a first instruction from said dispatch stack to an available functional unit when said instruction does not have any data dependencies with preceding issued instructions which have not yet been completed.

9. The method of claim 8 wherein an instruction is data dependent upon a preceding, uncompleted instruction if one of its source registers is the destination register of the uncompleted instruction.

10. The method of claim 8 wherein an instruction is data dependent upon a preceding, uncompleted instruction if its destination register is a source register of the uncompleted instruction.

11. The method of claim 7 wherein said step of determining comprises providing first values ((alpha)) to register fields S1(S2) indicative of the number of times register S1(S2) is used as destination registers in preceding, uncompleted instructions and a second value ((beta)) to register field D indicative of the number of times that register D is designated as a source register in to preceding, uncompleted instructions.

12. The method of claim 11 further comprising the steps of content addressing the SI, S2 and D fields following the completion of an issued instruction and, appropriately decrementing the values of (alpha)'s, the values of (beta)'s and updating the dispatch stack by advancing subsequent instructions stored therein and adding new instructions from said instruction stream occupy empty portions at the bottom of said dispatch stack.

13. The method of claim 11 further comprising the steps of updating the values of (alpha) and (beta) such that when a register is assigned to an instruction as a source register its present (beta) value is used as (alpha) (S1) or (alpha) (S2) and its (beta) value is incremented by 1 and, when a register is assigned to an instruction as its destination register, its present (beta) value is used as the (beta) (D) field and its (alpha) value is incremented by 1 and further when an issued instruction is completed the (beta) values of each of its source registers is decremented by 1 and the a value of its destination register is decremented by 1.

14. An instruction issuing system for a processor including an execution unit having multiple functional units comprising:

an instruction issuing unit receiving instructions from a memory, said instruction issuing unit operating on instructions and forwarding instructions to said execution unit, said instruction issuing unit including means for detecting the existence of a plurality of instructions received from said memory which are concurrently executable; and said instruction issuing unit further including means for issuing multiple instructions and non-sequential instructions to said execution unit within a single processor cycle when concurrently executable instructions are detected by said means for detecting the existence of concurrently executable instructions in said instructions.

15. A method of issuing instructions for a processor having an execution unit with multiple functional units comprising the steps of:

reading in and storing instructions from an instruction stream into a dispatch stack;

detecting the existence of plurality of instructions which are concurrently executable from those instructions stored in said dispatch stack; and

issuing multiple instructions and non-sequential instructions within a given processor cycle when said plurality of concurrently executable instructions are detected.

16. The method of claim 15 wherein said step of detecting further comprises the steps of:

determining the number of times that individual registers in said processor are used as destination registers in preceding, uncompleted instructions, determining the number of times the individual registers in said processor are used as source registers in preceding uncompleted instructions, and providing an indication of the determination in said instruction format for each instruction in said dispatch stack.

17. The method of claim 16 wherein said step of issuing multiple instructions further comprises the step of immediately issuing a first instruction from said dispatch stack to an available functional unit when said instruction does not have any data dependencies with preceding issued instructions which have not yet been completed.

18. The method of claim 15 wherein an instruction is data dependent upon a preceding, uncompleted instruction if one of its source registers is the destination register of the uncompleted instruction.

19. The method of claim 15 wherein an instruction is data dependent upon a preceding, uncompleted instruction if its destination register is a source register of the uncompleted instruction.

With respect to the prosecution history of the 115 patent, the Court finds no relevant amendment to the claims nor any other aspect of the prosecution history which could have the effect of narrowing the disputed claim limitations. The file history does show that claim 3 was amended for the purpose of differentiating it from the claim from which it depends; this will be discussed below where pertinent.

APPLICABLE LAW

[1] Analysis of a patent infringement complaint involves two inquiries: "determination of the scope of the claims, if there is a dispute as to claim interpretation or construction; followed by determination of whether properly interpreted claims encompass the accused structure." Mannesmann Demag Corp. v. Engineered Metal Prods. Co., Inc., 793 F.2d 1279, 1282 (Fed.Cir.1986). Presently before this Court is the first of these two inquiries, claim construction. FN6 Construction of the claims is a question of law for the court. *See* Markman, 517 U.S. at 388, 116 S.Ct. 1384.

FN6. Accordingly, the Court does not consider defendant's discussion of the infringement issue, in particular in paragraphs 60-63 of Dr. Flynn's September 11, 2002 declaration, on pages 18-19 of its *Markman* brief and on page 1 of its post- *Markman* brief.

[2] [3] [4] [5] Interpretation of a disputed claim "begin[s] with an examination of the intrinsic evidence, *i.e.,* the claims, the other portions of the specification, and the prosecution history (if any, and if in evidence)." Altiris, Inc. v. Symantec Corp., 318 F.3d 1363, 1369 (Fed.Cir.2003). The purpose of the claims is not to explain the technology or how it works, but to state the legal boundaries of the patent grant. *See* S3, Inc. v. nVIDIA Corp., 259 F.3d 1364, 1369 (Fed.Cir.2001). The specification, which sets forth a written description of the invention, is always "highly relevant to the claim construction analysis"; indeed, it is "the single best guide to the meaning of a disputed term." *See* Vitronics Corp. v. Conceptronic, Inc., 90 F.3d 1576, 1582 (Fed.Cir.1996). The court may also consider the prosecution history (or "file history"), if it is available. This history, containing the complete record of the proceedings before the Patent and Trademark Office including any representations made by the applicant regarding the scope of the claims, "is often of critical significance in determining the meaning of the claims." *See* Vitronics, 90 F.3d at 1582-83.

[6] [7] When analyzing the intrinsic evidence, a court should "start with the language of the claims and apply a heavy presumption that the claim terms carry their ordinary meaning as viewed by one of ordinary skill in the art." FN7 *See* Altiris, 318 F.3d at 1369. "[W]hile it is true that claims are to be interpreted *in light of* the specification and with a view to ascertaining the invention, it does not follow that limitations from the specification may be read into the claims." *See* Sjolund v. Musland, 847 F.2d 1573, 1581 (Fed.Cir.1988) (emphasis in original). Nor should particular embodiments in the specification be read into the claims; the general rule is that the claims of a patent are not limited to the preferred embodiment. *See* Texas Digital Systems, Inc. v. Telegenix, Inc., 308 F.3d 1193, 1204 (Fed.Cir.2002); *see also* Northrop Grumman Corp. v. Intel Corp., 325 F.3d 1346, 1355 (Fed.Cir.2003) ("Absent a clear disclaimer of particular subject matter, the fact that the inventor may have anticipated that the invention would be used in a particular way does not mean that the scope of the patent is limited to that context.").

FN7. Indeed, patent documents "need not include subject matter that is known in the field of the invention and is in the prior art, for patents are written for persons experienced in the field of the invention." *See* S3, Inc. v. nVIDIA Corp., 259 F.3d 1364, 1371 (Fed.Cir.2001).

[8] [9] [10] The court may receive extrinsic evidence, including the testimony of expert witnesses, to educate itself about the invention and the relevant technology; it is, however, ultimately for the court-not the experts-to construe the claims. *See* Markman, 517 U.S. at 388, 116 S.Ct. 1384. The court may not use such evidence to arrive at a claim construction that is clearly at odds with the construction mandated by the intrinsic evidence. *See* Karlin Technology, Inc. v. Surgical Dynamics, Inc., 177 F.3d 968, 971 (Fed.Cir.1999). The court may also consult dictionaries, encyclopedias and treatises which were publicly available at the time of the patent; these are reliable sources of information on the established meanings that would have been attributed to the terms of the claims by those of skill in the art at the pertinent time, provided that such definition does not contradict the patent documents. *See* Texas Digital, 308 F.3d at 1202 ("Dictionaries are always available to the court to aid in the task of determining meanings that would have been attributed by those of skill in the relevant art to any disputed terms used by the inventor in the claims."); Vitronics, 90 F.3d at 1584 n. 6. The general meanings gleaned from such sources must, however,

"always be compared against the use of the terms in context[.]" Brookhill-Wilk 1,LLC v. Intuitive Surgical, Inc., 334 F.3d 1294, 1300 (Fed.Cir.2003).

[11] [12] As will be seen below, in construing the 115 patent this Court applies the doctrine of claim differentiation. This doctrine is "based on the common sense notion that different words or phrases used in separate claims are presumed to indicate that the claims have different meanings and scope[.]" Karlin Technology, 177 F.3d at 971-72; *see* Comark Communications, Inc. v. Harris Corp., 156 F.3d 1182, 1187 (Fed.Cir.1998). Accordingly, limitations stated in dependent claims should not ordinarily be read into the independent claim from which they depend. *See* Karlin Technology, 177 F.3d at 972. Stated differently: "There is presumed to be a difference in meaning and scope when different words or phrases are used in separate claims. To the extent that the absence of such difference in meaning and scope would make a claim superfluous, the doctrine of claim differentiation states the presumption that the difference between claims is significant." Tandon Corp. v. United States Int'l Trade Comm'n, 831 F.2d 1017, 1023 (Fed.Cir.1987).

[13] [14] [15] In the case at bar, it is undisputed that certain claims are "means plus function" claims authorized by 35 U.S.C. s. 112(6). FN8 Section 112(6) "allows patent applicants to claim an element of a combination functionally, without reciting structures for performing those functions." Envirco Corp. v. Clestra Cleanroom, Inc., 209 F.3d 1360, 1364 (Fed.Cir.2000). Thus, a means-plus-function limitation recites a function to be performed rather than definite structure or materials for performing that function. *See* Chiuminatta Concrete Concepts, Inc. v. Cardinal Indus., Inc., 145 F.3d 1303, 1307 (Fed.Cir.1998). In construing such a claim, a court first identifies the claimed function; next, it determines what structures disclosed in the written description correspond to the "means" for performing that function. *See* Wenger Mfr., Inc. v. Coating Mach. Sys., Inc., 239 F.3d 1225, 1233 (Fed.Cir.2001). The court may not import functional limitations that are not recited in the claim, or structural limitations from the written description that are unnecessary to perform the claimed function. *See id.* In means-plus-function claims, if a conflict arises between the application of the doctrine of claim differentiation and the statutory requirements of section 112(6), the requirements of section 112(6) prevail. *See id.*

FN8. Paragraph 6 of section 112 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.

MOTION

[16] At the hearing, the Court reserved decision on plaintiff's objection to defendant's introduction of the book, *Superscalar Microprocessor Design,* by Mike Johnson of Advanced Micro Devices (Prentice-Hall, Inc.1991) and of Dr. Flynn's testimony based thereon. Plaintiff argues that this evidence is inadmissible because, in construing claim terms, it is not proper to consider references that are dated after the patent issued. *See* Brookhill-Wilk, 334 F.3d at 1299 (holding that it was error to rely on references dated well after the patent was issued; such references "are not contemporaneous with the patent [and] do not reflect the meanings that would have been attributed to the words in dispute by persons of ordinary skill in the art as of the grant of the ... patent[.]"). Defendant argues that the Johnson book is admissible as "historical perspective[.]"

Upon reviewing the portions of Johnson's book on which defendant relies, the Court sees no basis to conclude that Johnson's discussion in 1991 of "dispatch stack" and other "historical" references necessarily reflects the meanings that would have been attributed to the disputed terms by persons of ordinary skill in the art during the pertinent period. Thus, the Court grants plaintiff's motion and strikes all references to and discussion of Johnson's book.

DISCUSSION

The claim construction disputes between the parties center on two issues. First, defendant urges that the patent teaches only one type of dispatch stack, *i.e.,* one that is enriched with fields for both essential and false dependencies. According to defendant's construction, then, the patent does not cover a product in which the instructions entering the dispatch stack do not have false dependencies. In response, plaintiff argues that, although the preferred embodiment operates on sample instructions containing both essential and false dependencies, the claims themselves are not so limited. Thus, plaintiff contends, the invention covers both products in which the instructions entering the dispatch stack do not have false dependencies as well as products in which they do.

Second, the parties disagree as to the structure and function of the dispatch stack, precedence count memory and reservation circuit. These issues will be discussed in the context of the pertinent claims.

In order to perform its role of construing the patent as it would be understood by a person experienced in the field at the time of the invention, *see* S3 Inc., 259 F.3d at 1371, the Court must determine the level of knowledge possessed by one skilled in the art at the time in question. Based upon the record, including the extensive prosecution history of the 115 patent, the Court finds that the relevant field in the art as of October 7, 1983, and until the grant of the application on February 21, 1989, is that of computer architecture and more specifically of high performance processors. The Court further finds that the relevant person of skill in the art as of that time would possess a very high level of skill in this field, that is, the level of skill of "a graduate electrical engineering or computer sciences student intimately familiar with computer architecture and especially high performance processor design and operation," or "an individual who has at least a Bachelor of Science degree in electrical engineering or computer science and who has intimately worked for a few years in the field of high performance processor design and operation."

One of skill in the pertinent art as of the time of the invention would have understood processors and common components used with processors, such as instruction buffers and memories, and would have understood common terminology, such as dependencies and instructions, without requiring any description of their particulars or how they operate. Further, such a person would have known about the challenges associated with multiple and out-of-order instruction issuance and execution and would have known that such challenges presented a long-standing technical problem. This person would have understood register renaming, would have known that it was a technique which was separate and distinct from detection of dependencies, would have known that it could remove or eliminate nonessential dependencies in computer instructions, and would have known that it could not remove or eliminate essential dependencies.

One of skill in the art would also have readily understood that a reservation circuit was a common component used with processors and had conventional arbitration logic, and would appreciate from reading the patent that logic (i.e. circuitry) would be used to implement the dispatch stack ("DS"), precedence count memory ("PCM"), and reservation circuit ("RC"), including the various fields in the DS and PCM. Such a person would also be familiar with the Keller, Tomasulo, Tjaden-Flynn and Riseman-Foster articles (R.M. Keller, *Look-Ahead Processors,* Computing Surveys, Dec. 1975; R.M. Tomasulo, *An Efficient Algorithm for Exploiting Multiple Arithmetic Units,* IBM Journal, Jan. 1967; D.W. Anderson, F.J. Sparacio and R.M. Tomasulo, *The IBM System/360 Model 91: Machine Philosophy and Instruction Handling,* IBM Journal, Jan. 1967; Garold S. Tjaden and Michael J. Flynn, *Detection and Parallel Execution of Independent Instructions,* IEEE Transactions on Computers, Oct. 1970; Edward M. Riseman and Caxton C. Foster, *The Inhibition of Potential Parallelism by Conditional Jumps,* IEEE Transactions on Computers, Dec. 1972).

Dr. Smith satisfactorily explained the discrepancies between the draft tutorials and his first (subsequently corrected) declaration and his final declaration and testimony. The Court finds that these discrepancies, which Dr. Smith acknowledged reflected errors in the drafts and first declaration, do not undermine his credibility or cast doubt upon the veracity or reliability of his final declaration and testimony.

The Court has considered defendant's request that the Court not consider paragraphs 2 through 25 of the McKenzie declaration (Dkt. No. 56). The Court agrees that these paragraphs are not factual evidence but rather are in the nature of a memorandum of law advocating plaintiff's position. The Court evaluates the declaration accordingly. Defendant's clarification of paragraph 26 is noted.

As will be seen below, in construing the 115 patent, the Court finds the intrinsic evidence adequate to resolve all interpretation issues. The Court considers extrinsic evidence solely to educate itself about the invention and the relevant technology.

Stipulated Claim Term Definitions

The parties stipulated to the following claim term definitions:

1. Processor (claims 1, 6, 7, & 14-16)

A device that interprets and executes instructions.

2. Instruction issuing system (claims 1-5 & 14)

A system comprising an instruction issuing unit.

3. Issuing instructions (claims 6 & 15)

Instructions are sent to their respective functional units along with their respective operands for execution.

4. Arithmetic/logic operation (claims 2 & 6)

Instructions are sent to their respective functional units along with their respective operands for execution.

5. Concurrencies (claims 1-6); a plurality of concurrently executable instructions (claims 14, 15).

The parties dispute the meaning of these two terms, although they agree that both terms should be construed identically.

6. Functional Unit(s) (claims 1, 5, 6, 8, 14, 15, & 17)

A functional unit performs arithmetic/logic operations on various data types.

7. Memory (claims 1-4 & 14)

Data storage elements outside the processor for storage of instructions and data.

8. Non-sequential instructions (claims 1, 6, 14, & 15)

Instructions ordered differently than the order in which they appeared in the instruction stream.

9. Single processor cycle (claims 1 & 14)

The shortest amount of time in which a functional unit can complete an operation.

10. Source register (claims 3, 4, 7, 9-11, 13,16, 18, & 19)

A register storing an instruction operand.

11. Destination register (claims 3, 7, 9-11, 13, 16, 18, 19)

A register storing an instruction result.

12. (alpha) ("alpha") (claims 3 & 11-13)

(alpha) is the number of times that a register is used as a destination register in preceding, uncompleted instructions.

13. (alpha) (S1) (claims 5 & 13)

(alpha) (S1) is the number of times that an instruction's S1 register is used as a destination register in preceding, uncompleted instructions.

14. (alpha) (S2) (claims 5 & 13)

(alpha) (S2) is the number of times that an instruction's S2 register is used as a destination register in preceding, uncompleted instructions.

15. (beta) ("beta") (claims 3 & 11-13)

(beta) is the number of times that a register is designated as a source register in preceding, uncompleted instructions.

16. (beta) (D) (claims 5 & 13)

(beta) (D) is the number of times that an instruction's D register is designated as a source register in preceding, uncompleted instructions.

17. Register (claims 2-4, 6, 7, 9-11, 13, 16, 18, & 19)

A data storage element within the processor.

18. Operand (claims 2 & 6)

A value supplied for an operation performed by a functional unit.

19. Result (claims 2 & 6)

A value produced by an operation performed by a functional unit.

20. Increment (claim 13)

Add to the value by the specified amount.

21. Decrement (claims 12 & 13)

Subtract from the value by the specified amount.

22. Execution Unit (claim 1, 14, & 15)

An execution unit is a device containing multiple functional units for executing arithmetic/logic instructions.

Disputed Terms

The Court now turns to construe the disputed terms of the claims in the 115 patent.

Instruction (claims 1, 14 and 15)

[17] Defendant's proposed definition is as follows:

An arithmetic/logic operation, containing at least OP, SI, S2, and D fields, where:

OP denotes the arithmetic/logic operation to be performed;

S1 specifies the register which provides the first of two operands, or the only operand called for;

S2 specifies the register which yields the second of the two operands required; and

D denotes the register receiving the result of the arithmetic/logic operation.

Plaintiff urges that the term "instruction" in claims 1, 14 and 15 should be given its ordinary meaning, which is "an expression that specifies one or more operations and identifies the applicable operands." Plaintiff contends that there is no basis in the patent or prosecution history to require that the instruction be in a particular format.

Plaintiff's proposed definition is supported by the contemporaneous technical dictionaries cited by plaintiff; defendant cites no sources to the contrary. *See, e.g., IEEE STANDARD Dictionary of Electrical and Electronics Terms, 2d ed.* (Institute of Electrical and Electronics Engineers, Inc.1978) (defining "instruction" as "[a] statement that specifies an operation and the values or locations of its operands."). Moreover, Drs. Smith and Flynn agree that there is no single instruction format or set of fields that is universally used by computer programmers.

Defendant does not challenge plaintiff's representation of the ordinary and customary meaning of the term instruction, but urges that within the four corners of the patent, the term has a specialized meaning, that is, that it is limited to the particular format of OP, S1, S2, D. Although this particular instruction format is found in the preferred embodiment, it is not found in all of the claims. It is well established that it is improper to read a particular preferred embodiment into the claims. *See* Texas Digital, 308 F.3d at 1203. Moreover, inasmuch as the OP, S1, S2, D instruction format is found only in some claims (*e.g.,* dependent claim 2) but not in others (*e.g.,* independent claim 1, from which claim 2 depends), defendant's proposed construction violates the doctrine of claim differentiation. Thus, defendant has not overcome the heavy presumption that the claim term carries its ordinary and customary meaning. *See* Altiris, Inc., 318 F.3d at 1369.

The contemporaneous ordinary and customary meaning of the term "instruction" is fully consistent with the patent. There is no basis in the patent or the prosecution history to limit the instruction in claims 1, 14 and 15 to a particular format. As will be seen below, the Court construes the patent as teaching a technique for detecting dependency free computer instructions and finds no basis to construe the technique as limited to environments in which the instructions have particular formats. Thus, the Court construes the term "instruction" in independent claims 1, 14 and 15 to mean "an expression that specifies one or more operations and identifies the applicable operands."

Instruction (claim 6)

[18] Independent claim 6 recites a specific instruction format: OP, S1, S2, D. Accordingly, the term "instruction" in claim 6 means "an expression that has a specific format (*i.e.,* OP, S1, S2, D)."

Concurrencies (claims 1 and 6); a plurality of instructions which are concurrently executable (claims 14 and 15)

[19] The parties agree that the terms "concurrencies" and "a plurality of instructions which are concurrently executable" have the same meaning. They further agree that the terms have no special meaning in the computer field. The parties' definitions differ, however, with respect to two elements. Defendant defines the two terms as follows: "Instructions (1) for which it is determined that zero (alpha) and (beta) dependencies exist and the required functional units are available, and (2) that can be simultaneously issued for execution." Accordingly, defendant's definition of concurrencies comprises two elements: (1) the absence of (alpha) and (beta) dependencies, and (2) the availability of appropriate functional units.

Plaintiff's definition, on the other hand, requires only that the instructions be free of dependencies. Specifically, plaintiff defines the disputed terms as: "a plurality of instructions that are ready to be issued because they are dependency free." Therefore, plaintiff's definition does not refer to any particular dependencies (such as (alpha) and/or (beta) dependencies), nor does it require the availability of appropriate functional units.

The Court agrees with plaintiff's definition. First, there is no basis in the patent or the prosecution history to read the terms as requiring that the instructions be in any particular format or have a particular set of fields. Nor is there a basis to read the terms as requiring that the instructions be free of any particular type of dependency, as requiring that any particular type of dependency have previously existed, or as requiring that dependencies have been removed in any particular manner. Indeed, the Court has already construed the term "instruction" broadly to mean "an expression that specifies one or more operations and identifies the applicable operands"; this construction includes instructions which never had false ((beta)) dependencies, in which case there is no need to "determine" that no false dependencies exist.

Application of the doctrine of claim differentiation also supports this view. For example, dependent claim 7 specifies false dependencies and is thus differentiated from independent claim 6 from which it depends. Likewise, dependent claim 16 specifies false dependencies and is thus differentiated from claim 15 from which it depends. While the Court is aware that claim differentiation is not a rigid rule, *see* Comark, 156 F.3d at 1187, here, application of the doctrine is fully consistent with the patent as a whole.

Next, there is no basis to conclude that an appropriate functional unit must be available before instructions may constitute "concurrencies" or "a plurality of instructions which are concurrently executable." Rather, for an instruction to be ready to be executed it must simply be dependency free. This construction is supported by the separation of the detecting function and the issuing function in the means-plus-function claims (independent claims 1 and 14). In these claims, the detecting function concerns the detection of instructions which are ready to be executed because they lack dependencies. The issuing function operates on instructions which have been determined to be dependency free; the issuance of such instructions to appropriate and available functional units is a distinct function performed by a distinct structure. Similarly, the method claims (independent claims 6 and 15) describe a two-step process: first, detecting dependency free instructions, and second, issuing them to available functional units.

Plaintiff also refers the Court to the plain meaning of the words "concurrently" and "executable." The Oxford English Dictionary Online (Oxford University Press, 2003) FN9 ("OED") defines concurrently as "[i]n a concurrent or concurring manner; in concurrence." The OED definition of concurrent includes "occurring together, as events or circumstances," and the definition of concurrency includes "a running together in place or time." OED defines executable as "[t]hat can be executed, performed, or carried out." Thus, applying the ordinary meaning of the words, the terms refer to two or more instructions which are in a state of readiness to be carried out at the same time. The ordinary meaning of the words is thus consistent with the patent and supports the Court's construction.

FN9. The Court is aware that this source is not contemporaneous with the patent; however, there is no reason to believe that the meanings of these non-technical words has altered since the time of the patent, nor does defendant dispute the ordinary meanings of these words.

Accordingly, the Court reads "concurrencies" and "plurality of instructions which are concurrently executable" as meaning "a plurality of instructions that are ready to be executed at the same time because they are dependency free."

Dispatch stack

[20] The parties agree that the term "dispatch stack" was not a commonly used term in 1983. The inventor, Dr. Torng, stated in his deposition that he believed he had coined the term "dispatch stack" and that each of the two words "dispatch" and "stack" was used in the industry at the time of the patent. Dr. Smith testified that the term was used at the time,FN10 althoughperhaps not "commonly used." He further stated that the term was "easily understood" in 1983 because each of the two words "dispatch" and "stack" had commonly used meanings at the time. Dr. Smith testified that "dispatch" means "issue" or "instruction" and that "stack" means "buffer." Thus, dispatch stack simply means "issue buffer" or "instruction buffer."

FN10. Dr. Smith pointed out that the term dispatch stack had been used prior to the 115 patent application in the article by Edward M. Riseman and Caxton C. Foster, *The Inhibition of Potential Parallelism by Conditional Jumps* (IEEE Transactions on Computers, Dec. 1972, p 1407, col. 1, par 3), which referred to an "instruction stack" or "dispatch stack." The article uses the full-comparison method and describes register renaming as a method of eliminating false dependencies. As Dr. Smith explains, register renaming is not an alternative to the dispatch stack; it is independent, and the dispatch stack can operate both in systems with register renaming and in systems without it.

In general terms, as discussed above, the instruction buffer, which is part of the instruction issue unit, holds the instructions that the instruction issue unit will issue to the execution unit. The instructions, which have been placed in a certain format, or sequence of fields, are held in the instruction buffer in registers, or temporary memories, each of which corresponds to a particular field. For example, a typical instruction might comprise the following: an "operation code," which tells the processor what function to perform; "source operands," which tell the processor the location of the data on which the function is to be performed; and the "destination," which tells the processor where to deliver the result.

According to plaintiff, the patent teaches an instruction buffer enriched with a dependency detection technique, or a dispatch stack. This detection technique is the addition of fields which hold values that track dependencies associated with the instruction, and the accompanying logic. Essentially, defendant argues that the dispatch stack in the 115 invention is enriched in only one manner, that is, the manner described in the preferred embodiment. Plaintiff argues, however, that the dispatch stack is enriched by the addition of whatever fields correspond to the number and types of dependencies inherent in the particular instructions and format used.

As Dr. Smith explains, dependencies arise from inter-relationships among instructions. Because there is no single universal format or set of instruction fields, it follows that the number and types of dependency fields in the dispatch stack may differ according to the nature of the instructions and the possible dependencies inherent in them. For this reason, the number and types of dependency fields in the dispatch stack differ in the context of different patent claims. Thus, plaintiff defines dispatch stack in independent claims 1, 14 and 15 as:

[A]n enriched instruction buffer (or DS) including:

(i) A cell for each instruction, each cell having at least the following resultant fields:

(1) an operation field (*e.g*. OP) of the instruction;

(2) a field for each source register (one or more- *e.g*. S1) specified by the instruction;

(3) an essential dependency field (*i.e.* (alpha) (Si), *e.g.* (alpha) (S1)) corresponding to each source register specified by the instruction;

(4) a field for each destination register (one or more- *e.g.* D) specified by the instruction; and

(ii) Logic that

(1) decrements a value >0 in the essential dependency field;

(2) determines when the value corresponding to (alpha) (Si) is zero; and

(3) obtains an initial value for (alpha) (Si).

With respect to claims 2 and 6, in which the instruction format is specified as OP, S1, S2, D, thus specifying two source fields, the dispatch stack is defined as follows:

[A]n enriched instruction buffer (or DS) including:

(i) A cell for each instruction, each cell having at least the following resultant fields:

(1) an operation field (*e.g.* OP) of the instruction;

(2) a field for each of two source registers (*e.g.* S1 and S2) specified by the instruction;

(3) two essential dependency fields (*i.e.* (alpha) (S1) and (alpha) (S2)) corresponding to the two source registers (*e.g.* S1 and S2) specified by the instruction;

(4) a field for each destination register (one or more- *e.g.* D) specified by the instruction; and

(ii) Logic that

(1) decrements a value >0 in each essential dependency field;

(2) determines when the values corresponding to (alpha) (S1) and (alpha) (S2) are zero; and

(3) obtains initial values for (alpha) (S1) and (alpha) (S2).

Dispatch stack in claims 3 and 4, which explicitly refer to false dependency fields and to a precedence count memory, or PCM,FN11 means:

FN11. The precedence count memory, or PCM, is discussed below, at page 139.

An enriched instruction buffer (or DS) including:
(i) A cell for each instruction, each cell having at least the following resultant fields:

(1) an operation field (*e.g*. OP) of the instruction;

(2) a field for each of two source registers (*i.e*. S1 and S2) specified by the instruction;

(3) two essential dependency fields (*i.e*. (alpha) (S1) and (alpha) (S2)) corresponding to registers S1 and S2 specified by the instruction;

(4) a field for each destination register (one or more- *e.g*. D) specified by the instruction;

(5) a false dependency field (*i.e*. (beta) (D)) corresponding to the destination register specified by instruction; and

(ii) Logic that

(1) decrements a value >0 in the essential and false dependency fields;

(2) determines when the value corresponding to (alpha) (S1), (alpha) (S2) and (beta) (D) are zero;

(3) obtains initial values for (alpha) (S1), (alpha) (S2) and (beta) (D) from the PCM.

Dispatch stack in claim 5, which has an issue index, or I ^{2}, means:

An enriched instruction buffer (or DS) including:

(i) A cell for each instruction, each cell having at least the following resultant fields:

(1) an operation field (*e.g*. OP) of the instruction;

(2) a field for each of two source registers (*i.e*. S1 and S2) specified by the instruction;

(3) two essential dependency fields (*i.e*. (alpha) (S1) and (alpha) (S2)) corresponding to registers S1 and S2 specified by the instruction;

(4) a field for each destination register (one or more- *e.g*. D) specified by the instruction;

(5) a false dependency field (*i.e*. (beta) (D)) corresponding to the destination register specified by instruction;

(6) an Issue Index (I ^{2}) field equal to (alpha) (S1)+ (alpha) (S2)+ (beta) (D); and

(ii) Logic that

(1) decrements a value >0 in the essential and false dependency fields;

(2) determines when the values corresponding to (alpha) (S1), (alpha) (S2) and (beta) (D) are zero;

(3) obtains initial values for (alpha) (S1), (alpha) (S2) and (beta) (D).

Dispatch stack in claims 7-13 means:

An enriched instruction buffer (or DS) including:

(i) A cell for each instruction, each cell having at least the following resultant fields:

(1) an operation field (*e.g.* OP) of the instruction;

(2) a field for each of two source registers (*i.e.* S1 and S2) specified by the instruction;

(3) two essential dependency fields (*i.e.* (alpha) (S1) and (alpha) (S2)) corresponding to registers S1 and S2 specified by the instruction;

(4) a field for each destination register (one or more- *e.g.* D) specified by the instruction;

(5) a false dependency field (*i.e.* (beta) (D)) corresponding to the destination register specified by instruction; and

(ii) Logic that

(1) decrements a value >0 in the essential and false dependency fields;

(2) determines when the value corresponding to (alpha) (S1), (alpha) (S2) and (beta) (D) are zero;

(3) obtains initial values for (alpha) (S1), (alpha) (S2) and (beta) (D).

Dispatch stack in claims 16 and 17, which do not specify the number of source registers and which refer to both essential and false dependency fields, is defined as:

An enriched instruction buffer (or DS) including:

(i) A cell for each instruction, each cell having at least the following resultant fields:

(1) an operation field (*e.g.* OP) of the instruction;

(2) a field for each source register (one or more- *e.g.* S1) specified by the instruction;

(3) an essential dependency field (*i.e.* (alpha) (Si), *e.g.* (alpha) (S1)) corresponding to each source register specified by the instruction;

(4) a field for each destination register (one or more- *e.g.* D) specified by the instruction;

(5) a false dependency field (*i.e.* (beta) (D)) corresponding to the destination register specified by the instruction; and

(ii) Logic that

(1) decrements a value >0 in the essential and false dependency fields;

(2) determines when the value corresponding to (alpha) (Si) and (beta) (D) is zero; and

(3) obtains an initial value for (alpha) (Si) and (beta) (D).

Defendant argues that the term dispatch stack has only one meaning throughout all 19 claims, as follows:

[A] modified first-in, first-out (FIFO) stack containing multiple rows with capabilities to allow it to examine and issue one or more instructions for each machine cycle. Each DS row is implemented with registers and contains the following fields:

Instruction tag, OP, S1, (alpha) (S1), S2, (alpha) (S2), D, (beta) (D), and I ^{2}, where:

-> Instruction tag identifies the instruction in the row;

-> OP denotes the arithmetic/logic operation to be performed;

-> S1 specifies the register that provides the first of two operands, or the only operand called for;

-> (alpha) (S1) counts the number of times the register specified by S1 is used as a destination register in preceding, uncompleted instructions;

-> S2 specifies the register that provides the second of the two operands required;

-> (alpha) (S2) counts the number of times the register specified by S2 is used as a destination register in preceding, uncompleted instructions;

-> D denotes the register that receives the result of the arithmetic/logic operation;

-> (beta) (D) counts the number of times the register specified by D is used as a source register in preceding, uncompleted instructions; and

-> I ^{2} is the Issue Index counter, which is the sum of the (alpha) (S1), (alpha) (S2) and (beta) (D) counters.

Defendant's proposed construction runs afoul of the well-established rule of construction that particular embodiments appearing in the specification should not be read into the claims. *See* Texas Digital, 308 F.3d at 1203. Indeed, in urging that the dispatch stack in every claim of the 115 patent teaches all of these limitations, defendant relies principally on the preferred embodiment section of the patent; a number of the limitations defendant lists are not found in all of the claims. Thus, there is no basis to find that every claim of the 115 patent teaches all of these limitations.

Defendant emphasizes the principle that the claim terms must be interpreted consistently throughout the patent claims, *see* Knorr-Bremse Systeme Fuer Nutzfahrzeuge GmbH v. Dana Corp., 133 F.Supp.2d 833 at 838 (E.D.Va.2001), and characterizes plaintiff's proposed construction as "constru[ing] 'dispatch stack' five different ways." The Court finds, however, not that plaintiff's proposed construction comprises a number of inconsistent definitions, but rather that it demonstrates the manner in which the dispatch stack functions in a variety of environments.

In arguing that the term dispatch stack must be construed as having a single construction throughout all 19 claims, defendant further argues that if the Court were to apply the doctrine of claim differentiation among the claims, the result would be an improper modification of the term dispatch stack. The Court finds, to the contrary, that when the term dispatch stack is construed as proposed by plaintiff, the application of the doctrine of claim differentiation strongly supports plaintiff's proposed reading of dispatch stack as simply meaning an enriched instruction buffer and further supports plaintiff's contention that the patent teaches an enriched instruction buffer having additional fields which hold values that track any dependencies associated with the instruction format, and the accompanying logic. For example, independent claims 1, 14 and 15 broadly teach an invention which detects instructions which are dependency free; the dependent claims specify limitations relative to particular instruction formats and particular types of dependencies. Similarly, independent claim 6 broadly teaches an invention which detects OP, S1, S2, D instructions which are dependency free; the dependent claims specify limitations relative to particular types of dependencies.

With respect to the means-plus-function claims (independent claims 1 and 14 and their dependent claims), defendant further argues that the application of the doctrine of claim differentiation results in a conflict with the means-plus-function rule codified by 35 U.S.C. s. 112(6). Defendant argues that plaintiff's proposed construction amounts to an improper attempt to use the doctrine of claim differentiation to override the statutory mandate of s. 112(6). *See* Wenger, 239 F.3d at 1233. The Court finds no conflict because it finds that the application to these claims of the means-plus-function rule of section 112(6) supports plaintiff's construction; this issue is discussed below in the section headed: "Means for detecting the existence of concurrenciesin said instructions received from said memory (claim 1); means for detecting the existence of a plurality of instructions received from said memory which are concurrently executable (claim 14)."

Defendant also relies on the inventor's remarks accompanying the December 19, 1986 amendments to the patent in response to the examiner's September 18, 1986 rejection of claims 1-13 (page 7) based on two prior art references, the Tomasulo article and the Keller article (R.M. Tomasulo, *An Efficient Algorithm for Exploiting Multiple Arithmetic Units,* IBM Journal, Jan. 1967; R.M. Keller, *Look-Ahead Processors,* Computing Surveys, Dec. 1975). According to defendant, these remarks demonstrate that in distinguishing the prior art, the inventor disclaimed a dispatch stack which does not have a false dependency field. The Court rejects this contention. A review of the articles, the examiner's rejection and the inventor's remarks demonstrates that in distinguishing the Tomasulo and Keller prior art, the inventor did *not* distinguish the specific *types* of dependencies which would be eliminated or detected by the prior art references or the 115 invention. Rather, the inventor distinguished the Tomasulo and Keller prior art on the ground that they did not teach a technique for multiple and non-sequential issuance of dependency-free instructions.

The Court's rejection of defendant's contention that the 115 patent is limited to a dispatch stack which contains a false dependency field is reinforced by evidence in the record concerning a technique known as "register renaming." Both experts agreed and the parties do not dispute that as of 1983 it was known in the field (1) that register renaming can eliminate all false dependencies before the instruction enters the dispatch stack and (2) that register renaming always results in a (beta) (D) count of zero. These two assertions are fully consistent with the evidence in the record, including the Keller and Tomasulo publications cited in the patent (R.M. Keller, *Look-Ahead Processors,* Computing Surveys, Dec. 1975; R.M. Tomasulo, *An Efficient Algorithm for Exploiting Multiple Arithmetic Units,* IBM Journal, Jan. 1967, and an accompanying article, D.W. Anderson, F.J. Sparacio and R.M. Tomasulo, *The IBM System/360 Model 91: Machine Philosophy and*

Instruction Handling, IBM Journal, Jan. 1967), and the Tjaden-Flynn and Riseman-Foster articles discussed by the experts (Garold S. Tjaden and Michael J. Flynn, *Detection and Parallel Execution of Independent Instructions,* IEEE Transactions on Computers, Oct. 1970; Edward M. Riseman and Caxton C. Foster, *The Inhibition of Potential Parallelism by Conditional Jumps,* IEEE Transactions on Computers, Dec. 1972). FN12 Accordingly, the Court finds that, as of 1983 and thereafter, a person of skill in the field would have known that register renaming could eliminate all false dependencies before the instruction entered the dispatch stack and that register renaming always results in a (beta) (D) count of zero. The Court further adopts Dr. Smith's view that a person of skill in the art as of the time of the patent would not view a false dependency field ((beta) (D)) as a necessary feature of the invention, because it would not be required for the invention to work in a processor that used register renaming.

FN12. Defendant points out that the term "register renaming" does not appear in the 115 patent, nor did it appear in the articles cited. Defendant does, however, "agree that portions of each of these references not mentioned or discussed in the 115 patent do in a broad sense mention technology which today may generally fit the description of register renaming." Defendant thus does not dispute-and the record supports-the conclusion that the technique of register renaming was known as of the time of the 115 patent.

The Court rejects defendant's contention that plaintiff is improperly attempting to broaden the scope of the claims to include an undisclosed method of eliminating dependencies (*i.e.,* register renaming). To the contrary, the evidence regarding register renaming supports plaintiff's position that the 115 patent teaches an invention which can be used in a variety of different environments, including environments in which there are no false dependencies, whether because they were eliminated by register renaming or for some other reason. The Court also rejects Dr. Flynn's position that the patent requires that the dispatch stack have a field for (beta) (D) even where there are no false dependencies; this conclusion is not supported by the patent, the prior art or Dr. Flynn's testimony on cross-examination.FN13

FN13. Dr. Flynn was questioned extensively on cross-examination regarding his position that the patent requires that the dispatch stack have a field for (beta) (D) even where, due to register renaming or some other technique, there are no false dependencies. He does not articulate any reason why a (beta) (D) field would be needed in such a case, other than to repeat his interpretation of the language of the patent as requiring it. Of course, it is ultimately for the Court, not the experts, to construe the patent. *See* Markman, 517 U.S. at 388, 116 S.Ct. 1384. A representative segment of the cross-examination follows:
Q. Dr. Flynn, isn't it true that if one uses register renaming or some other technique to eliminate all false data dependencies and computer instructions before those instructions have completed their entry into the dispatch stack, then there's no need to detect false data dependencies?

A. Well, one might think that. But the patent deals with this issue explicitly, and as I say, [in] ... my deposition, the patent teaches that if you have a renamed machine, in this case the renaming is occurring after the instructions leave the issue mechanism, whatever it is, that you should still keep, determine a beta value and if that beta value is nonzero, keep track of that in the dispatch stack. I think, as I said before, the implicit message here is that the patent is suggesting to us that renaming wastes the extra registers. Renaming costs something, it's not for free, you have to have something to hold these registers in, and in the Model 91 case, it's reservation stations. In the renaming that Professor Smith was talking about, it's these extra registers.

Q. Dr. Flynn, if all the false data dependencies in the computer instructions have been eliminated before those instructions go into the dispatch stack, then they don't exist anymore, do they?

A. No, they don't.

Q. And so there's no need to detect them because they don't exist, right?

A. Sir, the patent says-go to column 6 of the patent and you'll follow with me, the patent says in the Model 91 case, that you should still keep a record of the beta value and you should not issue the instructions or not execute them because a nonzero beta value, and my point is that the patent is telling you not to do this.

The Court notes also that there is no basis to find that the instruction tags are a structure, or indeed that they are anything other than identifiers to assist the reader in reviewing the charts in the patent, as plaintiff points out. Likewise, the independent claims do not require a dispatch stack which operates in a particular first-in first-out ("FIFO") or modified FIFO mode. Again, the doctrine of claim differentiation supports this conclusion; dependent claim 2 specifies a dispatch stack operating in a first-in first-out manner, while independent claim 1 does not.

Finally, plaintiff's proposed construction of dispatch stack is supported by the problem identified in the patent and prior art (*i.e.,* the unavailability of multiple, non-sequential instruction issuance) and the inventor's solution to the problem (*i.e.,* enriching the instruction buffer with a novel technique for detecting dependency free computer instructions, thereby enabling simultaneousissuance of multiple, non-sequential instructions). Neither the patent nor the file history requires that this technique be limited in the manner defendant proposes. Accordingly, the Court adopts plaintiff's proposed construction of the term dispatch stack in its entirety.

Precedence Count Memory ("PCM")

[21] Defendant defines the PCM as "a table with (alpha)-field and (beta)-field entries for each general purpose register in the execution unit, where the (alpha)-field indicates the number of times that a specific register has been used as a destination register by instructions already in the dispatch stack, and the (beta)-field denotes the number of times that a specific register has been used as a source register by instructions already in the dispatch stack." Defendant does not contend that the PCM is a limitation in the independent method claims 6 and 15. Defendant contends, however, that it is a necessary structure corresponding to the function of detecting in independent means-plus-function claims 1 and 14.

Plaintiff explains that the PCM facilitates the detecting function. Because the PCM makes the detecting function more efficient, the patent includes a PCM in the preferred embodiment of the invention. According to plaintiff, however, the PCM is not a limitation in any of the independent claims.

The Court agrees with plaintiff that the PCM is not a necessary structure in claims 1 and 14. First, the preferred embodiment expressly states that the detecting task "can be *facilitated* with the introduction of the precedence count memory" (emphasis added); the use of the word "facilitated" (the Oxford English Dictionary Online defines "facilitate" as "[t]o render easier the performance of [an action]") supports plaintiff's view that a PCM is helpful but not essential to the detecting function. The conclusion that the PCM is not essential to the detecting task is bolstered by the undisputed construction of independent claims 6 and 15 as not having a PCM.

Next, a reading of independent claim 1 in the light of its dependent claims 3 and 4 demonstrates that the function of detecting in claim 1 is performed by a structure which does not include a PCM. (Claim 14, which in most respects is identical to claim 1, is properly read the same way with respect to the PCM.) Claim 1 does not include the term PCM, nor does it describe the PCM. The "precedent count memory" is added and described in dependent claim 3, and the term is used again in dependent claim 4. Therefore, application of the doctrine of claim differentiation supports plaintiff's construction of claim 1 as not requiring a PCM.

Further, plaintiff's construction of claim 1 is supported by the prosecution history of the 115 patent. In particular, on page 5 of the examiner's statement of January 7, 1986, the examiner stated: "In claim 3, the means for detecting must be stated as ' *further* comprising' to add a new limitation to the preceding claim[.]" (Emphasis added.) In addition, on page 5 of the examiner's statement of September 18, 1986, he rejected proposed wording of claim 3 on the basis that "[t]he connection and relation between the PCM and the previously claimed means must be stated for a functionally clear and complete system." These comments support the view that claim 3 adds the PCM to the "means for detecting" described in claim 1 and that therefore claim 1 does not include the PCM.

Indeed, plaintiff's expert Dr. Smith testified that the detecting function can be performed by the dispatch stack without a PCM by using the full comparison method, in which the entire dispatch stack is scanned for dependencies after every cycle. Defendant's expert Dr. Flynn did not dispute this assertion; rather, Dr. Flynn's testimony that the PCM was a required structure was based on his construction of the patent claims in light of the preferred embodiment. The Court rejects Dr. Flynn's view and concludes that the PCM is a structure which facilitates the function of detecting but is not necessary to the detecting function.FN14

FN14. Defendant also contends that the PCM always contains both (alpha)-field and (beta)-field entries. In support of this contention, defendant points out that in the preferred embodiment and in claims 3 and 4 the patent describes the PCM as having both (alpha) and (beta) fields. A review of the entire structure of the patent, however, supports the common-sense conclusion that the PCM would not have a false dependency field if it were used in a dispatch stack having no false dependencies. In any event, inasmuch as the Court finds that the PCM is not a necessary structure in the invention, a determination of the specific fields of the PCM in all environments is not essential to construing the disputed claims.

Means for detecting the existence of concurrencies in said instructions received from said memory (claim 1); means for detecting the existence of a plurality of instructions received from said memory

which are concurrently executable (claim 14)

[22] Defendant defines these terms as follows:

A PCM, DS, and Reservation Circuit, or an equivalent structure to track counts for (alpha) and (beta) dependencies, sum the counts in another counter, and determine functional unit availability, where Reservation Circuit is defined as a component that has OP, Issue Index and Functional Unit Status inputs, and an Issue output.

Plaintiff defines them as follows:

A means to determine the existence of a plurality of dependency free instructions; the structure corresponding to this recited function is as follows:

An enriched instruction buffer (or DS) including:

(i) A cell for each instruction, each cell having at least the following resultant fields:

(1) an operation field (e.g. OP) of the instruction;

(2) a field for each source register (one or more-e.g. S1) specified by the instruction;

(3) an essential dependency field (i.e.(alpha) (Si), e.g. (alpha) (S1)) corresponding to each source register specified by the instruction;

(4) a field for each destination register (one or more- *e.g.* D) specified by the instruction; and

(ii) Logic that

(1) decrements a value >0 in the essential dependency field;

(2) determines when the value corresponding to (alpha) (Si) is zero; and

(3) obtains an initial value for (alpha) (Si).

The parties agree and the Court finds that claims 1 and 14 are in a "means-plus-function" format. Thus, in construing these claims, the Court's first task is to identify the claimed function; next, it must determine what structures correspond to the "means" for performing that function. *See* 35 U.S.C. s. 112(6); Chiuminatta Concrete, 145 F.3d at 1307-08. In identifying the claimed function for the terms in issue, the Court notes that both claims 1 and 14 clearly teach two distinct functions-a detecting function and an issuing function-and that the terms presently under discussion pertain only to the detecting function.

Plaintiff urges that the detecting function in claims 1 and 14 is "determining the existence of a plurality of instructions that are ready to be issued at the same time because they are data dependency free." Plaintiff further urges that the structure corresponding to this function is the dispatch stack.

Defendant argues that the detecting function in claims 1 and 14 necessarily includes detection of (1) instructions without dependencies and (2) availability of functional units for those instructions. Therefore, defendant argues, the structure corresponding to this function comprises the dispatch stack, reservation circuit and PCM.

The Court rejects defendant's position. Considering first the claimed function, the Court observes that it has already construed the terms "concurrencies" and "plurality of instructions which are concurrently executable" as a "plurality of instructions that are ready to be executed at the same time because they are dependency free." Thus, the function claimed is "detecting a plurality of instructions that are ready to be executed at the same time because they are dependency free." As plaintiff contends, the structure corresponding to this detecting function is the dispatch stack having the characteristics set forth in plaintiff's proposed construction and in the Court's construction of dispatch stack for claims 1 and 14, above. Accordingly, the Court agrees with plaintiff's proposed construction.

In rejecting defendant's position, the Court again emphasizes that both claims 1 and 14 clearly teach two distinct functions: a detecting function and an issuing function. Defendant's proposed structure corresponding to the detecting function improperly includes structure which corresponds to the issuing function. It is well established that a court may not import structural limitations from the written description that are unnecessary to perform the claimed function. *See* Wenger, 239 F.3d at 1233. Here, the reservation circuit is not a corresponding structure with respect to the detecting function in claims 1 and 14; rather, it is the structure which corresponds to the issuing function. This may be seen from its description in the "Preferred Embodiment" section of the patent: "When an instruction with an I^2 value of 0 is encountered, the issuing mechanism reserves an appropriate functional unit if available and then issues the instruction to it. The implementation of this search and issue operation is in the form of a reservation circuit[.]" Nothing in the patent supports defendant's position that the reservation circuit is involved in the function of detecting dependency free instructions. A related flaw in defendant's proposed construction is that the issue index (I^2) is not part of the reservation circuit, which corresponds to an issuing function; rather, I^2 is a value reflecting the sum of dependencies in a certain instruction and thus I^2 corresponds to the detecting function.

Further, as discussed above, the PCM is not a necessary structure corresponding to the detecting function in claims 1 or 14. Rather, it is added in certain dependent claims to facilitate the detecting function. Thus, the dispatch stack is the only structure necessarily performing the function of detecting concurrencies and concurrently executable instructions.

In view of the Court's conclusion that the dispatch stack is the only structure necessarily corresponding to the detecting function in claims 1 and 14, there is no conflict between compliance with the statutory requirements of section 112(6) and application of the doctrine of claim differentiation, and no need to resort to the principle that where such a conflict exists, the statutory requirements prevail. *See id.*

Accordingly, in claims 1 and 14, the Court construes the detecting function as "determining the existence of a plurality of instructions that are ready to be issued at the same time because they are data dependencyfree," and the structure corresponding to this function as the dispatch stack as proposed by plaintiff.

Means for issuing multiple instructions and non-sequential instructions to said execution unit within a single processor cycle when a concurrency is detected by said means for detecting the existence of concurrencies in said instructions (claim 1); means for issuing multiple instructions and non-

sequential instructions to said execution unit within a single processor cycle when concurrently executable instructions are detected by said means for detecting the existence of concurrently executable instructions in said instructions (claim 14)

[23] Again, it is undisputed that claims 1 and 14 are in a "means-plus-function" format; accordingly, the Court first determines what the claimed function is; then it determines what structures correspond to the "means" for performing that function.

Defendant defines the issuing function as: "Issuing multiple and non-sequential instructions when concurrently executable instructions are detected." Defendant defines the corresponding structure as: "A reservation circuit, or an equivalent structure to issue multiple and non-sequential, concurrently executable instructions to functional units, where a reservation circuit is defined as a component that has OP, Issue Index and Functional Unit Status inputs, and an Issue output."

Plaintiff defines the function as: "To issue multiple and non-sequential (*i.e.,* out-of-order) instructions within a single processor cycle that have become dependency free." According to plaintiff, the structure is a reservation circuit.

There is no apparent difference between the two proposed constructions of the issuing function, provided that the phrase "concurrently executable instructions" in defendant's proposed construction is construed as meaning the same as "dependency free instructions" in accordance with the Court's above construction of "concurrencies" and "plurality of instructions which are concurrently executable," *i.e.,* provided that the phrase "concurrently executable instructions" is not read as specifically requiring that the instructions be free of false dependencies, as requiring that false dependencies have previously existed and been eliminated, or as requiring that false dependencies have been removed in any particular manner.

With respect to the structure, defendant's proposed construction of reservation circuit includes inappropriate features, *i.e.,* issue index (I^2), OP and functional unit status. As noted, the Court may not import structural limitations that are unnecessary to perform the claimed function. *See* Wenger, 239 F.3d at 1233. With respect to I^2, the Court finds that it is not a necessary structure in the reservation circuit. The patent does not require I^2 to be a structure, nor is it necessary to the issuing function. As Dr. Smith explained, in the absence of an I^2 counter, the reservation circuit simply receives individual signals from the dispatch stack that the value of zero has been encountered for the dependency fields; the reservation circuit then issues the instruction as long as there are sufficient available functional units. This conclusion is bolstered by application of the doctrine of claim differentiation; the dependent means-plus-function claim 5 adds an I^2, thus differentiating it from independent claim 1 and dependent claims 2 and 3 (from which it depends). Moreover, as stated above, I^2 represents an aspect of the detecting function, not of the issuing function, and thus would not in any event be associated with the reservation circuit.

OP, or operation code, is a value describing the type of operation required to carry out a particular instruction; thus, it describesthe type of functional unit requested. The functional unit status is a value that indicates whether a functional unit is available. Nothing in the patent suggests that these features constitute necessary structures.

Dr. Smith's explanation of the reservation circuit as comprising conventional arbitration logic to keep track of available functional units and to issue dependency free instructions to available functional units is fully consistent with the patent. Nothing in the patent requires that the instructions which are issued by the reservation circuit must be free of any particular type of dependency, that any particular type of dependency have previously existed, or that dependencies have been removed in any particular manner.

The Court finds that the instruction issuing function in claims 1 and 14 is "issuing multiple and non-sequential (*i.e.,* out-of-order) instructions within a single processor cycle that have become dependency free," and that the corresponding structure is a reservation circuit.

Detecting the existence of concurrencies in instructions stored in said dispatch stack (claim 6); detecting the existence of a plurality of instructions which are concurrently executable from those instructions stored in said dispatch stack (claim 15)

[24] The parties agree that these are "method" claims. Defendant defines the terms as follows:

Scanning the Dispatch Stack from top to bottom to identify instructions for which the sum of all (alpha) and (beta) dependencies, called the Issue Index (I ^{2}), is determined to be equal to zero and verifying that the required functional unit for each identified instruction is available using a reservation circuit, where reservation circuit is defined as a component that has OP, Issue Index, and Functional Unit Status inputs, and an Issue output.

Defendant's construction includes I ^{2}, and, because defendant defines I ^{2} as counting both essential and false dependencies, defendant's construction includes false dependencies as well as essential dependencies.

The Court agrees with plaintiff that the two terms under consideration, "detecting the existence of concurrencies in instructions stored in said dispatch stack" and "detecting the existence of a plurality of instructions which are concurrently executable from those instructions stored in said dispatch stack," simply mean "determining the existence of a plurality of dependency free instructions stored in the dispatch stack." The Court further agrees that nothing in the patent requires the inclusion of I ^{2} and/or false dependencies in the definition of these two terms. As discussed above, I ^{2} is not necessarily a counter, but rather a computed value derived from the values (alpha) (Si) (essential dependencies), and, if false dependencies are being tracked, (beta) (D) as well.

Further, for the reasons set forth above in the discussion of the phrase "concurrently executable instructions," the Court finds that the two terms do not require that the instructions be free of any particular type of dependency or that any particular type of dependency have previously existed and been removed. Therefore, there is no basis to construe the terms as teaching a dispatch stack which necessarily has a field for false dependencies. The doctrine of claim differentiation supports the conclusion that false dependencies are not a limitation in claims 6 and 15; false dependencies, while not limitations in independent claims 6 and 15, are specified as limitations in dependent claims 7 and 16.

Moreover, defendant improperly reads a reservation circuit into the detecting function. As discussed above in connection with claims 1 and 14, the reservation circuitserves an issuing function, not a detecting function. Thus, the Court rejects defendant's position and reads the terms as meaning "determining the existence of a plurality of dependency free instructions stored in the dispatch stack."

Issuing multiple instructions and non-sequential instructions within a given processor cycle when the

existence of concurrencies is detected (claim 6), and issuing multiple instructions and non-sequential instructions within a given processor cycle when said plurality of concurrently executable instructions are detected (claim 15)

[25] Defendant construes these terms as meaning: "forwarding by the reservation circuit of all instructions for which the Issue Index (I ^{2}) is equal to zero and the requisite functional unit available where Reservation circuit is defined as a component that has OP, Issue Index, and Functional Unit Status inputs, and an Issue output." Thus, I ^{2} (which according to defendant includes false dependencies) is an integral part of defendant's construction.

Plaintiff contends that the terms simply mean "issuing multiple and non-sequential instructions when the dispatch stack has detected a plurality of concurrently executable (*i.e.* data dependency free) instructions." The Court has already determined that there is no requirement that the dispatch stack include an I ^{2}. Further, the Court has already found that false dependencies are not a limitation in claims 6 and 15. Based in particular on its above reading of the phrases "detecting the existence of concurrencies in instructions stored in said dispatch stack" (claim 6) and "detecting the existence of a plurality of instructions which are concurrently executable from those instructions stored in said dispatch stack" (claim 15), the Court agrees with plaintiff. The Court adopts plaintiff's proposed construction.

Providing an indication of the determination (claims 7 and 16)

[26] The Court adopts plaintiff's construction of the phrase "providing an indication of the determination" in dependent claims 7 and 16 as meaning "providing initial values for (alpha) (S1), (alpha) (S2) and (beta) (D) in the dispatch stack." These two claims, which specify false dependency fields, depend from independent claims 6 and 15, which do not specify such fields. This reading appropriately differentiates claims 7 and 16 from the claims from which they depend.

Appropriately decrementing the values of (alpha)'s, the values of (beta)'s (claim 12)

[27] The Court adopts plaintiff's proposed construction of the phrase "appropriately decrementing the values of (alpha)'s, the values of (beta)'s" as meaning: "subtracting one from the (alpha) (S1) and/or (alpha) (S2) counts in the dispatch stack and subtracting one or two from the (beta) (D) count in the dispatch stack." This reading is consistent with the intrinsic evidence and appropriately differentiates dependent claim 12 from the independent claim 6 from which it depends.

Instruction does not have any data dependencies with preceding issued instructions which have not yet been completed (claims 8 and 17)

[28] Defendant argues that claims 8 and 17 are limited to "data dependencies" which include both essential and false dependencies. Defendant relies on the facts that claim 8 depends from claim 7, and claim 7 specifies both essential and false dependencies. Therefore, claim 8 must include both as well. The same argument applies to claim 17, which depends from claim 16.

Plaintiff's position on this issue is not clear. In its proposed Findings of Fact and Conclusions of Law plaintiff proposes that "data dependencies" in claims 8 and 17 be construed as meaning "an instruction that depends for its issuance on a value in another instruction." In this regard, plaintiff argues that defendant improperly imports into claims 8 and 17 the requirement of both essential and false dependencies. Plaintiff's argument is supported by application of the doctrine of claim differentiation to claims 8 and 17 and the claims which depend from them (claims 9, 10, 18 and 19). Plaintiff's proposed construction of dispatch stack for claims 8 and 17, however, includes both essential and false dependencies.

The Court is aware that claim differentiation is not a rigid rule, *see* Comark, 156 F.3d at 1187, and, in light of the fact that claims 8 and 17 depend from claims which recite both essential and false dependencies, the Court adopts defendant's construction. Accordingly, the Court construes the phrase "instruction does not have any data dependencies with preceding issued instructions which have not yet been completed" in claims 8 and 17 as meaning "an instruction that is free of (alpha) and (beta) dependencies."

CONCLUSION

It is therefore

ORDERED that plaintiff's motion to strike all references to and discussion of the book, *Superscalar Microprocessor Design,* by Mike Johnson of Advanced Micro Devices (Prentice-Hall, Inc.1991) and all testimony based thereon is **GRANTED**; and it is further

ORDERED that the disputed terms of the 115 patent are construed in accordance with the decision herein as follows:

Instruction (claims 1, 14 and 15):

An expression that specifies one or more operations and identifies the applicable operands.

Instruction (claim 6):

An expression that has a specific format (*i.e.,* OP, S1, S2, D).

Concurrencies (claims 1, 6); **a plurality of instructions which are concurrently executable** (claims 14, 15):

A plurality of instructions that are ready to be issued because they are dependency free.

Dispatch stack:

Claims 1, 14 and 15:

[A]n enriched instruction buffer (or DS) including:

(i) A cell for each instruction, each cell having at least the following resultant fields:

(1) an operation field (*e.g.* OP) of the instruction;

(2) a field for each source register (one or more- *e.g.* S1) specified by the instruction;

(3) an essential dependency field (*i.e.* (alpha) (Si), *e.g.* (alpha) (S1)) corresponding to each source register

specified by the instruction;

(4) a field for each destination register (one or more- *e.g.* D) specified by the instruction; and

(ii) Logic that

(1) decrements a value >0 in the essential dependency field;

(2) determines when the value corresponding to (alpha) (Si) is zero; and

(3) obtains an initial value for (alpha) (Si).

Claims 2 and 6:

[A]n enriched instruction buffer (or DS) including:

(i) A cell for each instruction, each cell having at least the following resultant fields:

(1) an operation field (*e.g.* OP) of the instruction;

(2) a field for each of two source registers (*e.g.* S1 and S2) specified by the instruction;

(3) two essential dependency fields (*i.e.* (alpha) (S1) and (alpha) (S2)) corresponding to the two source registers (*e.g.* S1 and S2) specified by the instruction;

(4) a field for each destination register (one or more- *e.g.* D) specified by the instruction; and

(ii) Logic that

(1) decrements a value >0 in each essential dependency field;

(2) determines when the values corresponding to (alpha) (S1) and (alpha) (S2) are zero; and

(3) obtains initial values for (alpha) (S1) and (alpha) (S2).

Claims 3 and 4:

An enriched instruction buffer (or DS) including:

(i) A cell for each instruction, each cell having at least the following resultant fields:

(1) an operation field (*e.g.* OP) of the instruction;

(2) a field for each of two source registers (*i.e.* S1 and S2) specified by the instruction;

(3) two essential dependency fields (*i.e.* (alpha) (S1) and (alpha) (S2)) corresponding to registers S1 and S2 specified by the instruction;

(4) a field for each destination register (one or more- *e.g.* D) specified by the instruction;

(5) a false dependency field (*i.e.* (beta) (D)) corresponding to the destination register specified by instruction; and

(ii) Logic that

(1) decrements a value >0 in the essential and false dependency fields;

(2) determines when the value corresponding to (alpha) (S1), (alpha) (S2) and (beta) (D) are zero;

(3) obtains initial values for (alpha) (S1), (alpha) (S2) and (beta) (D) from the PCM.

Precedence Count Memory ("PCM") in claims 3 and 4:

[A] memory or PCM including:

(i) (alpha) and (beta) fields or entries for each register implemented in the processor, each entry or field ((alpha)) contains a count (*i.e.* a value) that represents the number of instructions in the dispatch stack that will write the register as its destination register, and each entry or field ((beta)) contains a count (*i.e.* a value) that represents the number of instructions in the dispatch stack that are designated as source registers; and

(ii) logic that increments and decrements the respective counts (*i.e.* the respective values) for the (alpha) and (beta) fields or entries.

Claim 5:

An enriched instruction buffer (or DS) including:

(i) A cell for each instruction, each cell having at least the following resultant fields:

(1) an operation field (*e.g.* OP) of the instruction;

(2) a field for each of two source registers (*i.e.* S1 and S2) specified by the instruction;

(3) two essential dependency fields (*i.e.* (alpha) (S1) and (alpha) (S2)) corresponding to registers S1 and S2 specified by the instruction;

(4) a field for each destination register (one or more- *e.g.* D) specified by the instruction;

(5) a false dependency field (*i.e.* (beta) (D)) corresponding to the destination register specified by instruction;

(6) an Issue Index (I^2) field equal to (alpha) (S1)+ (alpha) (S2)+ (beta) (D); and

(ii) Logic that

(1) decrements a value >0 in the essential and false dependency fields;

(2) determines when the values corresponding to (alpha) (S1), (alpha) (S2) and (alpha) (D) are zero;

(3) obtains initial values for (alpha) (S1), (alpha) (S2) and (beta) (D).

Claims 7-13:

An enriched instruction buffer (or DS) including:

(i) A cell for each instruction, each cell having at least the following resultant fields:

(1) an operation field (*e.g.* OP) of the instruction;

(2) a field for each of two source registers (*i.e.* S1 and S2) specified by the instruction;

(3) two essential dependency fields (*i.e.* (alpha) (S1) and (alpha) (S2)) corresponding to registers S1 and S2 specified by the instruction;

(4) a field for each destination register (one or more- *e.g.* D) specified by the instruction;

(5) a false dependency field (*i.e.* (beta) (D)) corresponding to the destination register specified by instruction; and

(ii) Logic that

(1) decrements a value >0 in the essential and false dependency fields;

(2) determines when the value corresponding to (alpha) (S1), (alpha) (S2) and (beta) (D) are zero;

(3) obtains initial values for (alpha) (S1), (alpha) (S2) and (beta) (D).

Claims 16 and 17:

An enriched instruction buffer (or DS) including:

(i) A cell for each instruction, each cell having at least the following resultant fields:

(1) an operation field (*e.g.* OP) of the instruction;

(2) a field for each source register (one or more- *e.g.* S1) specified by the instruction;

(3) an essential dependency field (*i.e.* (alpha) (Si), *e.g.* (alpha) (S1)) corresponding to each source register specified by the instruction;

(4) a field for each destination register (one or more- *e.g.* D) specified by the instruction;

(5) a false dependency field (*i.e.* (beta) (D)) corresponding to the destination register specified by the instruction; and

(ii) Logic that

(1) decrements a value >0 in the essential and false dependency fields;

(2) determines when the value corresponding to (alpha) (Si) and (beta) (D) is zero; and

(3) obtains an initial value for (alpha) (Si) and (beta) (D).

Means for detecting the existence of concurrencies in said instructions received from said memory (claim 1); **means for detecting the existence of a plurality of instructions received from said memory which are concurrently executable** (claim 14):

Detecting function:

Determining the existence of a plurality of instructions that are ready to be issued at the same time because they are data dependency free.

Corresponding structure:

Dispatch stack as construed above for claims 1 and 14.

Means for issuing multiple instructions and non-sequential instructions to said execution unit within a single processor cycle when a concurrency is detected by said means for detecting the existence of concurrencies in said instructions (claim 1); **means for issuing multiple instructions and non-sequentialinstructions to said execution unit within a single processor cycle when concurrently executable instructions are detected by said means for detecting the existence of concurrently executable instructions in said instructions** (claim 14):

Issuing function:

To issue multiple and non-sequential (*i.e.*, out-of-order) instructions within a single processor cycle that have become dependency free.

Corresponding structure:

Reservation circuit.

Detecting the existence of concurrencies in instructions stored in said dispatch stack (claim 6); **detecting the existence of a plurality of instructions which are concurrently executable from those instructions stored in said dispatch stack** (claim 15):

Determining the existence of a plurality of dependency free instructions stored in the dispatch stack.

Issuing multiple instructions and non-sequential instructions within a given processor cycle when the existence of concurrencies is detected (claim 6)**, and issuing multiple instructions and non-sequential instructions within a given processor cycle when said plurality of concurrently executable instructions are detected** (claim 15):

Issuing multiple and non-sequential instructions when the dispatch stack has detected a plurality of concurrently executable (*i.e*. data dependency free) instructions.

Providing an indication of the determination (claims 7 and 16):

Providing initial values for (alpha) (S1), (alpha) (S2) and (beta) (D) in the dispatch stack.

Appropriately decrementing the values of (alpha)'s, the values of (beta)'s (claim 12):

Subtracting one from the (alpha) (S1) and/or (alpha) (S2) counts in the dispatch stack and subtracting one or two from the (beta) (D) count in the dispatch stack.

Instruction does not have any data dependencies with preceding issued instructions which have not yet been completed (claims 8 and 17):

An instruction that is free of (alpha) and (beta) dependencies.

IT IS SO ORDERED.