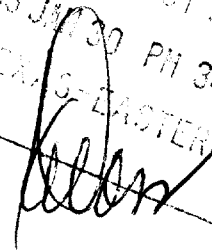


IN THE UNITED STATES DISTRICT COURT
FOR THE EASTERN DISTRICT OF TEXAS
MARSHALL DIVISION

FILED-CLERK
U.S. DISTRICT COURT
03 JAN 30 PM 3:51
TEXAS-EASTERN
BY 

(1) INTERGRAPH HARDWARE
TECHNOLOGIES COMPANY, INC., a
Nevada corporation,

Plaintiff,

vs.

(1) TEXAS INSTRUMENTS
INCORPORATED, a Delaware corporation,

Defendant.

Civil Action No.

2-03C v-034 D7

COMPLAINT FOR PATENT INFRINGEMENT

Plaintiff INTERGRAPH HARDWARE TECHNOLOGIES COMPANY ("IHTC") for its
Complaint against Defendant TEXAS INSTRUMENTS INCORPORATED alleges:

THE PARTIES

1. IHTC is a corporation duly organized and existing under the laws of the State of Nevada, and has a principal place of business at 2325-B Renaissance Drive, Suite 16, Las Vegas, NV 89119. IHTC is a wholly-owned subsidiary of Intergraph Corporation ("Intergraph Corp."), a Delaware corporation with its principle place of business in Huntsville, Alabama.

2. IHTC is informed and believes, and on that basis alleges, that Defendant TEXAS INSTRUMENTS INCORPORATED ("TI") is a Delaware corporation with its principal place of business at 12500 TI Boulevard, Dallas, Texas 75243-4136.

JURISDICTION AND VENUE

3. The court has subject matter jurisdiction pursuant to 28 U.S.C. §§1331 and 1338(a) because this action arises under the patent laws of the United States, 35 U.S.C. §§1 *et seq.* Venue is proper in this federal district pursuant to 28 U.S.C. §§1391(b)-(c) and 1400(b) in that TI has done business in this District, has committed acts of infringement in this District, and continues to commit acts of infringement in this District, entitling IHTC to relief.

INFRINGEMENT OF U.S. PATENT NO. 5,560,028

4. On September 24, 1996, United States Patent No. 5,560,028 (the "'028 patent") was duly and legally issued for an invention entitled "Software Scheduled Superscalar Computer Architecture." IHTC was assigned the '028 patent and IHTC continues to hold all rights and interest in the '028 patent. A true and correct copy of the '028 patent is attached hereto as Exhibit A.

5. Upon information and belief, TI has infringed and continues to infringe the '028 patent. The infringing acts include, but are not limited to, the manufacture, use, sale, importation, and/or offer for sale of Digital Signal Processors ("DSPs") utilizing TI's TMS320C6000 platform, including TI's C62x, C64x and C67x generations, and Application Specific Integrated Circuits ("ASICs") containing such DSPs; and inducement of others to manufacture, use, sell, import, and/or offer for sale of such DSPs and ASICs. TI is liable for infringement of the '028 patent pursuant to 35 U.S.C. § 271.

6. TI's acts of infringement have caused damage to IHTC and IHTC is entitled to recover from TI the damages sustained by IHTC as a result of TI's wrongful acts in an amount subject to proof at trial. TI's infringement of IHTC's exclusive rights under the '028 patent will continue to damage IHTC, causing irreparable harm, for which there is no adequate remedy at law, unless enjoined by this Court.

7. Upon information and belief, TI's infringement of the '028 patent is willful and deliberate, entitling IHTC to increased damages under 35 U.S.C. § 284 and to attorneys' fees and costs incurred in prosecuting this action under 35 U.S.C. § 285.

INFRINGEMENT OF U.S. PATENT NO. 5,794,003

8. On August 11, 1998, United States Patent No. 5,794,003 (the "'003 patent") was duly and legally issued for an invention entitled "Instruction Cache Associative Crossbar Switch System." IHTC was assigned the '003 patent and IHTC continues to hold all rights and interest in the '003 patent. A true and correct copy of the '003 patent is attached hereto as Exhibit B.

9. Upon information and belief, TI has infringed and continues to infringe the '003 patent. The infringing acts include, but are not limited to, the manufacture, use, sale,

importation, and/or offer for sale of DSPs utilizing TI's TMS320C6000 platform, including TI's C62x, C64x and C67x generations, and ASICs containing such DSPs; and inducement of others to manufacture, use, sell, import, and/or offer for sale of such DSPs and ASICs. TI is liable for infringement of the '003 patent pursuant to 35 U.S.C. § 271.

10. TI's acts of infringement have caused damage to IHTC and IHTC is entitled to recover from TI the damages sustained by IHTC as a result of TI's wrongful acts in an amount subject to proof at trial. TI's infringement of IHTC's exclusive rights under the '003 patent will continue to damage IHTC, causing irreparable harm, for which there is no adequate remedy at law, unless enjoined by this Court.

11. Upon information and belief, TI's infringement of the '003 patent is willful and deliberate, entitling IHTC to increased damages under 35 U.S.C. § 284 and to attorneys' fees and costs incurred in prosecuting this action under 35 U.S.C. § 285.

INFRINGEMENT OF U.S. PATENT NO. 6,360,313 B1

12. On March 19, 2002, United States Patent No. 6,360,313 B1 (the "'313 patent") was duly and legally issued for an invention entitled "Instruction Cache Associative Crossbar Switch." IHTC was assigned the '313 patent and IHTC continues to hold all rights and interest in the '313 patent. A true and correct copy of the '313 patent is attached hereto as Exhibit C.

13. Upon information and belief, TI has infringed and continues to infringe the '313 patent. The infringing acts include, but are not limited to, the manufacture, use, sale, importation, and/or offer for sale of DSPs utilizing TI's TMS320C6000 platform, including TI's C62x, C64x and C67x generations, and ASICs containing such DSPs; and inducement of others to manufacture, use, sell, import, and/or offer for sale of such DSPs and ASICs. TI is liable for infringement of the '003 patent pursuant to 35 U.S.C. § 271.

14. TI's acts of infringement have caused damage to IHTC and IHTC is entitled to recover from TI the damages sustained by IHTC as a result of TI's wrongful acts in an amount subject to proof at trial. TI's infringement of IHTC's exclusive rights under the '313 patent will continue to damage IHTC, causing irreparable harm, for which there is no adequate remedy at law, unless enjoined by this Court.

15. Upon information and belief, TI's infringement of the '313 patent is willful and deliberate, entitling IHTC to increased damages under 35 U.S.C. § 284 and to attorneys' fees and costs incurred in prosecuting this action under 35 U.S.C. § 285.

PRAYER FOR RELIEF

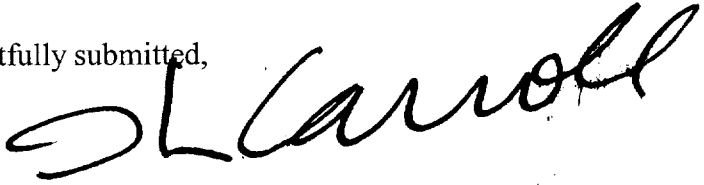
WHEREFORE, Plaintiff Intergraph Hardware Technologies Company requests entry of judgment in its favor and against Texas Instruments Incorporated as follows:

- a) Declaration that Texas Instruments Incorporated has infringed U.S. Patent Nos. 5,560,028, 5,794,003 and 6,360,313 B1;
- b) Permanently enjoining Texas Instruments Incorporated, its officers, agents, employees, and those acting in privity with them, from further infringement, contributory infringement and/or inducing infringement of U.S. Patent Nos. 5,560,028, 5,794,003 and 6,360,313 B1;
- c) Awarding the damages arising out of Texas Instruments Incorporated's infringement of U.S. Patent Nos. 5,560,028, 5,794,003 and 6,360,313 B1, including enhanced damages pursuant to 35 U.S.C. § 284, to Intergraph Hardware Technologies Company, together with prejudgment and post-judgment interest, in an amount according to proof;
- d) An award of attorneys' fees pursuant to 35 U.S.C. § 285 or as otherwise permitted by law; and
- e) For such other costs and further relief as the Court may deem just and proper.

DATED: January 30, 2003

Respectfully submitted,

By:


Franklin Jones Jr.
State Bar No. 00000055
JONES AND JONES, INC., P.C.
201 West Houston Street
P.O. Drawer 1249
Marshall, TX 75671-1249
Telephone: (903) 938-4395
Facsimile: (903) 938-3360
maizieh@millerfirm.com

Otis W. Carroll – Attorney-In-Charge
State Bar No. 03895700
nancy@icklaw.com
IRELAND CARROLL AND KELLEY, P.C.
6101 South Broadway, Suite 500
P.O. Box 7879
Tyler, TX 75711
Telephone: (903) 561-1600
Facsimile: (903) 561-1071

S. Calvin Capshaw
State Bar No. 03783900
ccapshaw@bmoh.com
BROWN McCARROLL LLP
1127 Judson Road, Suite 220,
P.O. Box 3999
Longview, Texas 75601-5157
Telephone: (903) 236-9800
Facsimile: (903) 236-8787

George M. Schwab
CA State Bar No. 058250
gms@townsend.com
K.T. Cherian
CA State Bar No. 133967
ktc@townsend.com
R. Scott Wales
CA State Bar No. 179804
rsw@townsend.com
April E. Abele
aeabele@townsend.com
CA State Bar No. 180638
Gregory S. Bishop
CA State Bar No. 184680
gsbishop@townsend.com
TOWNSEND AND TOWNSEND AND CREW LLP
Two Embarcadero Center, 8th Floor
San Francisco, California 94111
Telephone: (415) 576-0200
Facsimile: (415) 576-0300

David Vance Lucas
General Counsel
INTERGRAPH CORPORATION
Mail Stop IW2008
Huntsville, Alabama 35894-0001
Telephone: (256) 730-2032
Facsimile: (256) 730-2247
dvlucas@ingr.com

Attorneys for Plaintiff
INTERGRAPH CORPORATION

[45] **Date of Patent:** Sep. 24, 1996

EXHIBIT

5,560,028

Page 2

OTHER PUBLICATIONS

Bakoglu, H. B., et al., "The IBM RISC System/6000 Processor: Hardware Overview," *IBM J. Res. Develop.* (Jan. 1990) 34(1):12-22.

Fisher, Joseph A., et al., "Parallel Processing: A Smart Compiler and a Dumb Machine," Proceedings of the ACM SIGPLAN '84 Symposium on Compiler Construction, *SIGPLAN Notices* (Jun. 1984) 19(6):37-47.

Agerwala, Tilak, et al., "High Performance Reduced Instruction Set Processors," IBM Research Report No. 12434 (#55845) (Jan. 9, 1987), IBM Thomas J. Watson Research Center, Yorktown Heights, New York.

Patterson, David A., et al., *Computer Architecture—A Quantitative Approach*, Morgan Kaufmann Publishers, Inc., San Mateo Calif., 1990, Table of Contents, pp. xi-xv.

U.S. Patent

Sep. 24, 1996

Sheet 1 of 11

5,560,028

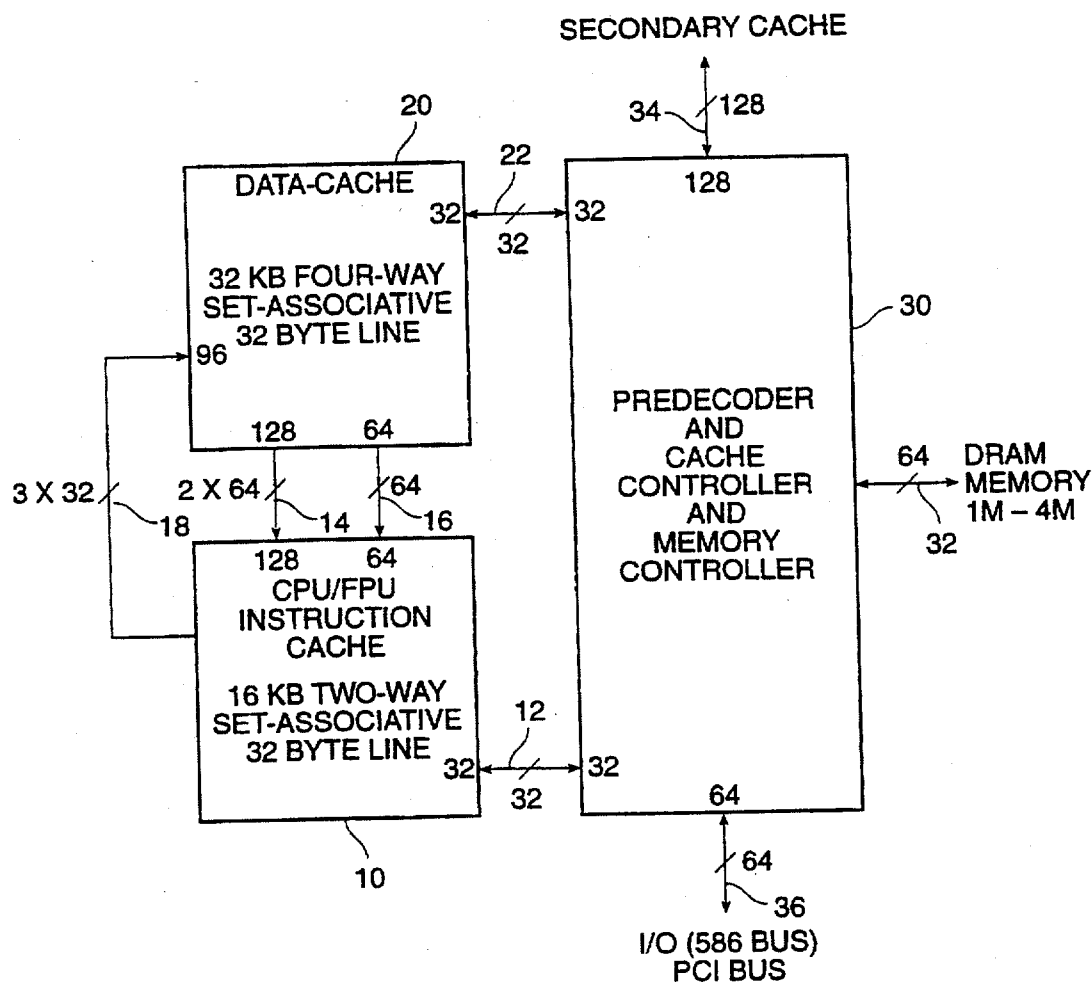


FIG. 1

U.S. Patent

Sep. 24, 1996

Sheet 2 of 11

5,560,028

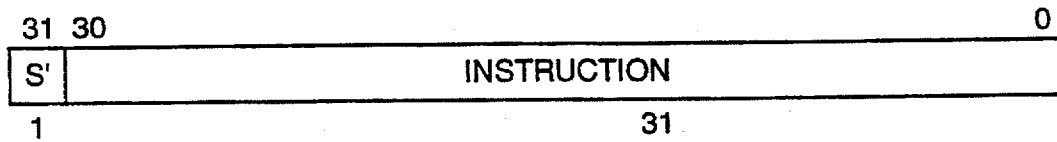


FIG. 2



FIG. 3

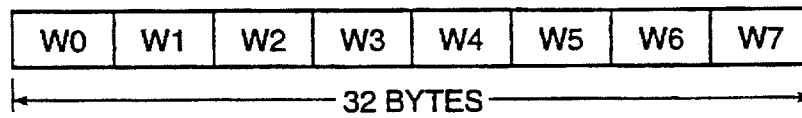


FIG. 4

U.S. Patent

Sep. 24, 1996

Sheet 3 of 11

5,560,028

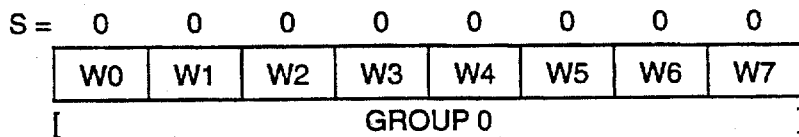


FIG. 5A

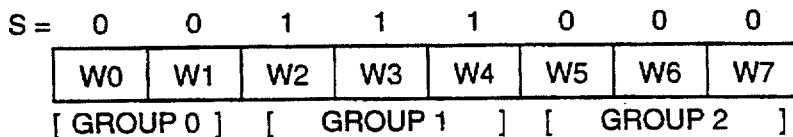


FIG. 5B

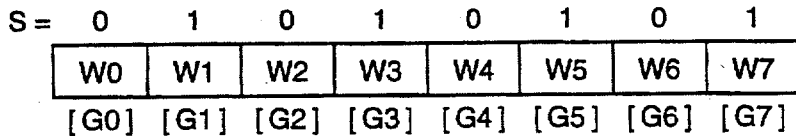


FIG. 5C



FIG. 6

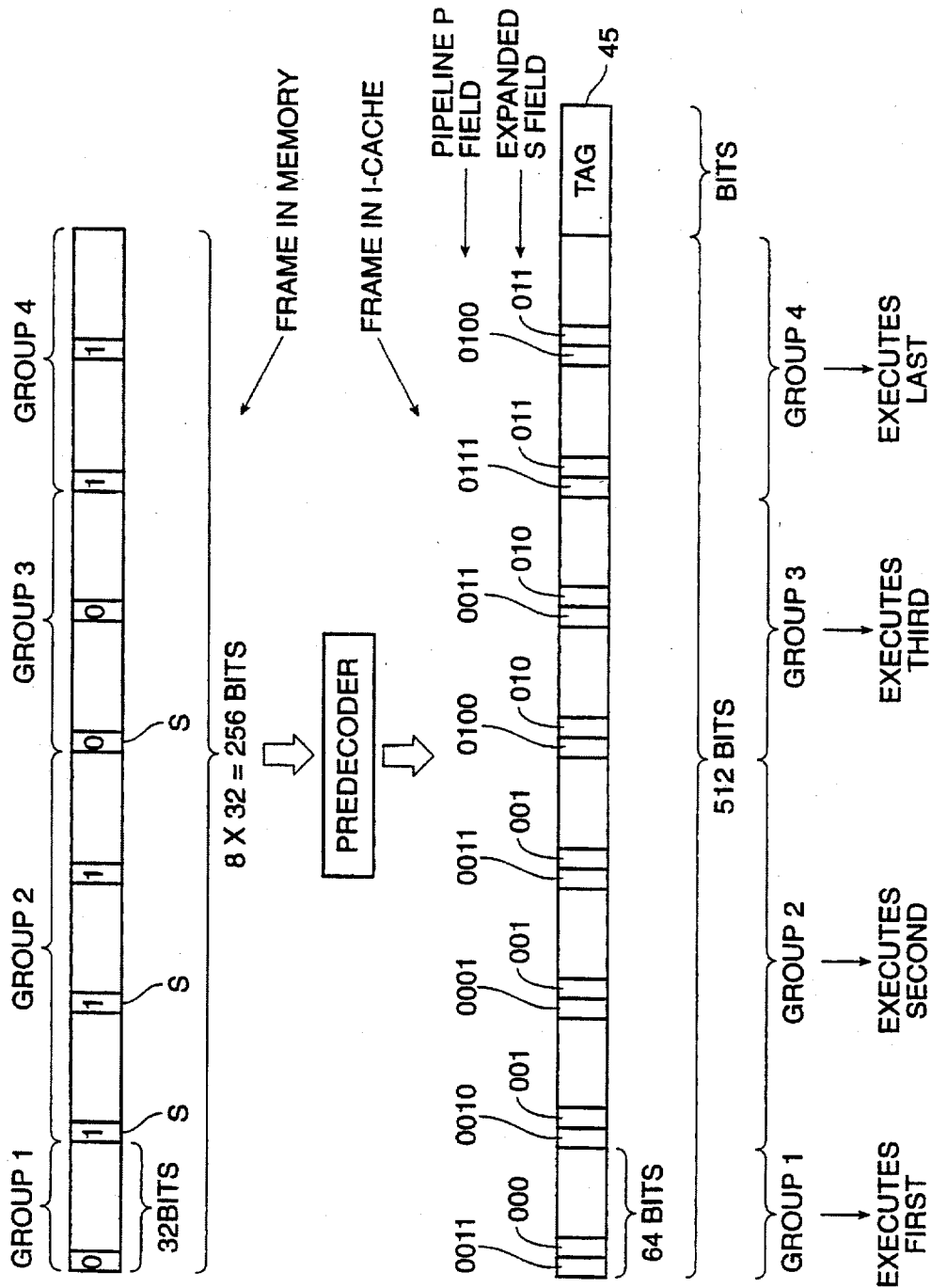


FIG. 7

U.S. Patent

Sep. 24, 1996

Sheet 5 of 11

5,560,028

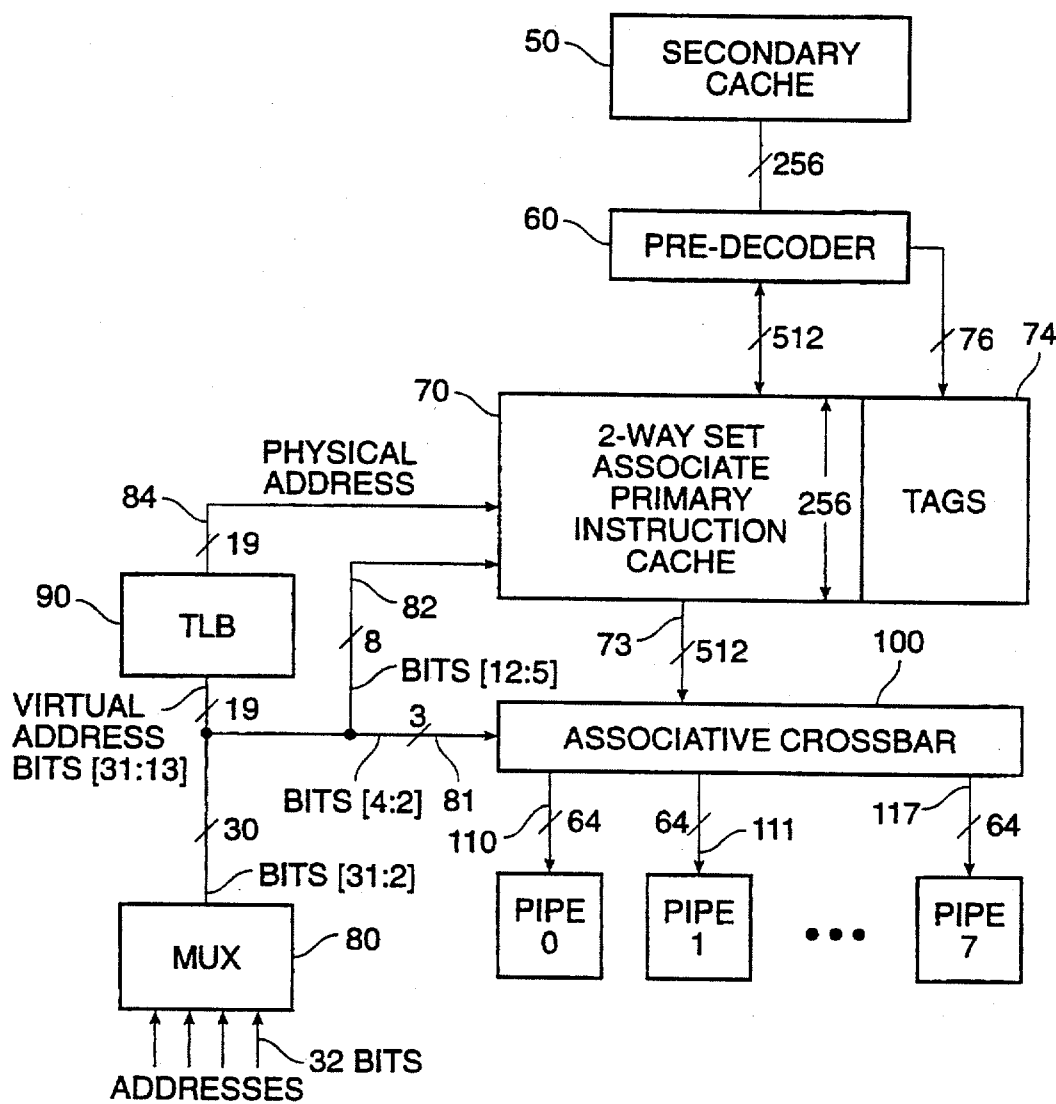


FIG. 8

U.S. Patent

Sep. 24, 1996

Sheet 6 of 11

5,560,028

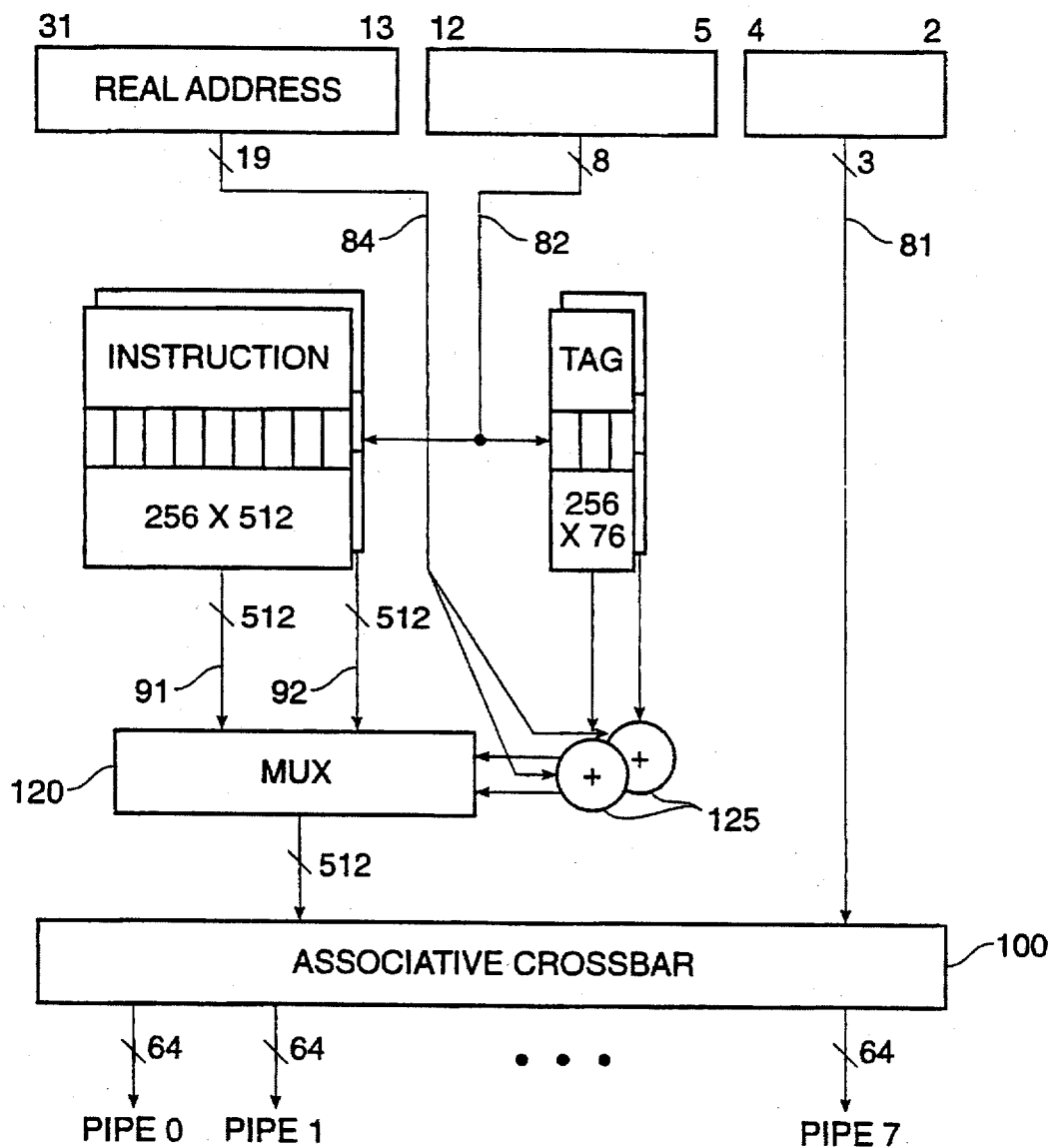


FIG. 9

U.S. Patent

Sep. 24, 1996

Sheet 7 of 11

5,560,028

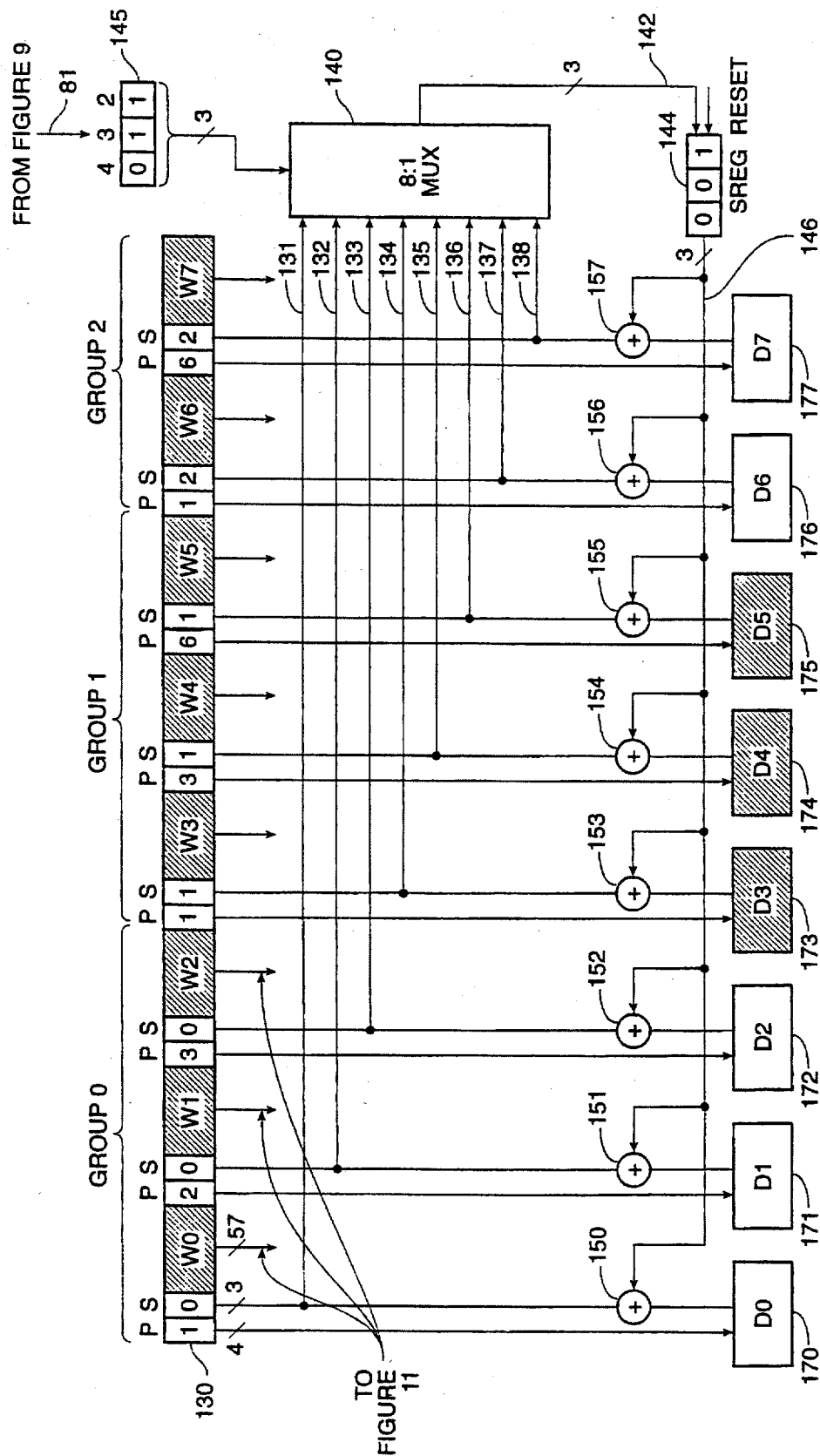


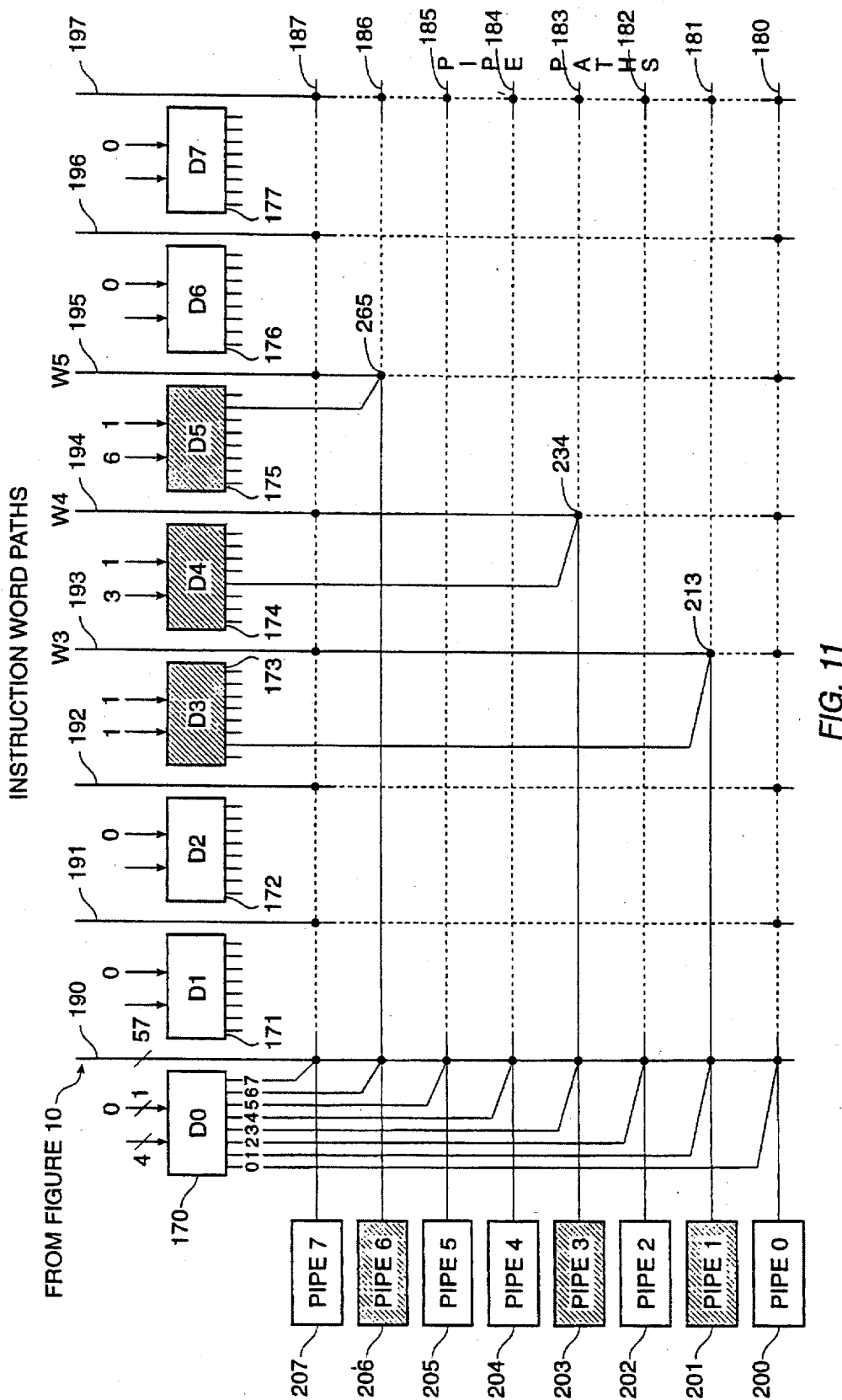
FIG. 10

U.S. Patent

Sep. 24, 1996

Sheet 8 of 11

5,560,028



U.S. Patent

Sep. 24, 1996

Sheet 9 of 11

5,560,028

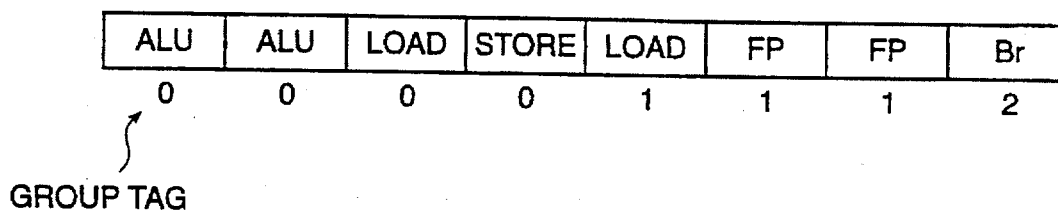


FIG. 12

CLOCK 1 —	ALU	ALU	LOAD	STORE
CLOCK 2 —	LOAD	FP	FP	
CLOCK 3 —	BRANCH			

FIG. 13

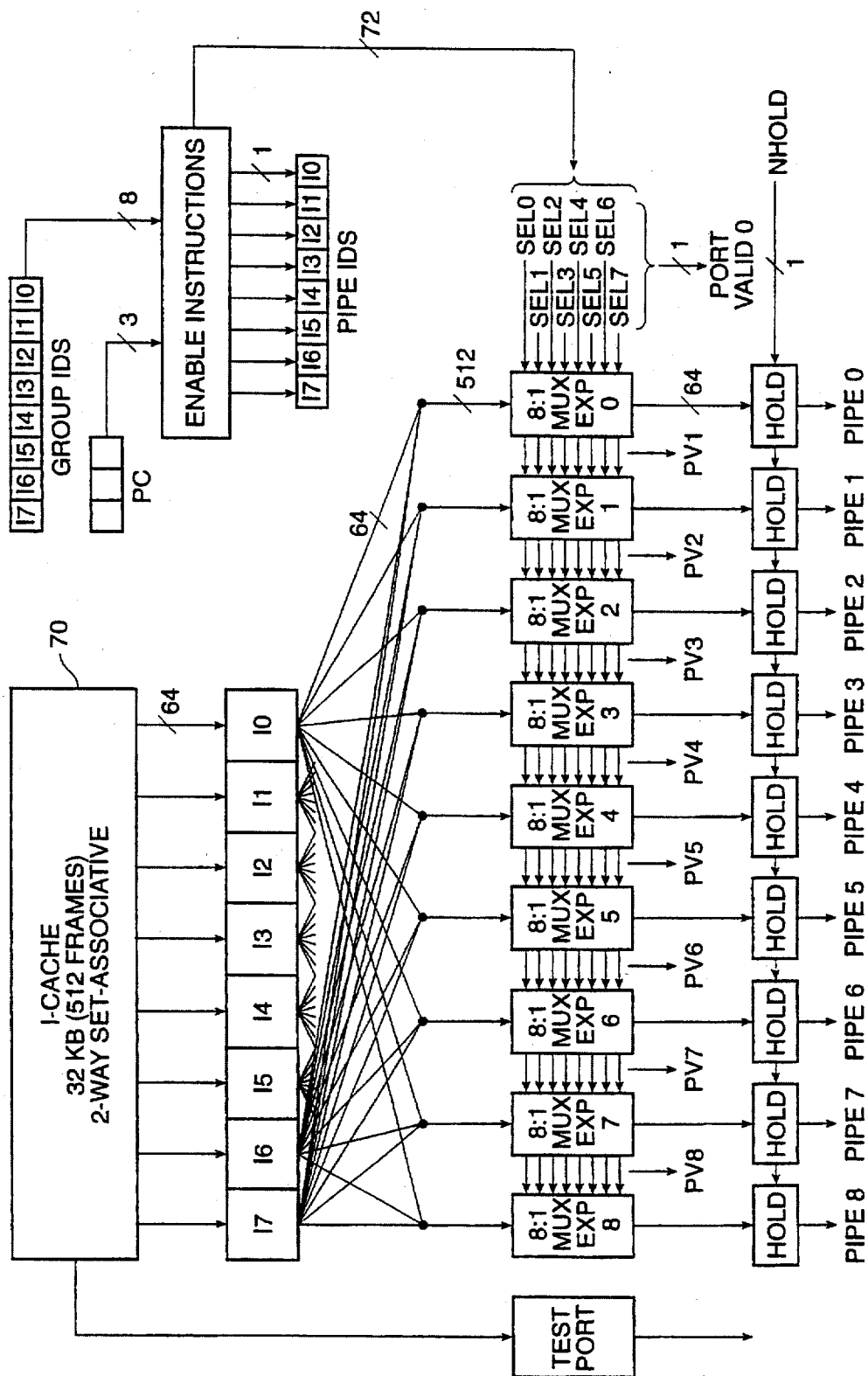


FIG. 14

U.S. Patent

Sep. 24, 1996

Sheet 11 of 11

5,560,028

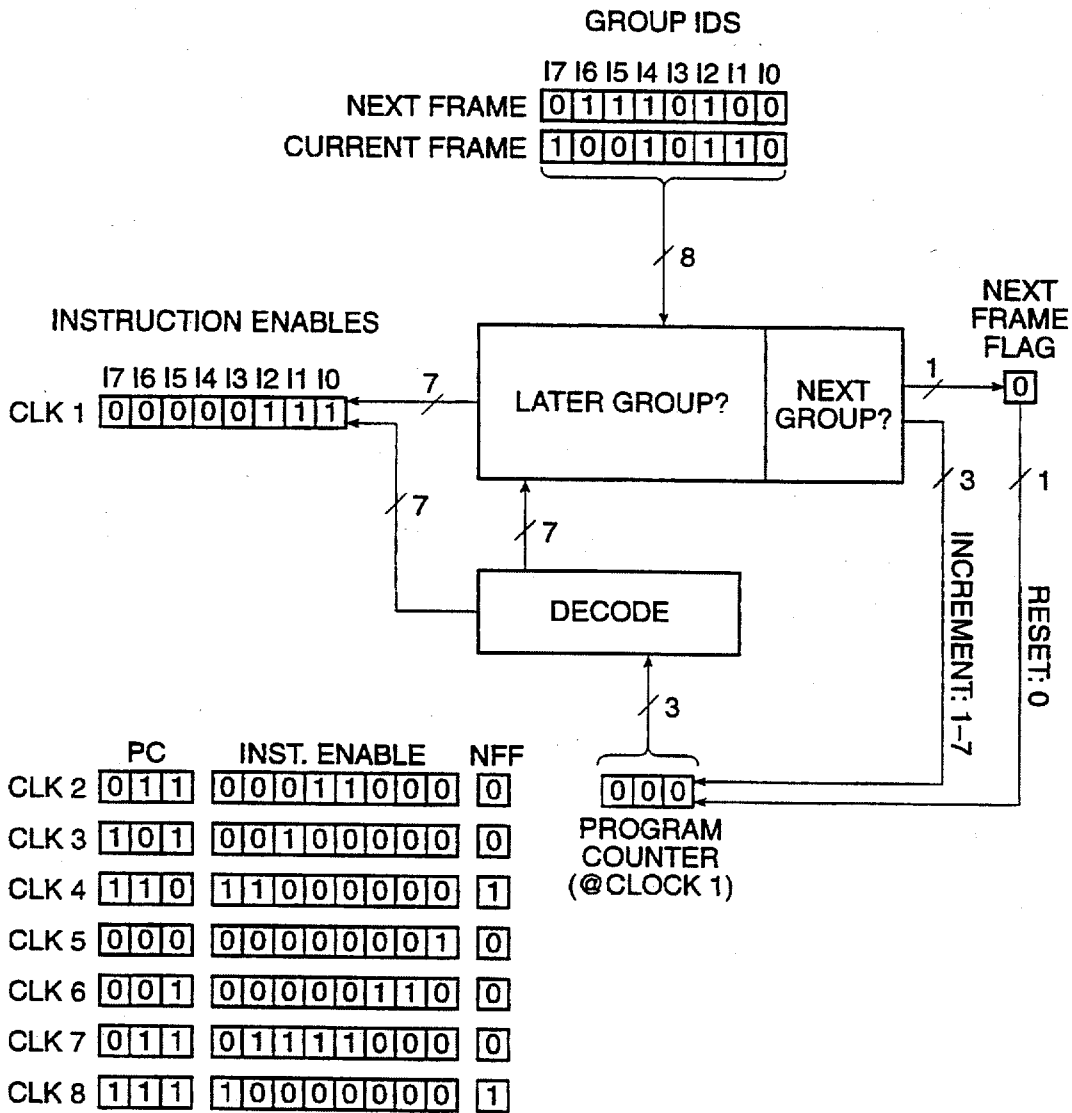


FIG. 15

5,560,028

1

SOFTWARE SCHEDULED SUPERSCALAR COMPUTER ARCHITECTURE

This is a continuation of application Ser. No. 08/147,800 filed Nov. 5, 1993, now abandoned.

BACKGROUND OF THE INVENTION

This invention relates to the architecture of computing systems, and in particular to an architecture in which groups of instructions may be executed in parallel, as well as to methods and apparatus for accomplishing that.

A common goal in the design of computer architectures is to increase the speed of execution of a given set of instructions. Many solutions have been proposed for this problem, and these solutions generally can be divided into two groups.

According to a first approach, the speed of execution of individual instructions is increased by using techniques directed to decreasing the time required to execute a group of instructions serially. Such techniques include employing simple fixed-width instructions, pipelined execution units, separate instruction and data caches, increasing the clock rate of the instruction processor, employing a reduced set of instructions, using branch prediction techniques, and the like. As a result it is now possible to reduce the number of clocks to execute an instruction to approximately one. Thus, in these approaches, the instruction execution rate is limited to the clock speed for the system.

To push the limits of instruction execution to higher levels, a second approach is to issue more than one instruction per clock cycle, in other words, to issue instructions in parallel. This allows the instruction execution rate to exceed the clock rate. There are two classical approaches to parallel execution of instructions.

Computing systems that fetch and examine several instructions simultaneously to find parallelism in existing instruction streams to determine if any can be issued together are known as superscalar computing systems. In a conventional superscalar system, a small number of independent instructions are issued in each clock cycle. Techniques are provided, however, to prevent more than one instruction from issuing if the instructions fetched are dependent upon each other or do not meet other special criteria. There is a high hardware overhead associated with this hardware instruction scheduling process. Typical superscalar machines include the Intel i960CA, the IBM RIOS, the Intergraph Clipper C400, the Motorola 88110, the Sun SuperSparc, the Hewlett-Packard PA-RISC 7100, the DEC Alpha, and the Intel Pentium.

Many researchers have proposed techniques for superscalar multiple instruction issue. Agerwala, T., and J. Cocke [1987] "High Performance Reduced Instruction Set Processors," *IBM Tech. Rep.* (March), proposed this approach and coined the name "superscalar." IBM described a computing system based on these ideas, and now manufactures and sells that machine as the RS/6000 system. This system is capable of issuing up to four instructions per clock and is described in "The IBM RISC System/6000 Processor," *IBM J. of Res. & Develop.* (January, 1990) 34:1.

The other classical approach to parallel instruction execution is to employ a "wide-word" or "very long instruction word" (VLIW) architecture. A VLIW machine requires a new instruction set architecture with a wide-word format. A VLIW format instruction is a long fixed-width instruction that encodes multiple concurrent operations. VLIW systems use multiple independent functional units. Instead of issuing

2

multiple independent instructions to the units, a VLIW system combines the multiple operations into one very long instruction. For example, in a VLIW system, multiple integer operations, floating point operations, and memory references may be combined in a single "instruction." Each VLIW instruction thus includes a set of fields, each of which is interpreted and supplied to an appropriate functional unit. Although the wide-word instructions are fetched and executed sequentially, because each word controls the entire breadth of the parallel execution hardware, highly parallel operation results. Wide-word machines have the advantage of scheduling parallel operation statically, when the instructions are compiled. The fixed width instruction word and its parallel hardware, however, are designed to fit the maximum parallelism that might be available in the code, and most of the time far less parallelism is available in the code. Thus for much of the execution time, most of the instruction bandwidth and the instruction memory are unused.

There is often a very limited amount of parallelism available in a randomly chosen sequence of instructions, especially if the functional units are pipelined. When the units are pipelined, operations being issued on a given clock cycle cannot depend upon the outcome of any of the previously issued operations already in the pipeline. Thus, to efficiently employ VLIW, many more parallel operations are required than the number of functional units.

Another disadvantage of VLIW architectures which results from the fixed number of slots in the very long instruction word for classes of instructions, is that a typical VLIW instruction will contain information in only a few of its fields. This is inefficient, requiring the system to be designed for a circumstance that occurs only rarely—a fully populated instruction word.

Another disadvantage of VLIW systems is the need to increase the amount of code. Whenever an instruction is not full, the unused functional units translate to wasted bits, no-ops, in the instruction coding. Thus useful memory and/or instruction cache space is filled with useless no-op instructions. In short, VLIW machines tend to be wasteful of memory space and memory bandwidth except for only a very limited class of programs.

The term VLIW was coined by J. A. Fisher and his colleagues in Fisher, J. A., J. R. Ellis, J. C. Ruttenberg, and A. Nicolau [1984], "Parallel Processing: A Smart Compiler and a Dumb Machine," *Proc. SIGPLAN Conf. on Compiler Construction* (June), Palo Alto, CA, 11-16. Such a machine was commercialized by Multiflow Corporation.

For a more detailed description of both superscalar and VLIW architectures, see *Computer Architecture—a Quantitative Approach*, John L. Hennessy and David A. Patterson, Morgan Kaufmann Publishers, 1990.

SUMMARY OF THE INVENTION

We have developed a computing system architecture, which we term software-scheduled superscalar, which enables instructions to be executed both sequentially and in parallel, yet without wasting space in the instruction cache or registers. Like a wide-word machine, we provide for static scheduling of concurrent operations at program compilation. Instructions are also stored and loaded into fixed width frames (equal to the width of a cache line). Like a superscalar machine, however, we employ a traditional instruction set, in which each instruction encodes only one basic operation (load, store, etc.). We achieve concurrence by fetching and dispatching "groups" of simple individual

5,560,028

3

instructions, arranged in any order. The architecture of our invention relies upon the compiler to assign instruction sequence codes to individual instructions at the time they are compiled. During execution these instruction sequence codes are used to sort the instructions into appropriate groups and execute them in the desired order. Thus our architecture does not suffer the high hardware overhead and runtime constraints of the superscaler strategy, nor does it suffer the wasted instruction bandwidth and memory typical of VLIW systems.

Our system includes a mechanism, an associative crossbar, which routes in parallel each instruction in an arbitrarily selected group to an appropriate pipeline, as determined by a pipeline tag applied to that instruction during compilation. Preferably, the pipeline tag will correspond to the type of functional unit required for execution of that instruction, e.g., floating point unit 1. All instructions in a selected group can be dispatched simultaneously.

Thus, in one implementation, our system includes a cache line, register, or other means for holding at least one group of instructions to be executed in parallel, each instruction in the group having associated therewith a pipeline identifier indicative of the pipeline for executing that instruction and a group identifier indicative of the group of instructions to be executed in parallel. The group identifier causes all instructions having the same group identifier to be executed simultaneously, while the pipeline identifier causes individual instructions in the group to be supplied to an appropriate pipeline.

In another embodiment the register holds multiple groups of instructions, and all of the instructions in each group having a common group identifier are placed next to each other, with the group of instructions to be executed first placed at one end of the register, and the instructions in the group to be executed last placed at the other end of the register.

In another embodiment of our invention a method of executing arbitrary numbers of instructions in a stream of instructions in parallel includes the steps of compiling the instructions to determine which instructions can be executed simultaneously, assigning group identifiers to sets of instructions that can be executed in parallel, determining a pipeline for execution of each instruction, assigning a pipeline identifier to each instruction, and placing the instructions in a cache line or register for execution by the pipelines.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is a block diagram illustrating a preferred implementation of this invention;

FIG. 2 is a diagram illustrating the data structure of an instruction word in this system;

FIG. 3 is a diagram illustrating a group of instruction words;

FIG. 4 is a diagram illustrating a frame containing from one to eight groups of instructions;

FIG. 5a illustrates the frame structure for one maximum-sized group of eight instructions;

FIG. 5b illustrates the frame structure for a typical mix of three intermediate sized group of instructions;

FIG. 5c illustrates the frame structure for eight minimum-sized groups, each of one instruction;

FIG. 6 illustrates an instruction word after predecoding;

FIG. 7 illustrates the operation of the predecoder;

4

FIG. 8 is a diagram illustrating the overall structure of the instruction cache;

FIG. 9 is a diagram illustrating the manner in which frames are selected from the instruction cache;

FIG. 10 is a diagram illustrating the group selection function in the associative crossbar;

FIG. 11 is a diagram illustrating the group dispatch function in the associative crossbar;

FIG. 12 is a diagram illustrating a hypothetical frame of instructions; and

FIG. 13 is a diagram illustrating the manner in which the groups of instructions in FIG. 12 are issued on different clock cycles.

FIG. 14 is a diagram illustrating another embodiment of the associative crossbar.

FIG. 15 is a diagram illustrating the group select function in further detail.

DESCRIPTION OF THE SPECIFIC EMBODIMENTS

FIG. 1 is a block diagram of a computer system according to the preferred embodiment of this invention. FIG. 1 illustrates the organization of the integrated circuit chips by which the computing system is formed. As depicted, the system includes a first integrated circuit 10 that includes a central processing unit, a floating point unit, and an instruction cache.

In the preferred embodiment the instruction cache is a 16 kilobyte two-way set-associative 32 byte line cache. A set associative cache is one in which the lines (or blocks) can be placed only in a restricted set of locations. The line is first mapped into a set, but can be placed anywhere within that set. In a two-way set associative cache, two sets, or compartments, are provided, and each line can be placed in one compartment or the other.

The system also includes a data cache chip 20 that comprises a 32 kilobyte four-way set-associative 32 byte line cache. The third chip 30 of the system includes a predecoder, a cache controller, and a memory controller. The predecoder and instruction cache are explained further below. For the purposes of this invention, the CPU, FPU, data cache, cache controller and memory controller all may be considered of conventional design.

The communication paths among the chips are illustrated by arrows in FIG. 1. As shown, the CPU/FPU and instruction cache chip communicates over a 32 bit wide bus 12 with the predecoder chip 30. The asterisk is used to indicate that these communications are multiplexed so that a 64 bit word is communicated in two cycles. Chip 10 also receives information over 64 bit wide buses 14, 16 from the data cache 20, and supplies information to the data cache 20 over three 32 bit wide buses 18.

The specific functions of the predecoder are described in much greater detail below; however, essentially it functions to decode a 32 bit instruction received from the secondary cache into a 64 bit word, and to supply that 64 bit word to the instruction cache on chip 10.

The cache controller on chip 30 is activated whenever a first level cache miss occurs. Then the cache controller either goes to main memory or to the secondary cache to fetch the needed information. In the preferred embodiment the secondary cache lines are 32 bytes and the cache has an 8 kilobyte page size.

5,560,028

5

The data cache chip 20 communicates with the cache controller chip 30 over another 32 bit wide bus. In addition, the cache controller chip 30 communicates over a 64 bit wide bus 32 with the DRAM memory, over a 128 bit wide bus 34 with a secondary cache, and over a 64 bit wide bus 36 to input/output devices.

As will be described, the system shown in FIG. 1 includes both conventional and novel features. The system includes multiple pipelines able to operate in parallel on separate instructions. The instructions that can be dispatched to these parallel pipelines simultaneously, in what we term "instruction groups," have been identified by the compiler and tagged with a group identification tag. Thus, the group tag designates instructions that can be executed simultaneously. Instructions within the group are also tagged with a pipeline tag indicative of the specific pipeline to which that instruction should be dispatched. This operation is also performed by the compiler.

In this system, each group of instructions can contain an arbitrary number of instructions ordered in an arbitrary sequence. The only limitation is that all instructions in the group must be capable of simultaneous execution; e.g., there cannot be data dependency between instructions. The instruction groups are collected into larger sets and are organized into fixed width "frames" and stored. Each frame can contain a variable number of tightly packed instruction groups, depending upon the number of instructions in each group and on the width of the frame.

Below we describe this concept more fully, as well as describe a mechanism to route in parallel each instruction in an arbitrarily selected group to its appropriate pipeline, as determined by the pipeline tag of the instruction.

In the following description of the word, group, and frame concepts mentioned above, specific bit and byte widths are used for the word, group and frame. It should be appreciated that these widths are arbitrary, and can be varied as desired. None of the general mechanisms described for achieving the result of this invention depends upon the specific implementation.

In one embodiment of this system the central processing unit includes eight functional units and is capable of executing eight instructions in parallel. We designate these pipelines using the digits 0 to 7. Also, for this explanation each instruction word is 32 bits (4 bytes) long, with a bit, for example, the high order bit S being reserved as a flag for group identification. FIG. 2 therefore shows the general format of all instructions. As shown by FIG. 2, bits 0 to 30 represent the instruction, with the high order bit 31 reserved to flag groups of instructions, i.e., collections of instructions the compiler has determined may be executed in parallel.

FIG. 3 illustrates a group of instructions. A group of instructions consists of one to eight instructions (because there are eight pipelines in the preferred implementation) ordered in any arbitrary sequence; each of which can be dispatched to a different parallel pipeline simultaneously.

FIG. 4 illustrates the structure of an instruction frame. In the preferred embodiment an instruction frame is 32 bytes wide and can contain up to eight instruction groups, each comprising from one to eight instructions. This is explained further below.

When the instruction stream is compiled before execution, the compiler places instructions in the same group next to each other in any order within the group and then places that group in the frame. The instruction groups are ordered within the frame from left to right according to their issue sequence. That is, of the groups of instructions in the frame,

6

the first group to issue is placed in the leftmost position, the second group to issue is placed in the next position to the right, etc. Thus, the last group of instructions to issue within that frame will be placed in the rightmost location in the frame. As explained, the group affiliation of all instructions in the same group is indicated by setting the S bit (bit 31 in FIG. 2) to the same value. This value toggles back and forth from 0 to 1 to 0, etc., between adjacent groups to thereby identify the groups. Thus, all instructions in the first group in a frame have the S bit set to 0, all instructions in the second group have the S bit set to 1, all instructions in the third group have the S bit set to 0, etc., for all groups of instructions in the frame.

To clarify the use of a frame, FIG. 5 illustrates three different frame structures for different hypothetical groups of instructions. In FIG. 5a the frame structure for a group of eight instructions, all of which can be issued simultaneously, is shown. The instruction words are designated W0, W1, . . . , W7. The S bit for each one of the instruction words has been set to 0 by the compiler, thereby indicating that all eight instructions can be issued simultaneously.

FIG. 5b illustrates the frame structure for a typical mixture of three intermediate sized groups of instructions. In FIG. 5b these three groups of instructions are designated Group 0, Group 1 and Group 2. Shown at the left-hand side of FIG. 5b is Group 0 that consists of two instruction words W0 and W1. The S bits for each of these instructions has been set to 0. Group 1 of instructions consists of three instruction words, W2, W3 and W4, each having the S bit set to 1. Finally, Group 2 consists of three instruction words, W5, W6 and W7, each having its S bit set to 0.

FIG. 5c illustrates the frame structure for eight minimum sized groups, each consisting of a single instruction. Because each "group" of a single instruction must be issued before the next group, the S bits toggle in a sequence 01010101 as shown.

As briefly mentioned above, in the preferred embodiment the group identifiers are associated with individual instructions in a group during compilation. In the preferred embodiment, this is achieved by compiling the instructions to be executed using a well-known compiler technology. During the compilation, the instructions are checked for data dependencies, dependence upon previous branch instructions, or other conditions that preclude their execution in parallel with other instructions. These steps are performed using a well-known compiler. The result of the compilation is a group identifier being associated with each instruction. It is not necessary that the group identifier be added to the instruction as a tag, as shown in the preferred embodiment and described further below. In an alternative approach, the group identifier is provided as a separate tag that is later associated with the instruction. This makes possible the execution of programs on our system, without need to revise the word width.

In addition, in some embodiments the compiler will determine the appropriate pipeline for execution of an individual instruction. This determination is essentially a determination of the type of instruction provided. For example, load instructions will be sent to the load pipeline, store instructions to the store pipeline, etc. The association of the instruction with the give pipeline can be achieved either by the compiler, or by later examination of the instruction itself, for example during predecoding.

Referring again to FIG. 1, in normal operation the CPU will execute instructions from the instruction cache, according to well-known principles. On an instruction cache miss,

5,560,028

7

however, the entire frame containing the instruction missed is transferred from the main memory into the secondary cache and then into the primary instruction cache, or from the secondary cache to the primary instruction cache, where it occupies one line of the instruction cache memory. Because instructions are only executed out of the instruction cache, all instructions ultimately undergo the following procedure.

At the time a frame is transferred into the instruction cache, the instruction word in that frame is predecoded by the predecoder 30 (FIG. 1), which as is explained below decodes the retrieved instruction into a full 64 bit word. As part of this predecoding the S bit of each instruction is expanded to a full 3 bit field 000, 001, . . . , 111, which provides the explicit binary group number of the instruction. In other words, the predecoder, by expanding the S bit to a three bit sequence explicitly provides information that the instruction group 000 must execute before instruction group 010, although both groups would have all instructions within the group have S bits set to 0. Because of the frame rules for sequencing groups, these group numbers correspond to the order of issue of the groups of instructions. Group 0 (000) will be issued first, Group 1 (001), if present, will be issued second, Group 2 (010) will be issued third. Ultimately, Group 7 (111), if present, will be issued last. At the time of predecoding of each instruction, the S value of the last word in the frame, which belongs to the last group in the frame to issue, is stored in the tag field for that line in the cache, along with the 19 bit real address and a valid bit. The valid bit is a bit that specifies whether the information in that line in the cache is valid. If the bit is not set to "valid," there cannot be a match or "hit" on this address. The S value from the last instruction, which S value is stored in the tag field of the line in the cache, provides a "countdown" value that can be used to know when to increment to the next cache line.

As another part of the predecoding process, a new 4 bit field prefix is added to each instruction giving the explicit pipe number of the pipeline to which that instruction will be routed. The use of four bits, rather than three allows the system to be later expanded with additional pipelines. Thus, at the time an instruction is supplied from the predecoder to the instruction cache, each instruction will have the format shown in FIG. 6. As shown by FIG. 6, bits 0 to 56 provide 57 bits for the instruction, bits 57, 58 and 59 form the full 3 bit S field, and bits 60-63 provide the 4 bit P field.

FIG. 7 illustrates the operation of the predecoder in transferring a frame from memory to the instruction cache. In the upper portion of FIG. 7, the frame is shown with a hypothetical four groups of instructions. The first group consists of a single instruction, the second group of three instructions, and each of the third and fourth groups of two instructions. As described, instruction is 32 bits in length and include an S bit to separate the groups. The predecoder decodes the instruction shown in the upper portion of FIG. 7 into the instruction shown in the lower portion of FIG. 7. As shown, the instructions are expanded to 64 bit length, with each instruction including a 4 bit identification of the pipeline to which the instruction is to be assigned, and the expanded group field to designate the groups of instructions that can be executed together. For illustration, hypothetical pipeline tags have been applied. Additionally, the predecoder examines each frame for the minimum number of clocks required to execute the frame, and that number is appended to the address tag 45 for the line. The address tag consists of bits provided for the real address for the line, 1 bit to designate the validity of the frame, and 3 bits to specify the minimum time in number of clock cycles, for that frame

8

to issue. The number of clocks for the frame to issue is determined by the group identification number of the last word in the frame. At this stage, the entire frame shown in the lower portion of FIG. 7 is present in the instruction cache.

It may be desirable to implement the system of this invention on computer systems that already are in existence and therefore have instruction structures that have already been defined without fields for the group information, pipeline information, or both. In this case in another embodiment of this invention the group and pipeline information is supplied on a different clock cycle, then combined with the instructions in the cache. Such an approach can be achieved by adding a "no-op" instruction with fields that identify which instructions are in which group, and identify the pipeline for execution of the instruction, or by supplying the information relating to the parallel instructions in another manner. It therefore should be appreciated that the manner in which the data arrives at the crossbar to be processed is somewhat arbitrary. We use the word "associated" herein to designate the concept that the pipeline and group identifiers are not required to have a fixed relationship to the instruction words. That is, the pipeline and group identifiers need not be imbedded within the instructions themselves as shown in FIG. 7. Instead they may arrive from another means, or on a different cycle.

FIG. 8 is a simplified diagram illustrating the secondary cache, the predecoder, and the instruction cache. This drawing, as well as FIGS. 9, 10 and 11, are used to explain the manner in which the instructions tagged with the P and S fields are routed to their designated instruction pipelines.

In FIG. 8 instruction frames are fetched in a single transfer across a 256 bit (32 byte) wide path from a secondary cache 50 into the predecoder 60. As explained above, the predecoder expands each 32 bit instruction in the frame to its full 64 bit wide form and prefixes the P and S fields. After predecoding the 512 bit wide instruction is transferred into the primary instruction cache 70. At the same time, tag is placed into the tag field 74 for that line.

The instruction cache operates as a conventional physically-addressed instruction cache. In the example depicted in FIG. 8, the instruction cache will contain 512 bit fully-expanded instruction frames of eight instructions each organized in two compartments of 256 lines.

Address sources for the instruction cache arrive at a multiplexer 80 that selects the next address to be fetched. Because instructions are always machine words, the low order two address bits <1:0> of the 32 bit address field supplied to multiplexer 80 are discarded. These two bits designate byte and half-word boundaries. Of the remaining 30 bits, the next three low order address bits <4:2>, which designate a particular instruction word in a frame, are sent directly via bus 81 to the associative crossbar (explained in conjunction with subsequent figures). The next low eight address bits <12:5> are supplied over bus 82 to the instruction cache 70 where they are used to select one of the 256 lines in the instruction cache. Finally, the remaining 19 bits of the virtual address <31:13> are sent to the translation lookaside buffer (TLB) 90. The TLB translates these bits into the high 19 bits of the physical address. The TLB then supplies them over bus 84 to the instruction cache. In the cache they are compared with the tag of the selected line, to determine if there is a "hit" or a "miss" in the instruction cache.

If there is a hit in the instruction cache, indicating that the addressed instruction is present in the cache, then the

5,560,028

9

selected frame containing the addressed instruction is transferred across the 512 bit wide bus 73 into the associative crossbar 100. The associative crossbar 100 then dispatches the addressed instruction, with the other instructions in its group, if any, to the appropriate pipelines over buses 110, 111, . . . , 117. Preferably the bit lines from the memory cells containing the bits of the instruction are themselves coupled to the associative crossbar. This eliminates the need for numerous sense amplifiers, and allows the crossbar to operate on the lower voltage swing information from the cache line directly, without the normally intervening driver circuitry to slow system operation.

FIG. 9 is a block diagram illustrating in more detail the frame selection process. As shown, bits <4:2> of the virtual address are supplied directly to the associative crossbar 100 over bus 81. Bus 81, as explained above will preferably include a pair of conductors, the bit lines, for each data bit in the field. Bits <12:5> supplied over bus 82 are used to select a line in the instruction cache. The remaining 19 bits, translated into the 19 high order bits <31:13> of physical address, are used to compare against the tags of the two selected lines (one from each compartment of the cache) to determine if there is a hit in either compartment. If there is a hit, the two 512 bit wide frames are supplied to multiplexer 120. The choice of which line is ultimately supplied to associative crossbar 100 depends upon the real address bits <31:13> that are compared by comparators 125. The output from comparators 125 thus selects the appropriate frame for transfer to the crossbar 100.

FIG. 10 illustrates in more detail the group select function of the associative crossbar. A 512 bit wide register 130, preferably formed by the SRAM cells in the instruction cache contains the frame of the instructions to be issued. For the purposes of illustration, register 130 is shown as containing a frame having three groups of instructions, with Group 0 including words W0, W1 and W2; Group 1 containing words W3, W4 and W5; and Group 2 containing words W6 and W7. For illustration, the instructions in Group 0 are to be dispatched to pipelines 1, 2 and 3; the instructions in Group 1 to pipelines 1, 3 and 6; and the instructions in Group 2 to pipelines 1 and 6. The three S bits (group identification field) of each instruction in the frame are brought out to an 8:1 multiplexer 140 over buses 131, 132, 133, . . . , 138. The S field of the next group of instructions to be executed is present in a 3 bit register 145. As shown in FIG. 10, the hypothetical contents of register 145 are 011. These bits have been loaded into register 145 using bus 81 described in conjunction with FIG. 9. Multiplexer 140 then compares the value in this register against the contents of the S field in each of the instruction words. If the two values match, the appropriate decoder 150 is enabled, permitting the instruction word to be processed on that clock cycle. If the values do not match, the decoder is disabled and the instruction words are not processed on that clock cycle. In the example depicted in FIG. 10, the contents of register 145 match the S field of the Group 1 instructions. The resulting output, supplied over bus 142, is communicated to S register 144 and then to the decoders via bus 146. The S register contents enable decoders 153, 154 and 155, all of which are in Group 001. As will be shown in FIG. 11, this will enable these instructions W3, W4 and W5 to be sent to the pipelines for processing.

FIG. 11 is a block diagram illustrating the group dispatching of the instructions in the group to be executed. The same registers are shown across the upper portion of FIG. 11 as in the lower portion of FIG. 10. As shown in FIG. 11, the crossbar switch itself consists of two sets of crossing path-

10

ways. In the horizontal direction are the pipeline pathways 180, 181, . . . , 187. In the vertical direction are the instruction word paths, 190, 191, . . . , 197. Each of these pipeline and instruction pathways is themselves a bus for transferring the instruction word. Each horizontal pipeline pathway is coupled to a pipeline execution unit 200, 201, 202, . . . , 207. Each of the vertical instruction word pathways 190, 191, . . . , 197 is coupled to an appropriate portion of register 130 (FIG. 10).

The decoders 170, 171, . . . , 177 associated with each instruction word pathway receive the 4 bit pipeline code from the instruction. Each decoder, for example decoder 170, provides as output eight 1 bit control lines. One of these control lines is associated with each pipeline pathway crossing of that instruction word pathway. Selection of a decoder as described with reference to FIG. 10 activates the output bit control line corresponding to that input pipe number. This signals the crossbar to close the switch between the word path associated with that decoder and the pipe path selected by that bit line. Establishing the cross connection between these two pathways causes a selected instruction word to flow into the selected pipeline. For example, decoder 173 has received the pipeline bits for word W3. Word W3 has associated with it pipeline path 1. The pipeline path 1 bits are decoded to activate switch 213 to supply instruction word W3 to pipeline execution unit 201 over pipeline path 181. In a similar manner, the identification of pipeline path 3 for decoder D4 activates switch 234 to supply instruction word W4 to pipeline path 3. Finally, the identification of pipeline 6 for word W5 in decoder D5 activates switch 265 to transfer instruction word W5 to pipeline execution unit 206 over pipeline pathway 186. Thus, instructions W3, W4 and W5 are executed by pipes 201, 203 and 206, respectively.

The pipeline processing units 200, 201, . . . , 207 shown in FIG. 11 can carry out desired operations. In a preferred embodiment of the invention, each of the eight pipelines first includes a sense amplifier to detect the state of the signals on the bit lines. In one embodiment the pipelines include first and second arithmetic logic units; first and second floating point units; first and second load units; a store unit and a control unit. The particular pipeline to which a given instruction word is dispatched will depend upon hardware constraints as well as data dependencies.

FIG. 12 is an example of a frame and how it will be executed by the pipeline processors 200-207 of FIG. 11. As shown in FIG. 12 the frame includes three groups of instructions. The first group, with group identification number 0, includes two instructions that can be executed by the arithmetic logic unit, a load instruction and a store instruction. Because all these instructions have been assigned the same group identification number by the compiler, all four instructions can execute in parallel. The second group of instructions consists of a single load instruction and two floating point instructions. Again, because each of these instructions has been assigned "Group 1," all three instructions can be executed in parallel. Finally, the last instruction word in the frame is a branch instruction that, based upon the compiler's decision, must be executed last.

FIG. 13 illustrates the execution of the instructions in the frame shown in FIG. 12. As shown, during the first clock the Group 0 instructions execute, during the second clock the load and floating point instructions execute, and during the third clock the branch instruction executes. To prevent groups from being split across two instruction frames, an instruction frame may be only partially filled, where the last group is too large to fit entirely within the remaining space of the frame.

5,560,028

11

FIG. 14 is a diagram illustrating another embodiment of the associative crossbar. In FIG. 14 nine pipelines 0-8 are shown coupled to the crossbar. The three bit program counter PC points to one of the instructions in the frame, in combination with the set of 8 group identification bits for the frame, indicating the group affiliation of each instruction, are used to enable a subset of the instructions in the frame. The enabled instructions are those at or above the address indicated by the PC that belong to the current group.

The execution ports that connect to the pipelines specified by the pipeline identification bits of the enabled instructions are then selected to multiplex out the appropriate instructions from the current frame. If one or more of the pipelines is not ready to receive a new instruction, a set of hold latches at the output of the execution ports prevents any of the enabled instructions from issuing until the "busy" pipeline is free. Otherwise the instructions pass transparently through the hold latches into their respective pipelines. Accompanying the output of each port is a "port valid" signal that indicates whether the port has valid information to issue to the hold latch.

FIG. 15 is a diagram illustrating the group select function in further detail. This figure illustrates the mechanism used to enable an addressed group of instructions within a frame. The program counter is first decoded into a set of 14 bit signals. Seven of these signals are combined with the eight group identifiers of the current frame to determine whether each of the seven instructions, 11 to 17, is or is not the start of a later group. This information can then be combined with the other 7 bit signals from the PC decoder to determine which of the eight instructions in the frame should be enabled. Using the pipeline identifying field each enabled instruction can be combined with the other 7 bit signal to determine which of the eight instructions in the frame should be enabled. Each such enabled instruction can then signal the execution port, as determined by the pipeline identifier, to multiplex out the enabled instruction. Thus if I2 is enabled, and the pipeline code is 5, the select line from I2 to port 5 is activated, causing I2 to flow to the hold latch at pipe 5.

Because the instructions that start later groups are known, the system can decide easily which instruction starts the next group. This information is used to update the PC to the address of the next group of instructions. If no instruction in the frame begins the next group, i.e., the last instruction group has been dispatched to the pipelines, a flag is set. The flag causes the next frame of instructions to be brought into the crossbar. The PC is then reset to I0. Shown in the figure is an exemplary sequence of the values that the PC, the instruction enable bits and the next frame flag take on over a sequence of eight clocks extending over two frames.

The processor architecture described above provides many unique advantages to a system using this invention. The system described is extremely flexible, enabling instructions to be executed sequentially or in parallel, depending entirely upon the "intelligence" of the compiler. As compiler technology improves, the described hardware can execute programs more rapidly, not being limited to any particular frame width, number of instructions capable of parallel execution, or other external constraints. Importantly, the associative crossbar aspect of this invention relies upon the content of the message being decoded, not upon an external control circuit acting independently of the instructions being executed. In essence, the associative crossbar is self directed. In the preferred embodiment the system is capable of a parallel issue of up to eight operations per cycle. For a more complete description of the associative crossbar, see

12

copending U.S. application Ser. No. 08/147,797, filed Nov. 5, 1993, and entitled "Instruction Cache Associative Crossbar Switch."

Although the foregoing has been a description of the preferred embodiment of the invention, it will be apparent to those of skill in the art that numerous modifications and variations may be made to the invention without departing from the scope as described herein. For example, arbitrary numbers of pipelines, arbitrary numbers of decoders, and different architectures may be employed, yet rely upon the system we have developed.

We claim:

1. A computing system having a plurality of processing pipelines for executing groups of individual instructions, within very long instruction words, each individual instruction to be executed in each group being executed by different processing pipelines in parallel, the computing system comprising:

a main memory for storing a very long instruction word; a very long instruction word storage, coupled to the main memory, for receiving the very long instruction word from the main memory, and for holding the very long instruction word, the very long instruction word including a predetermined number N of individual instructions, and including at least one group of M individual instructions to be executed in parallel, where $M \leq N$, each individual instruction in the very long instruction word storage to be executed having a pipeline identifier indicative of a processing pipeline for executing the individual instruction, and having a group identifier indicative of a group of individual instructions to which the individual instruction is assigned for execution in parallel;

group decoder means responsive to the group identifier for each individual instruction in the very long instruction word storage to be executed for enabling each individual instruction in the very long instruction word storage having a similar group identifier, to be executed in parallel by the plurality of processing pipelines; and pipeline decoder means responsive to the pipeline identifier of each individual instructions in the very long instruction word storage to be executed for causing each individual instruction in a group of individual instructions having the similar group identifier to be supplied to the different processing pipelines.

2. The computing system in claim 1, wherein the very long instruction word storage includes the at least one group of M individual instructions, and also includes group identifiers and pipeline identifiers for each individual instruction in the at least one group of M individual instructions.

3. The computing system in claim 2, wherein each individual instruction in the at least one group of M individual instructions has associated therewith a different pipeline identifier.

4. The computing system of claim 1, wherein the very long instruction word storage holds a first group of individual instructions to be executed in parallel and a second group of individual instructions to be executed in parallel after the first group, each individual instruction in the first group having associated therewith a first group identifier different from a second group identifier associated with each individual instruction in the second group, the first group and the second group being placed adjacent to each other in the very long instruction word storage.

5. The computing system of claim 4 wherein:

the very long instruction word storage comprises a line in a cache memory having a fixed number of storage locations; and

5,560,028

13

the first group of individual instructions is placed at one end of the line in the cache memory, and the second group of individual instructions is placed next to the first group of individual instructions.

6. A method of executing in a plurality of processing pipelines arbitrary numbers of instructions in a stream of instructions in parallel which have been compiled to determine which instructions can be executed in parallel, the method comprising:

in response to the compilation, assigning a common group identifier to a group of instructions which can be executed in parallel;

determining a processing pipeline for execution of each instruction in the group of instructions to be executed;

assigning a pipeline identifier to each instruction in the group;

embedding the common group identifier and the pipeline identifier into the group of instructions;

forming a very long instruction word with a fixed number of the instructions including at least the group of instructions having the common group identifier as well as at least one other instruction having a different group identifier; and

storing the very long instruction word in a main memory.

7. A method as in claim 6 further comprising the step of: placing the very long instruction word retrieved from the main memory into a very long instruction word register; and

executing the group of instructions in the plurality of processing pipelines in parallel.

8. A method as in claim 7,

wherein the very long instruction word register holds at least two groups of instructions; and

wherein the step of placing the instructions in the very long instruction word register comprises placing the group of instructions adjacent to the at least one other instruction having the different group identifier in the very long instruction word register.

9. A method as in claim 8 wherein the step of executing the group of instructions in parallel comprises:

coupling the very long instruction word register to a detection means to receive group identifiers of each instruction to be executed in the very long instruction word; and

supplying only instructions having the common group identifier to the processing pipelines.

10. In a computing system having a plurality of processing pipelines in which groups of individual instructions, within very long instruction words, are executable in parallel by processing pipelines, a method for supplying each individual instruction in a group to be executed in parallel to corresponding appropriate processing pipelines, the method comprising:

retrieving a very long instruction word from a main memory;

storing in a very long instruction word storage the very long instruction word, the very long instruction word including groups of individual instructions to be executed in parallel, each individual instruction to be executed in the very long instruction word having embedded therein a pipeline identifier indicative of the corresponding appropriate processing pipeline which will execute that instruction and a group identifier indicative of the group identification;

comparing the group identifier of each individual instruction in the very long instruction word to an execution group identifier to identify an execution group; and

14

using the pipeline identifier of individual instructions in the execution group to execute each individual instruction in the execution group in the corresponding appropriate processing pipelines.

11. In a computing system having a plurality of processing pipelines in which groups of individual instructions, from a very long instruction word, are executable in parallel by the plurality of processing pipelines, an apparatus for routing each individual instruction in a particular group to be executed in parallel to an appropriate processing pipeline, the apparatus comprising:

a main memory for storing the very long instruction word;

a very long instruction word storage coupled to the main memory, for receiving the very long instruction word from the main memory and for holding the very long instruction word, the very long instruction word including groups of individual instructions, each individual instruction to be executed in the very long instruction word storage having associated therewith a pipeline identifier indicative of a processing pipeline for executing that individual instruction and also having associated therewith a group identifier to designate a group of individual instructions to which that individual instruction is assigned, the pipeline identifier and the group identifier embedded in the very long instruction word;

a crossbar switch having a first set of connectors coupled to the very long instruction word storage and a second set of connectors coupled to the plurality of processing pipelines;

a router coupled to the very long instruction word storage and the crossbar switch, responsive to a pipeline identifier for each individual instruction to be executed in the group for routing each individual instruction in the group from connectors of the first set of connectors onto appropriate connectors of the second set of connectors, to thereby supply each individual instruction in the group to be executed in parallel to the appropriate processing pipeline.

12. The apparatus of claim 11,

wherein the first set of connectors includes a set of first communication buses, one first communication bus for each individual instruction to be executed in the very long instruction word storage;

wherein the second set of connectors includes a set of second communication buses, one second communication bus for each processing pipeline; and

wherein the router comprises:

a set of decoders coupled to the very long instruction word storage, each decoder for receiving as input signals the pipeline identifier of each individual instruction in the very long instruction word storage and in response thereto for supplying as output signals switch control signals corresponding to each individual instruction in the very long instruction word storage; and

a set of switches coupled to the set of decoders and to the crossbar switch, one switch of the set of switches at each intersection of each of the first set of communication buses with each of the second set of communication buses, each switch for receiving the switch control signals and for providing connections in response to receiving a corresponding switch control signal to thereby supply each individual instruction in the group to be executed in parallel to the appropriate processing pipeline.

13. The apparatus of claim 12 further comprising:

5,560,028

15

detection means coupled to the very long instruction word storage, for receiving the group identifier of each individual instruction in the very long instruction word storage to be executed and in response thereto supply a group control signal; and

wherein the set of decoders are also coupled to the detection means for receiving the group control signal and in response thereto supply the switch control signal for only those individual instructions in the group to be supplied to the plurality of processing pipelines.

14. The apparatus of claim 13,

wherein the detection means comprises a multiplexer coupled to receive group identifiers of each individual instruction in the very long instruction word storage and a group identifier for a group of individual instructions to be next executed, and in response thereto allow the group of individual instructions to be supplied to the plurality of processing pipelines.

15. Apparatus as in claim 14 wherein the multiplexer supplies output signals to the set of decoders to indicate a group identifier of a group of individual instructions to be next supplied to the plurality of processing pipelines.

16. In a computing system having a plurality of processing pipelines in which groups of individual instructions, within a very long instruction word, are executable by the plurality of processing pipelines, each individual instruction in the very long instruction word to be executed having embedded therein a group identifier and a pipeline identifier, an apparatus for routing each individual instruction of a group of individual instructions to be executed in parallel to an appropriate processing pipeline of the plurality of processing pipelines, the apparatus comprising:

a main memory for storing the very long instruction word; a very long instruction word storage coupled to the main memory, for receiving the very long instruction word from the main memory and for holding the very long instruction word the very long instruction word including groups of instructions to be executed in parallel, including pipeline identifiers and group identifiers;

selection means coupled to the very long instruction word storage for receiving the group identifier for each individual instruction in the very long instruction word, for determining in response thereto a group of individual instructions to be executed in parallel, and for outputting a control signal;

decoder means coupled to the selection means and to the very long instruction word storage, for receiving the control signal and the pipeline identifier for each individual instruction in the very long instruction word, for determining in response thereto the appropriate processing pipeline for each individual instruction of the group, and for outputting switch control signals;

a crossbar switch coupled to the decoder means, having a first set of connectors coupled to the very long instruction word storage for receiving the very long instruction word therefrom and a second set of connectors coupled to the plurality of processing pipelines, for coupling each individual instruction of the group to an appropriate processing pipeline in response to the switch control signals.

17. The apparatus of claim 16,

wherein the first set of connectors comprises a set of first communication buses, one first communication bus for each individual instruction held in the very long instruction word storage;

wherein the second set of connectors comprises a set of second communication buses, one second communication bus for each processing pipeline;

16

wherein the decoder means comprises a set of decoders coupled to receive as first input signals the pipeline identifiers for each individual instruction in the group and as second input signals the pipeline identifiers for remaining individual instructions in the very long instruction word; and

wherein the crossbar switch comprises a set of switches, one switch for every intersection between each of the first set of connectors and each of the second set of connectors, each switch for providing connections, in response to receiving the switch control signals, between each individual instruction in the group to be executed in parallel to the appropriate processing pipeline.

18. The apparatus of claim 17,

wherein the selection means comprises a multiplexer coupled to receive the group identifiers for each individual instruction in the very long instruction word storage, and in response to the group identifiers, enable the decoder means to output switch control signals for each individual instructions of the group.

19. The apparatus of claim 18,

wherein the multiplexer supplies a switch control signal to the decoder means to enable the decoder means to output switch control signals for each individual instruction of the group of individual instructions from the very long instruction word.

20. In a computing system having a plurality of processing pipelines in which groups of individual instructions are executable, each individual instruction in a group executable in parallel by the plurality of processing pipelines, a method for transferring each individual instruction in a group to be executed through a crossbar switch having a first set of connectors coupled to a very long instruction word storage for receiving individual instructions therefrom, a second set of connectors coupled to the plurality of processing pipelines, and switches between the first set and the second set. Of connectors, the method comprising:

retrieving the very long instruction word from a main memory;

storing in the very long instruction word storage, the very long instruction word, the very long instruction word having a set of individual instructions including at least one group of individual instructions to be executed in parallel, each individual instruction in the at least one group having embedded therein a unique pipeline identifier indicative of the processing pipeline which will execute that individual instruction, the very long instruction word storage also including at least one other individual instruction not in the at least one group of individual instructions, the at least one other individual instruction having embedded therein a different pipeline identifier; and

using the unique pipeline identifiers of the individual instructions in the at least one group of individual instructions to control the switches between the first set of connectors and the second set of connectors to thereby supply each individual instruction in the at least one group to be executed in parallel to an appropriate processing pipeline.

21. A method as in claim 20 wherein the step of using the pipeline identifiers comprises:

supplying the unique pipeline identifiers of each individual instructions in the at least one group of individual instructions to individual decoders of a set of decoders, each decoder of which provides an output

5,560,028

17

signal indicative of the Unique pipeline identifiers of the individual instruction supplied thereto; and
 using the output signals of the sets of decoders to control the switches between the first set of connectors and the second set of connectors to thereby supply each individual instruction in the at least one group to be executed in parallel to an appropriate processing pipeline.

22. A method as in claim 21 wherein each individual instruction in the storage further includes a group identifier embedded therein to designate among the instructions present in the very long instruction word storage, which of the individual instructions may be simultaneously supplied to the plurality of processing pipelines, and the method further comprises:

supplying a group identifier for a group of instructions to be executed by the processing pipelines together with the group identifiers of the individual instructions in the at least one group of individual instructions to a selector;

comparing the group identifier of the group of instructions to be executed by the processing pipelines with the group identifiers of the individual instructions in the at least one group of instructions, to provide output comparison signals; and

using both the output comparison signals and the output signals to control the switches between the first set of connectors and the second set of connectors to thereby supply each instruction in the at least one group to be executed in parallel to the appropriate processing pipeline.

23. In a computing system having a plurality of processing pipelines in which groups of individual instructions are

18

executable by the plurality of processing pipelines, a method for supplying each individual instruction in a group of individual instructions to be executed in parallel to an appropriate processing pipeline, the method comprising:

retrieving a very long instruction word from a main memory;

storing in a very long instruction word storage the very long instruction word retrieved from the main memory, the very long instruction word including groups of individual instructions to be executed in parallel, each individual instruction in a group of individual instructions having embedded therein a pipeline identifier indicative of a processing pipeline which will execute that individual instruction and having embedded therein a group identifier indicative of a group identification;

comparing a group identifier for each individual instruction in the very long instruction word with an execution group identifier of those instructions to be next executed in parallel; and

using a pipeline identifier for those instructions to be next executed in parallel to control switches in a crossbar switch having a first set of connectors coupled to the very long instruction word storage for receiving the very long instruction word therefrom and a second set of connectors coupled to the plurality of processing pipelines to thereby supply each individual instruction in the at least one group to be executed in parallel to the appropriate processing pipeline.

* * * * *

UNITED STATES PATENT AND TRADEMARK OFFICE
CERTIFICATE OF CORRECTION

PATENT NO. : 5,560,028
DATED : Sept. 24, 1996
INVENTOR(S) : Howard G. Sachs

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

On title page, item [54], and in col. 1, line 1:

In the title, please delete "SUPERSCALAR" and insert -SUPERSCALER-.

Signed and Sealed this

Thirty-first Day of December, 1996

Attest:



BRUCE LEHMAN

Attesting Officer

Commissioner of Patents and Trademarks

US005794003A

United States Patent [19]
Sachs

[11] Patent Number: 5,794,003

[45] **Date of Patent:** Aug. 11, 1998

[54] INSTRUCTION CACHE ASSOCIATIVE CROSSBAR SWITCH SYSTEM

[75] Inventor: **Howard G. Sachs, Belvedere, Calif.**

[73] Assignee: **Intergraph Corporation, Huntsville, Ala.**

[21] Appl. No.: 754,337

[22] Filed: Nov. 22, 1996

Related U.S. Application Data

[63] Continuation of Ser. No. 498,135, Jul. 5, 1995, abandoned, which is a continuation of Ser. No. 147,797, Nov. 5, 1993, abandoned.

[51] Int. Cl.⁶ G06F 9/30

[52] U.S. Cl. **395/391**; 395/379; 395/800

[58] **Field of Search** 395/709, 390,
395/391, 392, 379, 312, 800

[56] References Cited

FOREIGN PATENT DOCUMENTS

0 449 661 A2 10/1991 European Pat. Off. .

0499 661 A2 10/1991 European Pat. Off.

0 496 928 A2 8/1992 European Pat. Off.

OTHER PUBLICATIONS

Minagawa et al., "Pre-decoding mechanism for superscalar architecture", IEEE Pacific Rim Conference on Communications, Computers, and Signal Processing, pp. 21-24, May 9, 1991.

DeGloria et al., "A programmable instruction format extension to VLIW architecture", *Computer Systems and Software Engineering 6th Annual European Computer Conference*, pp. 36-37, May 4, 1992.

Agerwala et al., "High performance reduced instruction set processors," RC 12434 (#55845), *Computer Science*, Jan. 9, 1987.

Bakoglu et al., "The IBM RISC system/6000 processor: hardware overview." *IBM J. Res. Develop.*, 34(1):12-22 (Jan., 1990).

(List continued on next page.)

U.S. PATENT DOCUMENTS

4,437,149	3/1984	Pomerene et al.	395/389
4,847,755	7/1989	Morrison et al.	395/379
4,933,837	6/1990	Freidin	395/452
5,055,997	10/1991	Sluiter et al.	395/312
5,081,575	1/1992	Hiller et al.	395/312
5,101,341	3/1992	Circello et al.	395/389
5,121,502	6/1992	Rau et al.	395/800
5,129,067	7/1992	Johnson	395/389
5,151,981	9/1992	Westcott et al.	395/185.03
5,179,680	1/1993	Colwell et al.	395/452
5,197,137	3/1993	Kumar et al.	395/677
5,203,002	4/1993	Wetzel	395/800
5,214,763	5/1993	Blaner et al.	395/388
5,226,169	7/1993	Gregor	395/800
5,233,696	8/1993	Suzuki	395/380
5,239,654	8/1993	Ing-Simmons	395/800
5,297,255	3/1994	Hamanaka et al.	395/800
5,297,281	3/1994	Emma et al.	395/392
5,299,321	3/1994	Iizuka	395/388
5,367,694	11/1994	Ueno	395/800
5,442,760	8/1995	Rustad et al.	395/391

Primary Examiner—Parshotam S. Lall

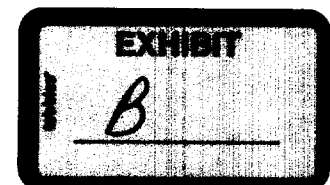
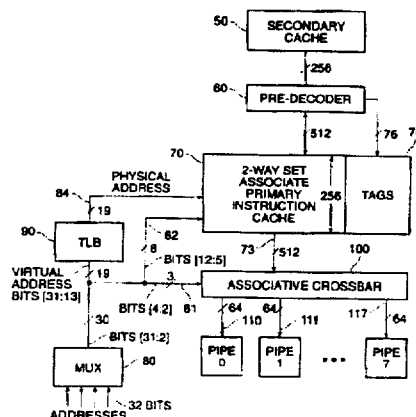
Assistant Examiner—Viet Vu

Attorney, Agent, or Firm—Townsend and Townsend and Crew LLP

[57] **ABSTRACT**

A computing system as described in which individual instructions are executable in parallel by processing pipelines, and instructions to be executed in parallel by different pipelines are supplied to the pipelines simultaneously. The system includes storage for storing an arbitrary number of the instructions to be executed. The instructions to be executed are tagged with pipeline identification tags indicative of the pipeline to which they should be dispatched. The pipeline identification tags are supplied to a system which controls a crossbar switch, enabling the tags to be used to control the switch and supply the appropriate instructions simultaneously to the differing pipelines.

33 Claims, 5 Drawing Sheets



5,794,003

Page 2

OTHER PUBLICATIONS

De Gloria et al., "A programmable instruction format extension to VLIW architectures," *Proceedings Comp. Euro. 1992*, pp. 35-40 (May 4, 1992).

Fisher et al., "Parallel processing: a smart compiler and a dumb machine," *SIGPLAN Notices*, 19(6):37-47 (Jun., 1984).

Hennessy et al., *Computer Architecture; A Quantitative Approach*, ISBN 1-55880-069-8, Morgan Kaufmann Publishers, Inc. (1990).

Brian Case, et al., "DEC Enters Microprocessor Business with Alpha," *Microprocessor Report* (Mar. 4, 1992) 6(3):1, 6-14.

Todd A. Dutton, "The Design of the DEC 3000 Model 500 AXP Workstation," *IEEE* (1993) 1063-6390/93, pp. 449-455.

Brian Allison, "DEC 7000/10000 Model 600 AXP Multi-processor Server," *IEEE* (1993) 1063-6390/93, pp. 456-464.

R. B. Grove, et al., "GEM Optimizing Compilers for Alpha AXP Systems," *IEEE* (1993) 1063-6390/93, pp. 465-473.

Johnson, "Superscalar Microprocessor Design", Prentice-Hall 1991 pp. 233-235.

Minagawa et al, "Pre-Decoding Mechanism for Superscalar Architecture", *IEEE Computers & Signal Processing*, Dec. 1991, pp. 21-24.

U.S. Patent

Aug. 11, 1998

Sheet 1 of 5

5,794,003

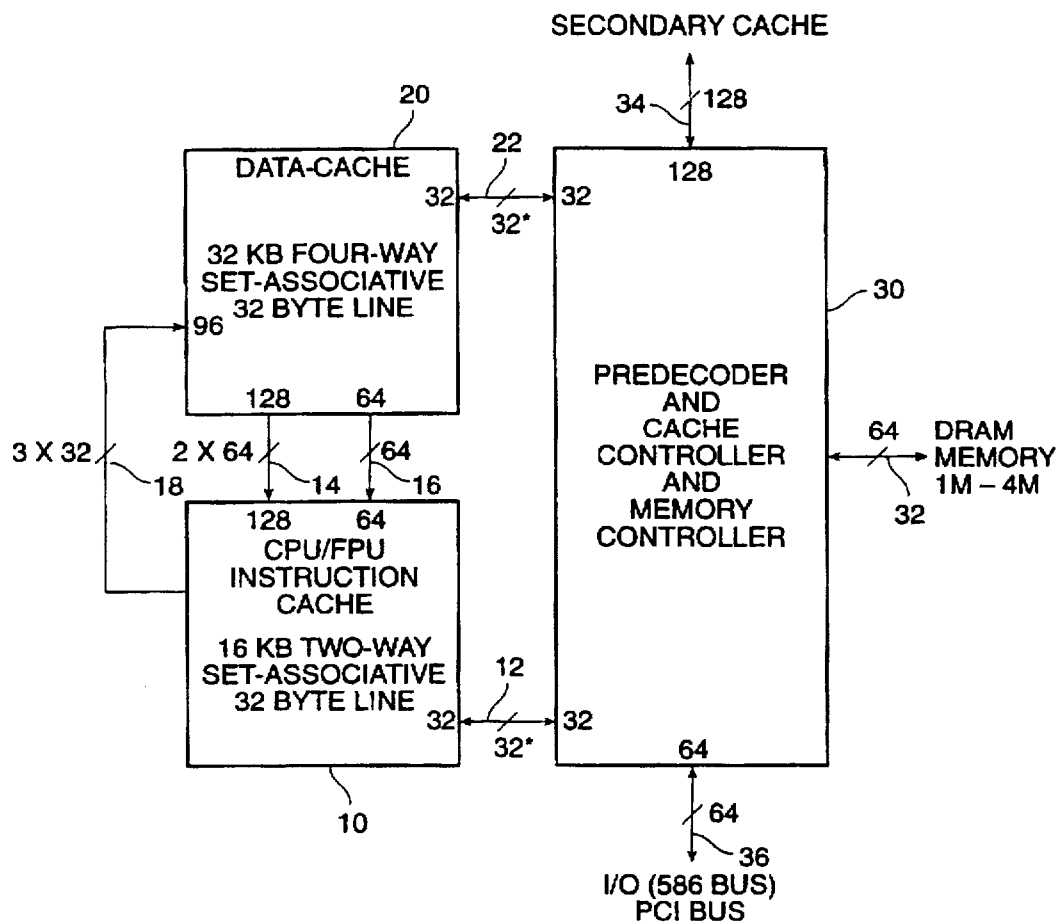


FIG. 1

U.S. Patent

Aug. 11, 1998

Sheet 2 of 5

5,794,003

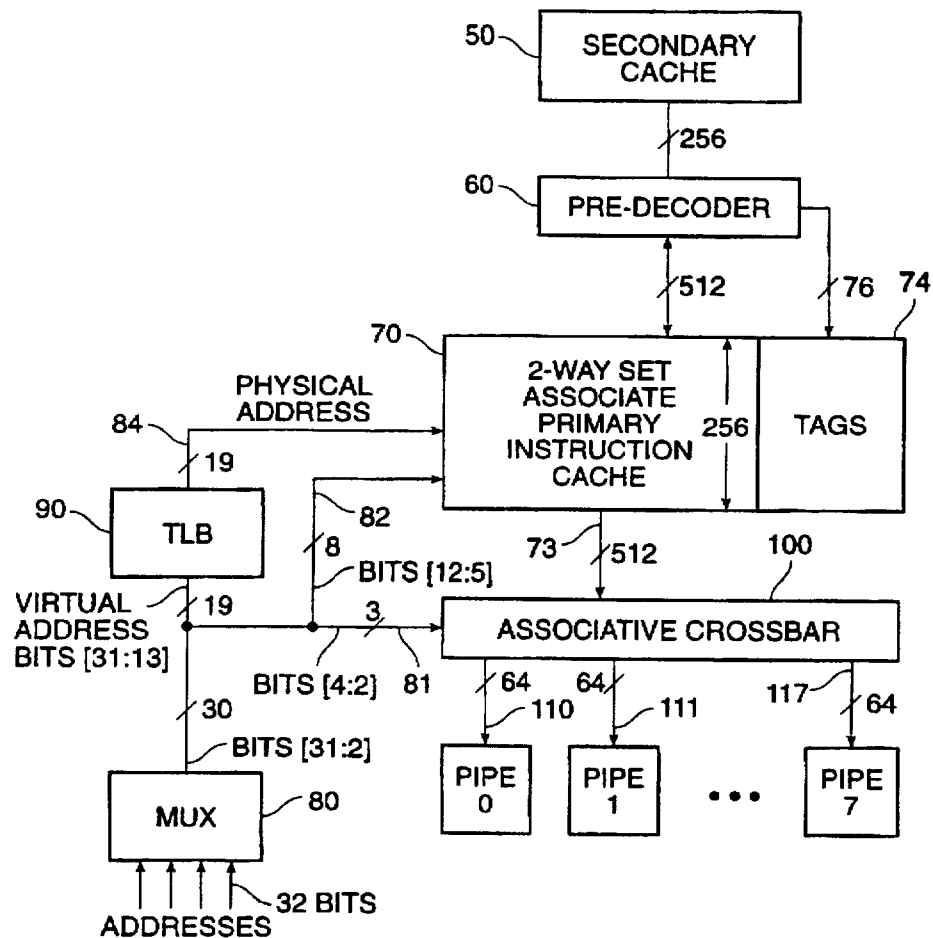


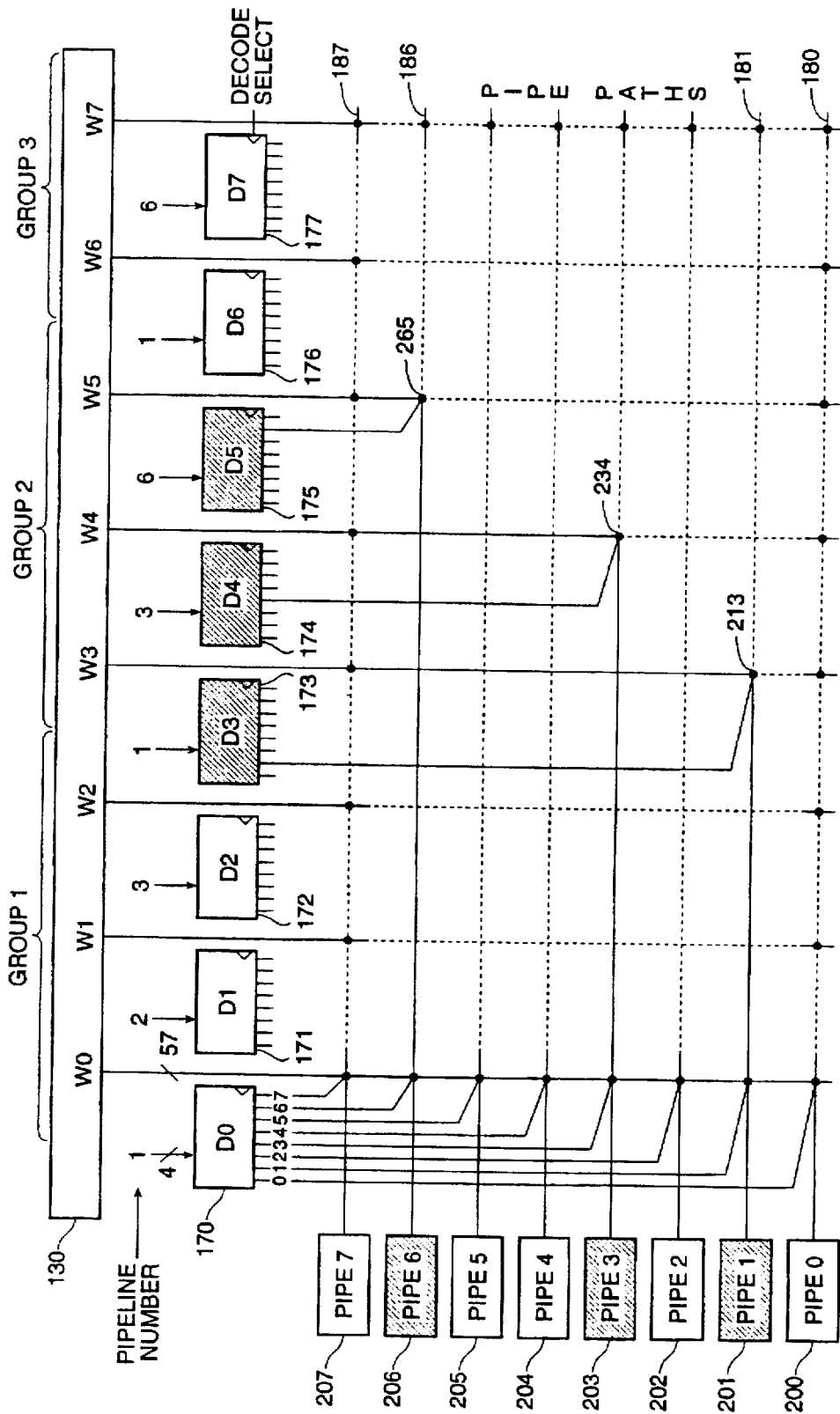
FIG. 2

U.S. Patent

Aug. 11, 1998

Sheet 3 of 5

5,794,003



U.S. Patent

Aug. 11, 1998

Sheet 4 of 5

5,794,003

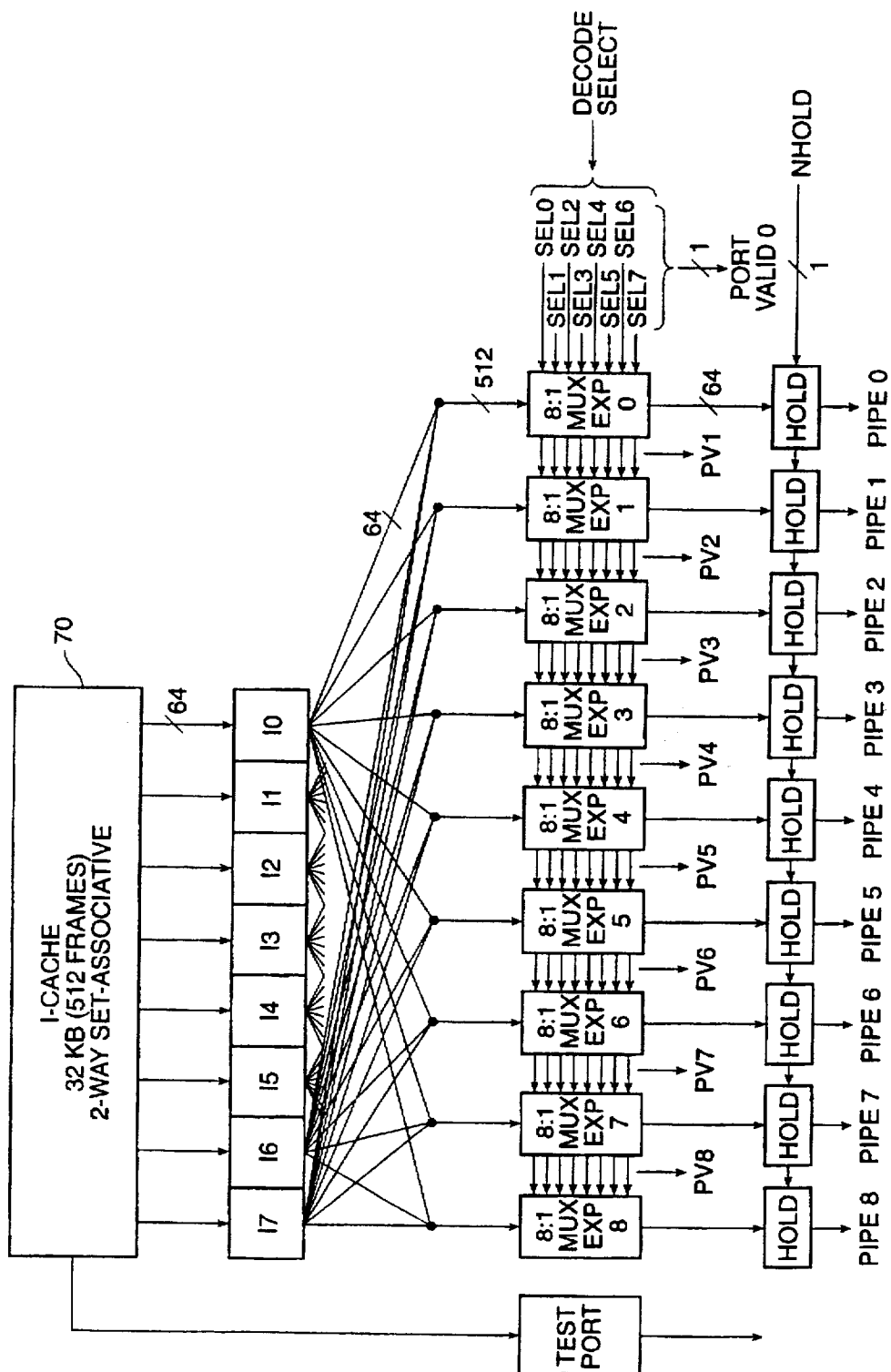


FIG. 4

U.S. Patent

Aug. 11, 1998

Sheet 5 of 5

5,794,003

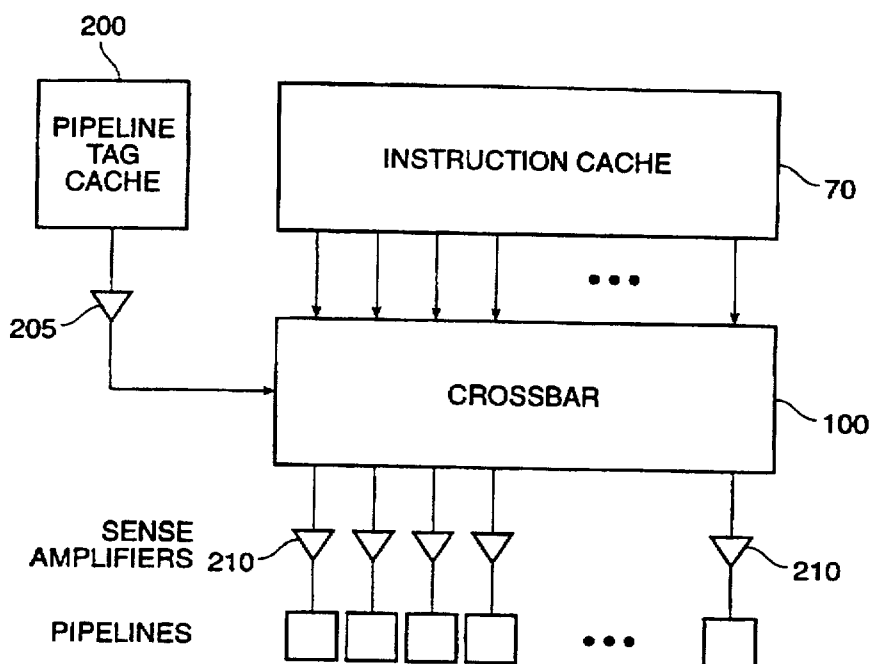


FIG. 5

5,794,003

1

INSTRUCTION CACHE ASSOCIATIVE CROSSBAR SWITCH SYSTEM

This is a Continuation of application No. 08/498,135, filed Jul. 5, 1995, now abandoned; which is a continuation of Ser. No. 08/147,797 filed Nov. 5, 1993, now abandoned, the disclosure of which is incorporated by reference.

BACKGROUND OF THE INVENTION

This invention relates to the architecture of computing systems, and in particular to an architecture in which individual instructions may be executed in parallel, as well as to methods and apparatus for accomplishing that.

A common goal in the design of computer architectures is to increase the speed of execution of a given set of instructions. One approach to increasing instruction execution rates is to issue more than one instruction per clock cycle, in other words, to issue instructions in parallel. This allows the instruction execution rate to exceed the clock rate. Computing systems that issue multiple independent instructions during each clock cycle must solve the problem of routing the individual instructions that are dispatched in parallel to their respective execution units. One mechanism used to achieve this parallel routing of instructions is generally called a "crossbar switch."

In present state of the art computers, e.g. the Digital Equipment Alpha, the Sun Microsystems SuperSparc, and the Intel Pentium, the crossbar switch is implemented as part of the instruction pipeline. In these machines the crossbar is placed between the instruction decode and instruction execute stages. This is because the conventional approach requires the instructions to be decoded before it is possible to determine the pipeline to which they should be dispatched. Unfortunately, decoding in this manner slows system speed and requires extra surface area on the integrated circuit upon which the processor is formed. These disadvantages are explained further below.

SUMMARY OF THE INVENTION

We have developed a computing system architecture that enables instructions to be routed to an appropriate pipeline more quickly, at lower power, and with simpler circuitry than previously possible. This invention places the crossbar switch earlier in the pipeline, making it a part of the initial instruction fetch operation. This allows the crossbar to be a part of the cache itself, rather than a stage in the instruction pipeline. It also allows the crossbar to take advantage of circuit design parameters that are typical of regular memory structures rather than random logic. Such advantages include: lower switching voltages (200-300 millivolts rather than 3-5 volts); more compact design, and higher switching speeds. In addition, if the crossbar is placed in the cache, the need for many sense amplifiers is eliminated, reducing the circuitry required in the system as a whole.

To implement the crossbar switch, the instructions coming from the cache, or otherwise arriving at the switch, must be tagged or otherwise associated with a pipeline identifier to direct the instructions to the appropriate pipeline for execution. In other words, pipeline dispatch information must be available at the crossbar switch at instruction fetch time, before conventional instruction decode has occurred. There are several ways this capability can be satisfied: In one embodiment this system includes a mechanism that routes each instruction in a set of instructions to be executed in parallel to an appropriate pipeline, as determined by a pipeline tag applied to each instruction during compilation,

2

or placed in a separate identifying instruction that accompanies the original instruction. Alternately the pipeline affiliation can be determined after compilation at the time that instructions are fetched from memory into the cache, using a special predecoder unit.

Thus, in one implementation, this system includes a register or other means, for example, the memory cells providing for storage of a line in the cache, for holding instructions to be executed in parallel. Each instruction has associated with it a pipeline identifier indicative of the pipeline to which that instruction is to be issued. A crossbar switch is provided which has a first set of connectors coupled to receive the instructions, and a second set of connectors coupled to the processing pipelines to which the instructions are to be dispatched for execution. Means are provided which are responsive to the pipeline identifiers of the individual instructions in the group supplied to the first set of connectors for routing those individual instructions onto appropriate paths of the second set of connectors, thereby supplying each instruction in the group to be executed in parallel to the appropriate pipeline.

In a preferred embodiment of this invention the associative crossbar is implemented in the instruction cache. By placing the crossbar in the cache all switching is done at low signal levels (approximately 200-300 millivolts). Switching at these low levels is substantially faster than switching at higher levels (5 volts) after the sense amplifiers. The lower power also eliminates the need for large driver circuits, and eliminates numerous sense amplifiers. Additionally by implementing the crossbar in the cache, the layout pitch of the crossbar lines matches the pitch of the layout of the cache.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is a block diagram illustrating a typical environment for a preferred implementation of this invention;

FIG. 2 is a diagram illustrating the overall structure of the instruction cache of FIG. 1;

FIG. 3 is a diagram illustrating one embodiment of the associative crossbar;

FIG. 4 is a diagram illustrating another embodiment of the associative crossbar; and

FIG. 5 is a diagram illustrating another embodiment of the associative crossbar.

DESCRIPTION OF THE SPECIFIC EMBODIMENTS

FIG. 1 is a block diagram of a computer system incorporating the associative crossbar switch according to the preferred embodiment of this invention. The following briefly describes the overall preferred system environment within which the crossbar is incorporated. For additional information about the system, see copending U.S. Patent application Ser. No. 08/147,800, filed Nov. 5, 1995, and entitled "Software Scheduled Superscaler Computer Architecture," which is incorporated by reference herein. FIG. 1 illustrates the organization of the integrated circuit chips by which the computing system is formed. As depicted, the system includes a first integrated circuit 10 that includes a central processing unit, a floating point unit, and an instruction cache.

In the preferred embodiment the instruction cache is a 16 kilobyte two-way set-associative 32 byte line cache. A set associative cache is one in which the lines (or blocks) can be placed only in a restricted set of locations. The line is first

5.794.003

3

mapped into a set, but can be placed anywhere within that set. In a two-way set associative cache, two sets, or compartments, are provided, and each line can be placed in one compartment or the other.

The system also includes a data cache chip 20 that comprises a 32 kilobyte four-way set-associative 32 byte line cache. The third chip 30 of the system includes a predecoder, a cache controller, and a memory controller. The predecoder and instruction cache are explained further below. For the purposes of this invention, the CPU, FPU, data cache, cache controller and memory controller all may be considered of conventional design.

The communication paths among the chips are illustrated by arrows in FIG. 1. As shown, the CPU/FPU and instruction cache chip communicates over a 32 bit wide bus 12 with the predecoder chip 30. The asterisk is used to indicate that these communications are multiplexed so that a 64 bit word is communicated in two cycles. Chip 10 also receives information over 64 bit wide buses 14, 16 from the data cache 20, and supplies information to the data cache 20 over three 32 bit wide buses 18. The predecoder decodes a 32 bit instruction received from the secondary cache into a 64 bit word, and supplies that 64 bit word to the instruction cache on chip 10.

The cache controller on chip 30 is activated whenever a first level cache miss occurs. Then the cache controller either goes to main memory or to the secondary cache to fetch the needed information. In the preferred embodiment the secondary cache lines are 32 bytes and the cache has an 8 kilobyte page size.

The data cache chip 20 communicates with the cache controller chip 30 over another 32 bit wide bus. In addition, the cache controller chip 30 communicates over a 64 bit wide bus 32 with the DRAM memory, over a 128 bit wide bus 34 with a secondary cache, and over a 64 bit wide bus 36 to input/output devices.

As will be described further below, the system shown in FIG. 1 includes multiple pipelines able to operate in parallel on separate instructions which are dispatched to these parallel pipelines simultaneously. In one embodiment the parallel instructions have been identified by the compiler and tagged with a pipeline identification tag indicative of the specific pipeline to which that instruction should be dispatched.

In this system, an arbitrary number of instructions can be executed in parallel. In one embodiment of this system the central processing unit includes eight functional units and is capable of executing eight instructions in parallel. These pipelines are designated using the digits 0 to 7. Also, for this explanation each instruction word is assumed to be 32 bits (4 bytes) long.

As briefly mentioned above, in the preferred embodiment the pipeline identifiers are associated with individual instructions in a set of instructions during compilation. In the preferred embodiment, this is achieved by compiling the instructions to be executed using a well-known compiler technology. During the compilation, the instructions are checked for data dependencies, dependence upon previous branch instructions, or other conditions that preclude their execution in parallel with other instructions. The result of the compilation is identification of a set or group of instructions which can be executed in parallel. In addition, in the preferred embodiment, the compiler determines the appropriate pipeline for execution of an individual instruction. This determination is essentially a determination of the type of instruction provided. For example, load instructions will

4

be sent to the load pipeline, store instructions to the store pipeline, etc. The association of the instruction with the given pipeline can be achieved either by the compiler, or by later examination of the instruction itself, for example, during predecoding.

Referring again to FIG. 1, in normal operation the CPU will execute instructions from the instruction cache according to well-known principles. On an instruction cache miss, however, a set of instructions containing the instruction missed is transferred from the main memory into the secondary cache and then into the primary instruction cache, or from the secondary cache to the primary instruction cache, where it occupies one line of the instruction cache memory. Because instructions are only executed out of the instruction cache, all instructions ultimately undergo the following procedure.

At the time a group of instructions is transferred into the instruction cache, the instruction words are predecoded by the predecoder 30. As part of the predecoding process, a multiple bit field prefix is added to each instruction based upon a tag added to the instruction by the compiler. This prefix gives the explicit pipe number of the pipeline to which that instruction will be routed. Thus, at the time an instruction is supplied from the predecoder to the instruction cache, each instruction will have a pipeline identifier.

It may be desirable to implement the system of this invention on computer systems that already are in existence and therefore have instruction structures that have already been defined without available blank fields for the pipeline information. In this case, in another embodiment of this invention, the pipeline identifier information is supplied on a different clock cycle, then combined with the instructions in the cache or placed in a separate smaller cache. Such an approach can be achieved by adding a "no-op" instruction with fields that identify the pipeline for execution of the instruction, or by supplying the information relating to the parallel instructions in another manner. It therefore should be appreciated that the manner in which the instruction and pipeline identifier arrives at the crossbar to be processed is somewhat arbitrary. I use the word "associated" herein to designate the concept that the pipeline identifiers are not required to have a fixed relationship to the instruction words. That is, the pipeline identifiers need not be embedded within the instructions themselves by the compiler. Instead they may arrive from another means, or on a different cycle.

FIG. 2 is a simplified diagram illustrating the secondary cache, the predecoder, and the instruction cache. This figure, as well as FIGS. 3, 4 and 5, are used to explain the manner in which the instructions tagged with the pipeline identifier are routed to their designated instruction pipelines.

In FIG. 2, for illustration, assume that groups of instructions to be executed in parallel are fetched in a single transfer across a 256 bit (32 byte) wide path from a secondary cache 50 into the predecoder 60. As explained above, the predecoder prefixes the pipeline "P" field to the instruction. After predecoding the resulting set of instructions is transferred into the primary instruction cache 70. At the same time, a tag is placed into the tag field 74 for that line.

In the preferred embodiment the instruction cache operates as a conventional physically-addressed instruction cache. In the example depicted in FIG. 2, the instruction cache will contain 512 bit sets of instructions of eight instructions each, organized in two compartments of 256 lines.

Address sources for the instruction cache arrive at a multiplexer 80 that selects the next address to be fetched.

5.794.003

5

Because preferably instructions are always machine words, the low order two address bits <1:0> of the 32 bit address field supplied to multiplexer 80 are discarded. These two bits designate byte and half-word boundaries. Of the remaining 30 bits, the next three low order address bits <4:2>, which designate a particular instruction word in the set, are sent directly via bus 81 to the associative crossbar. The next low eight address bits <12:5> are supplied over bus 82 to the instruction cache 70 where they are used to select one of the 256 lines in the instruction cache. Finally, the remaining 19 bits of the virtual address <31:13> are sent to the translation lookaside buffer (TLB) 90. The TLB translates these bits into the high 19 bits of the physical address. The TLB then supplies them over bus 84 to the instruction cache. In the cache they are compared with the tag of the selected line, to determine if there is a "hit" or a "miss" in the instruction cache.

If there is a hit in the instruction cache, indicating that the addressed instruction is present in the cache, then the selected set of instructions is transferred across the 512 bit wide bus 73 into the associative crossbar 100. The associative crossbar 100 then dispatches the addressed instructions to the appropriate pipelines over buses 110, 111, . . . , 117. Preferably the bit lines from the memory cells storing the bits of the instruction are themselves coupled to the associative crossbar. This eliminates the need for numerous sense amplifiers, and allows the crossbar to operate on the lower voltage swing information from the cache line directly, without the normally intervening driver circuitry to slow system operation.

FIG. 3 illustrates in more detail one embodiment of the associative crossbar. A 512 bit wide register 130, which represents the memory cells in a line of the cache (or can be a physically separate register), contains at least the set of instructions capable of being issued. For the purposes of illustration, register 130 is shown as containing up to eight instruction words W0 to W7. Using means described in the copending application referred to above, the instructions have been sorted into groups for parallel execution. For illustration here, assume the instructions in Group 1 are to be dispatched to pipelines 1, 2 and 3; the instructions in Group 2 to pipelines 1, 3 and 6; and the instructions in Group 3 to pipelines 1 and 6. The decoder select signal enables only the appropriate set of instructions to be executed in parallel, essentially allowing register 130 to contain more than just one set of instructions. Of course, by only using register 130 only for one set of parallel instructions at a time, the decoder select signal is not needed.

As shown in FIG. 3, the crossbar switch itself consists of two sets of crossing pathways. In the horizontal direction are the pipeline pathways 180, 181, . . . , 187. In the vertical direction are the instruction word paths, 190, 191, . . . , 197. Each of these pipeline and instruction pathways is themselves a bus for transferring the instruction word. Each horizontal pipeline pathway is coupled to a pipeline execution unit 200, 201, 202, . . . , 207. Each of the vertical instruction word pathways 190, 191, . . . , 197 is coupled to an appropriate portion of register or cache line 130.

The decoders 170, 171, . . . , 177 associated with each instruction word pathway receive the 4 bit pipeline code from the instruction. Each decoder, for example decoder 170, provides eight 1 bit control lines as output. One of these control lines is associated with each pipeline pathway crossing of that instruction word pathway. Selection of a decoder as described with reference to FIG. 3 activates the output bit control line corresponding to that input pipe number. This signals the crossbar to close the switch between the word

6

path associated with that decoder and the pipe path selected by that bit line. Establishing the cross connection between these two pathways causes a selected instruction word to flow into the selected pipeline. For example, decoder 173 has received the pipeline bits for word W3. Word W3 has associated with it pipeline path 1. The pipeline path 1 bits are decoded to activate switch 213 to supply instruction word W3 to pipeline execution unit 201 over pipeline path 181. In a similar manner, the identification of pipeline path 3 for decoder D4 activates switch 234 to supply instruction word W4 to pipeline path 3. Finally, the identification of pipeline 6 for word W5 in decoder D5 activates switch 265 to transfer instruction word W5 to pipeline execution unit 206 over pipeline pathway 186. Thus, instructions W3, W4 and W5 are executed by pipes 201, 203 and 206, respectively.

The pipeline processing units 200, 201, . . . , 207 shown in FIG. 3 can carry out desired operations. In a preferred embodiment of the invention, each of the eight pipelines first includes a sense amplifier to detect the state of the signals on the bit lines from the crossbar. In one embodiment the pipelines include first and second arithmetic logic units; first and second floating point units; first and second load units; a store unit and a control unit. The particular pipeline to which a given instruction word is dispatched will depend upon hardware constraints as well as data dependencies.

FIG. 4 is a diagram illustrating another embodiment of the associative crossbar. In FIG. 4 nine pipelines 0-8 are shown coupled to the crossbar. The decode select is used to enable a subset of the instructions in the register 130 for execution just as in the system of FIG. 3.

The execution ports that connect to the pipelines specified by the pipeline identification bits of the enabled instructions are then selected to multiplex out the appropriate instructions from the contents of the register. If one or more of the pipelines is not ready to receive a new instruction, a set of hold latches at the output of the execution ports prevents any of the enabled instructions from issuing until the "busy" pipeline is free. Otherwise the instructions pass transparently through the hold latches into their respective pipelines. Accompanying the output of each port is a "port valid" signal that indicates whether the port has valid information to issue to the hold latch.

FIG. 5 illustrates an alternate embodiment for the invention where pipeline tags are not included with the instruction, but are supplied separately, or where the cache line itself is used as the register for the crossbar. In these situations, the pipeline tags may be placed into a high speed separate cache memory 200. The output from this memory can then control the crossbar in the same manner as described in conjunction with FIG. 3. This approach eliminates the need for sense amplifiers between the instruction cache and the crossbar. This enables the crossbar to switch very low voltage signals more quickly than higher level signals, and the need for hundreds of sense amplifiers is eliminated. To provide a higher level signal for control of the crossbar, sense amplifier 205 is placed between the pipeline tag cache 200 and the crossbar 100. Because the pipeline tag cache is a relatively small memory, however, it can operate more quickly than the instruction cache memory, and the tags therefore are available in time to control the crossbar despite the sense amplifier between the cache 200 and the crossbar 100. Once the switching occurs in the crossbar, then the signals are amplified by sense amplifiers 210 before being supplied to the various pipelines for execution.

The architecture described above provides many unique advantages to a system using this crossbar. The crossbar

5.794.003

7

described is extremely flexible, enabling instructions to be executed sequentially or in parallel, depending entirely upon the "intelligence" of the compiler. Importantly, the associative crossbar relies upon the content of the message being decoded, not upon an external control circuit acting independently of the instructions being executed. In essence, the associative crossbar is self directed.

Another important advantage of this system is that it allows for more intelligent compilers. Two instructions which appear to a hardware decoder (such as in the prior art described above) to be dependent upon each other can be determined by the compiler not to be interdependent. For example, a hardware decoder would not permit two instructions $R1+R2=R3$ and $R3+R5=R6$ to be executed in parallel. A compiler, however, can be "intelligent" enough to determine that the second $R3$ is a previous value of $R3$, not the one calculated by $R1+R2$, and therefore allow both instructions to issue at the same time. This allows the software to be more flexible and faster.

Although the foregoing has been a description of the preferred embodiment of the invention, it will be apparent to those of skill in the art the numerous modifications and variations may be made to the invention without departing from the scope as described herein. For example, arbitrary numbers of pipelines, arbitrary numbers of decoders, and different architectures may be employed, yet rely upon the system we have developed.

I claim:

1. A computing system comprising:

means for forming groups of software-scheduled instructions, software-scheduled instructions within each of the groups executable in parallel; and

a super-scaler cache for routing each of the software-scheduled instructions within the groups to be executed in parallel to an appropriate instruction pipeline of a plurality of instruction pipelines, the super-scaler cache comprising:

super-scaler storage for holding one group of the groups of software-scheduled instructions, each software-scheduled instruction within the one group having embedded therein an instruction pipeline identifier of a plurality of instruction pipeline identifiers;

an associative crossbar having a first set of connectors coupled to the super-scaler storage for receiving each of the software-scheduled instructions therefrom, and a second set of connectors coupled to the plurality of instruction pipelines; and

means responsive to the instruction pipeline identifier of each of the software-scheduled instructions, for coupling appropriate connectors of the first set of connectors to appropriate connectors of the second set of connectors, to thereby supply each of the software-scheduled instructions to the appropriate instruction pipeline for parallel execution.

2. The computing system of claim 1 wherein:

the first set of connectors comprises a set of first communication buses, one first communication bus for each of the software-scheduled instructions in the super-scaler storage;

the second set of connectors comprises a set of second communication buses, one second communication bus for each of the plurality of instruction pipelines; and

the means responsive to the plurality of instruction pipeline identifiers comprising:

a set of selectors coupled to the super-scaler storage to receive as first input signals the instruction pipeline

8

identifier of each of the software-scheduled instructions and in response thereto supply as output signals switch control signals for each of the software-scheduled instructions; and

a set of switches, coupled to the set of selectors, one switch located at each intersection of each of the first communication buses with each of the second communication buses, the set of switches providing connections in response to the switch control signals to thereby supply each of the software-scheduled instructions in parallel to the appropriate instruction pipelines.

3. A computing system comprising:

means for assembling sets of software-scheduled instructions to be executed in parallel; and

a super-scaler cache for routing each of the software-scheduled instructions in a group to be executed in parallel, to an appropriate instruction pipeline of a plurality of instruction pipelines, the super-scaler cache comprising:

a super-scaler storage for holding the sets of software-scheduled instructions, including at least a set of software-scheduled instructions, and a set of instruction pipeline identifiers, each individual instruction of the first set of software-scheduled instructions having associated therewith an instruction pipeline identifier of the set of instruction pipeline identifiers; an associative crossbar having a first set of connectors coupled to the super-scaler storage for receiving the set of software-scheduled instructions therefrom and a second set of connectors coupled to the plurality of instruction pipelines;

selection means connected to receive the instruction pipeline identifiers of the set of instruction pipeline identifiers, the selection means for supplying in response thereto output signals; and

switching means coupled to receive the output signals for selectively connecting connectors of the first set of connectors to connectors of the second set of connectors to thereby supply each software-scheduled instruction in the set of software-scheduled instructions to be executed in parallel to the appropriate instruction pipeline.

4. The computing system of claim 3 wherein

the first set of connectors comprises a set of first communication buses, one of the first communication buses for each of the software-scheduled instructions in the set of software-scheduled instructions in the super-scaler storage;

the second set of connectors comprises a set of second communication buses, one of the second communication buses for each of the plurality of instruction pipelines;

the selection means comprises a set of selectors coupled to receive the instruction pipeline identifiers and in response thereto supply as the output signals a plurality of switch control signals; and

the switching means includes a set of switches, one of the switches at each intersection of each of the first set of communication buses with the second set of communication buses, the set of switches providing connections in response to receiving the plurality of switch control signals to thereby supply each of the software-scheduled instructions of the set of software-scheduled instructions to the appropriate instruction pipeline.

5. The computing system of claim 4 wherein each selector in the set of selectors comprise a multiplexer.

5,794,003

9

6. In a computing system, a method for transferring software-scheduled instructions to be executed through an associative crossbar switch in a super-scaler cache, the associative crossbar switch having a first set of connectors coupled to a super-scaler storage in the super-scaler cache for receiving each of the software-scheduled instructions therefrom and a second set of connectors coupled to a plurality of instruction pipelines, the method comprising the steps of:

forming groups of software-scheduled instructions, software-scheduled instructions within each group executable in parallel;

storing in the super-scaler storage in the super-scaler cache one group of the groups of software-scheduled instructions to be executed in parallel, each software-scheduled instruction in the one group having embedded therein an instruction pipeline identifier of a plurality of instruction pipeline identifiers; and

using the instruction pipeline identifier of each of the software-scheduled instructions to control switches in the associative crossbar switch in the super-scaler cache between the first set of connectors and the second set of connectors to thereby supply each of the software-scheduled instructions to an appropriate instruction pipeline.

7. The method of claim 6 wherein the step of using the instruction pipeline identifier comprises:

supplying the instruction pipeline identifiers of each of the software-scheduled instructions in the one group of software-scheduled instructions to a corresponding number of selectors, each of the selectors providing an output signal indicative of the instruction pipeline identifier; and

using the output signal of each of the selectors to control the switches between the first set of connectors and the second set of connectors to thereby supply each of the software-scheduled instructions to the appropriate instruction pipeline.

8. An apparatus for routing software-scheduled instruction words to a plurality of instruction pipelines, the apparatus comprising:

means for forming groups of software-scheduled instruction words, software-scheduled instruction words in each group executable in parallel; and

a very long instruction word cache for storing a group of software-scheduled instruction words, each of the software-scheduled instruction words in the group having embedded therein a unique instruction pipeline identifier, the very long instruction word cache further comprising:

an associative crossbar having a plurality of connectors coupled to the plurality of instruction pipelines, for selecting which of the plurality of connectors to couple to which of the plurality of instruction pipelines in response to switch selection signals;

a selector for asserting output signals in response to the instruction pipeline identifiers embedded with each of the group of software-scheduled instruction words; and

a switch coupled to the associative crossbar, and to the selector, for asserting switch selection signals to the associative crossbar in response to the output signals.

9. An apparatus for routing software-scheduled instruction words to a plurality of instruction pipelines, the apparatus comprising:

means for assembling a group of software-scheduled instruction words, software-scheduled instruction words in the group executable in parallel; and

10

an very long instruction word cache for storing the group of software-scheduled instruction words and a group of instruction pipeline identifiers, the group of instruction pipeline identifiers associated with the group of software-scheduled instruction words, the very long instruction word cache further comprising:

an associative crossbar including a plurality of connectors coupled to the plurality of instruction pipelines, for selecting which of the plurality of connectors to couple to which of the plurality of instruction pipelines in response to switch selection signals;

a selector for asserting output signals in response to the instruction pipeline identifiers associated with each of the group of software-scheduled instruction words; and

a switch coupled to the associative crossbar, and to the selector, for asserting switch selection signals to the associative crossbar in response to the output signals.

10. The computing system of claim 2, wherein the instruction pipeline identifier embedded within each software-scheduled instruction is determined by a compiler.

11. The computing system of claim 1, wherein the means for forming the groups of software-scheduled instructions comprises a compiler.

12. The computing system of claim 3, wherein the means for assembling the sets of software-scheduled instructions comprises a compiler.

13. The computing system of claim 4, wherein the instruction pipeline identifiers of the set of instruction pipeline identifiers are determined by a compiler.

14. The method of claim 6, wherein the step of forming groups of software-scheduled instructions comprises using a compiler.

15. The method of claim 7, wherein the instruction pipeline identifiers in the one group of instruction pipeline identifiers are determined by a compiler.

16. The apparatus of claim 8, wherein the means for forming groups of software-scheduled instruction words comprises a compiler.

17. The apparatus of claim 8, wherein the instruction pipeline identifiers of the group of instruction pipeline identifiers are determined by a compiler.

18. The apparatus of claim 9, wherein the software-scheduled instructions of the group of software-scheduled instructions are determined by a compiler.

19. The apparatus of claim 9, wherein the means for assembling a group of software-scheduled instruction words comprises a compiler.

20. In a computing system in which groups of software-scheduled instructions are formed and in which software-scheduled instructions within each of the groups are executable in parallel, a super-scaler cache for routing each of the software-scheduled instructions within the groups to be executed in parallel to an appropriate instruction pipeline of a plurality of instruction pipelines, the super-scaler cache comprising:

super-scaler storage for holding one group of the groups of software-scheduled instructions, each software-scheduled instruction within the one group having embedded therein an instruction pipeline identifier of a plurality of instruction pipeline identifiers;

an associative crossbar having a first set of connectors coupled to the super-scaler storage for receiving each of the software-scheduled instructions therefrom, and a second set of connectors coupled to the plurality of instruction pipelines; and

means responsive to the instruction pipeline identifier of each of the software-scheduled instructions, for cou-

5,794,003

11

pling appropriate connectors of the first set of connectors to appropriate connectors of the second set of connectors, to thereby supply each of the software-scheduled instructions to the appropriate instruction pipeline for parallel execution;

wherein the instruction pipeline identifier embedded within each software-scheduled instruction is determined by a compiler.

21. The super-scalar cache of claim 20 wherein:

the first set of connectors comprises a set of first communication buses, one first communication bus for each of the software-scheduled instructions in the super-scalar storage;

the second set of connectors comprises a set of second communication buses, one second communication bus for each of the plurality of instruction pipelines; and the means responsive to the plurality of instruction pipeline identifiers comprising:

a set of selectors coupled to the super-scalar storage to receive as first input signals the instruction pipeline identifier of each of the software-scheduled instructions and in response thereto supply as output signals switch control signals for each of the software-scheduled instructions; and

a set of switches, coupled to the set of selectors, one switch located at each intersection of each of the first communication buses with each of the second communication buses, the set of switches providing connections in response to the switch control signals to thereby supply each of the software-scheduled instructions in parallel to the appropriate instruction pipelines.

22. In a computing system in which sets of software-scheduled instructions are executable in parallel, a super-scalar cache for routing each of the software-scheduled instructions in a group to be executed in parallel, to an appropriate instruction pipeline of a plurality of instruction pipelines, the super-scalar cache comprising:

a super-scalar storage for holding the sets of software-scheduled instructions, including at least a set of software-scheduled instructions, and a set of instruction pipeline identifiers, each individual instruction of the first set of software-scheduled instructions having associated therewith an instruction pipeline identifier of the set of instruction pipeline identifiers;

an associative crossbar having a first set of connectors coupled to the super-scalar storage for receiving the set of software-scheduled instructions therefrom and a second set of connectors coupled to the plurality of instruction pipelines;

selection means connected to receive the instruction pipeline identifiers of the set of instruction pipeline identifiers, the selection means for supplying in response thereto output signals; and

switching means coupled to receive the output signals for selectively connecting connectors of the first set of connectors to connectors of the second set of connectors to thereby supply each software-scheduled instruction in the set of software-scheduled instructions to be executed in parallel to the appropriate instruction pipeline;

wherein the instruction pipeline identifiers of the set of instruction pipeline identifiers are determined by a compiler.

23. The super-scalar cache of claim 22 wherein the first set of connectors comprises a set of first communication buses, one of the first communication buses

12

for each of the software-scheduled instructions in the set of software-scheduled instructions in the super-scalar storage;

the second set of connectors comprises a set of second communication buses, one of the second communication buses for each of the plurality of instruction pipelines;

the selection means comprises a set of selectors coupled to receive the instruction pipeline identifiers and in response thereto supply as the output signals a plurality of switch control signals; and

the switching means includes a set of switches, one of the switches at each intersection of each of the first set of communication buses with the second set of communication buses, the set of switches providing connections in response to receiving the plurality of switch control signals to thereby supply each of the software-scheduled instructions of the set of software-scheduled instructions to the appropriate instruction pipeline.

24. The super-scalar cache of claim 23 wherein each selector in the set of selectors comprise a multiplexer.

25. In a computing system in which groups of software-scheduled instructions are executable in parallel, a method for transferring each software-scheduled instruction in a group to be executed through an associative crossbar switch in a super-scalar cache, the associative crossbar switch having a first set of connectors coupled to a super-scalar storage in the super-scalar cache for receiving each of the software-scheduled instructions therefrom and a second set of connectors coupled to a plurality of instruction pipelines, the method comprising:

storing in the super-scalar storage in the super-scalar cache one group of the groups of software-scheduled instructions to be executed in parallel, each software-scheduled instruction in the one group having embedded therein an instruction pipeline identifier of a plurality of instruction pipeline identifiers; and

using the instruction pipeline identifier of each of the software-scheduled instructions to control switches in the associative crossbar switch in the super-scalar cache between the first set of connectors and the second set of connectors to thereby supply each of the software-scheduled instructions to an appropriate instruction pipeline;

wherein the instruction pipeline identifiers in the one group of instruction pipeline identifiers are determined by a compiler.

26. The method of claim 25 wherein the step of using the instruction pipeline identifier comprises:

supplying the instruction pipeline identifiers of each of the software-scheduled instructions in the one group of software-scheduled instructions to a corresponding number of selectors, each of the selectors providing an output signal indicative of the instruction pipeline identifier; and

using the output signal of each of the selectors to control the switches between the first set of connectors and the second set of connectors to thereby supply each of the software-scheduled instructions to the appropriate instruction pipeline.

27. In a computing system in which groups of software-scheduled instructions are executed in parallel by parallel processors, an apparatus for routing groups of software-scheduled instruction words to a plurality of instruction pipelines, the apparatus comprising:

a very long instruction word cache for storing a group of software-scheduled instruction words, each of the

5.794.003

13

software-scheduled instruction words in the group having embedded therein a unique instruction pipeline identifier, the very long instruction word cache further comprising:

an associative crossbar having a plurality of connectors 5
coupled to the plurality of instruction pipelines, for selecting which of the plurality of connectors to couple to which of the plurality of instruction pipelines in response to switch selection signals;

a selector for asserting output signals in response to the 10
instruction pipeline identifiers embedded with each of the group of software-scheduled instruction words; and

a switch coupled to the associative crossbar, and to the 15
selector, for asserting switch selection signals to the associative crossbar in response to the output signals;

wherein the instruction pipeline identifiers of the group of instruction pipeline identifiers are determined by a compiler.

28. In a computing system in which a group of software- 20
scheduled instructions is executed in parallel by parallel processors, an apparatus for routing the group of software-scheduled instruction words to a plurality of instruction pipelines, the apparatus comprising:

an very long instruction word cache for storing the group 25
of software-scheduled instruction words and a group of instruction pipeline identifiers, the group of instruction pipeline identifiers associated with the group of software-scheduled instruction words, the very long instruction word cache further comprising:

an associative crossbar including a plurality of connec- 30
tors coupled to the plurality of instruction pipelines.

14

for selecting which of the plurality of connectors to couple to which of the plurality of instruction pipelines in response to switch selection signals;

a selector for asserting output signals in response to the instruction pipeline identifiers associated with each of the group of software-scheduled instruction words; and

a switch coupled to the associative crossbar, and to the selector, for asserting switch selection signals to the associative crossbar in response to the output signals;

wherein the instruction pipeline identifiers of the group of instruction pipeline identifiers are determined by a compiler.

29. The super-scaler cache of claim 20, wherein the software-scheduled instructions of the one group are determined by a compiler.

30. The super-scaler cache of claim 22, wherein the software-scheduled instructions of the set of software-scheduled instructions are determined by a compiler.

31. The method of claim 25, wherein the software-scheduled instructions in the one group of software-scheduled instructions are determined by a compiler.

32. The very long instruction word cache of claim 27, 25
wherein the software-scheduled instructions of the group of software-scheduled instructions are determined by a compiler.

33. The apparatus of claim 28, wherein the software- 30
scheduled instructions of the group of software-scheduled instructions are determined by a compiler.

* * * * *

UNITED STATES PATENT AND TRADEMARK OFFICE
CERTIFICATE OF CORRECTION

PATENT NO. : 5,794,003
DATED : August 11, 1998
INVENTOR(S) : Howard G. Sachs

Page 1 of 1

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

Column 2,

Line 56, delete " filed November 5, 1995" and insert -- filed November 5, 1993 --.

Signed and Sealed this

Twenty-third Day of October, 2001

Attest:

Nicholas P. Godici

Attesting Officer

NICHOLAS P. GODICI
Acting Director of the United States Patent and Trademark Office



US006360313B1

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

(10) Patent No.: **US 6,360,313 B1**
(45) Date of Patent: ***Mar. 19, 2002**

(54) **INSTRUCTION CACHE ASSOCIATIVE CROSSBAR SWITCH**

(75) Inventors: **Howard G. Sachs, Los Altos; Siamak Arya, Cupertino, both of CA (US)**

(73) Assignee: **Intergraph Corporation, Huntsville, AL (US)**

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

This patent is subject to a terminal disclaimer.

5,121,502 A 6/1992 Rau et al.
5,129,067 A 7/1992 Johnson
5,151,981 A 9/1992 Westcott et al.
5,179,680 A 1/1993 Colwell et al.
5,197,137 A 3/1993 Kumar et al.
5,203,002 A 4/1993 Wetzel
5,214,763 A 5/1993 Blaner et al.

(List continued on next page.)

FOREIGN PATENT DOCUMENTS

EP 0 363 222 A2 4/1990
EP 0 426 393 A2 5/1991
EP 0 449 661 A2 10/1991
EP 0463296 A2 1/1992

(List continued on next page.)

OTHER PUBLICATIONS

Adams et al., "HARP: A Statically Scheduled Multiple-Instruction-Issue Architecture and its Compiler", *Technical Report 163*, University of Hertfordshire, Hatfield, Herts UK, pp. 1-8, Sep. 1993.

Agerwala et al., "High Performance Reduced Instruction Set Processors", RC 12434 (#55845), *Computer Science*, Jan. 9, 1987.

(List continued on next page.)

Primary Examiner—Viet D. Vu

(74) Attorney, Agent, or Firm—Townsend&Townsend&Crew

(57) **ABSTRACT**

A computing system as described in which individual instructions are executable in parallel by processing pipelines, and instructions to be executed in parallel by different pipelines are supplied to the pipelines simultaneously. The system includes storage for storing an arbitrary number of the instructions to be executed. The instructions to be executed are tagged with pipeline identification tags indicative of the pipeline to which they should be dispatched. The pipeline identification tags are supplied to a system which controls a crossbar switch, enabling the tags to be used to control the switch and supply the appropriate instructions simultaneously to the differing pipelines.

122 Claims, 5 Drawing Sheets

Related U.S. Application Data

(63) Continuation of application No. 09/057,861, filed on Apr. 9, 1998, which is a continuation of application No. 08/754,337, filed on Nov. 22, 1996, now Pat. No. 5,794,003, which is a continuation of application No. 08/498,135, filed on Jul. 5, 1995, now abandoned, which is a continuation of application No. 08/147,797, filed on Nov. 5, 1993, now abandoned.

(51) Int. Cl.⁷ **G06F 9/38**

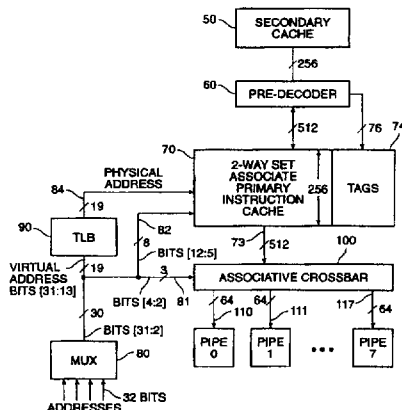
(52) U.S. Cl. **712/215; 712/24; 712/207; 712/213; 712/234**

(58) Field of Search **712/23, 24, 208, 712/210, 211, 212, 213, 214, 215, 234, 207; 395/706**

(56) **References Cited**

U.S. PATENT DOCUMENTS

4,437,149 A 3/1984 Pomerene et al.
4,833,599 A 5/1989 Colwell et al.
4,847,755 A 7/1989 Morrison et al.
4,933,837 A 6/1990 Freidin
5,055,997 A 10/1991 Sluijter et al.
5,057,837 A 10/1991 Colwell et al.
5,081,575 A 1/1992 Hiller et al.
5,101,341 A 3/1992 Circello et al.



EXHIBIT

C

US 6,360,313 B1

Page 2

U.S. PATENT DOCUMENTS

5,226,169 A	7/1993	Gregor	
5,233,696 A	8/1993	Suzuki	
5,239,654 A	8/1993	Ing-Simmons et al.	
5,297,255 A	3/1994	Hamanaka et al.	
5,297,281 A	3/1994	Emma et al.	
5,299,321 A	3/1994	Iizuka	
5,333,280 A *	7/1994	Ishikawa et al.	712/241
5,337,415 A	8/1994	DeLano et al.	
5,355,460 A	10/1994	Eickemeyer et al.	
5,367,694 A	11/1994	Ueno	
5,442,760 A *	8/1995	Rustad et al.	712/215
5,459,844 A	10/1995	Eickemeyer et al.	
5,500,942 A	3/1996	Eickemeyer et al.	
5,761,470 A	6/1998	Yoshida	
5,819,088 A	10/1998	Reinders	
5,922,065 A	7/1999	Hull	

FOREIGN PATENT DOCUMENTS

EP	0 463 299 A2	1/1992
EP	0 496 928 A2	8/1992
EP	0 652 510 A2	5/1995
WO	WO 98/38791	9/1998

OTHER PUBLICATIONS

- Allison, DEC 7000/10000 Model 600 AXP Multiprocessor Server, *IEEE*, 1063-6390/93, pp. 456-464 (1993).
- Anderson, D.W. et al. [1967] "The IBM 360 model 91: Processor Philosophy and instruction handling." IBM J. Research and Development 11:1 (Jan.) pp. 8-24.
- Arya et al., "An Architecture for High Instruction Level Parallelism", pp. 1-21, Jan. 1995.
- Bakoglu et al., "The IBM RISC system/6000 processor: hardware overview", *IBM J. Res. Develop.*, 34(1):12-22 (Jan., 1990).
- Beck et al., "The Cydra 5 Minisupercomputer: Architecture and Implementation", *J. Supercomputing*, 7:143-179 (1993).
- Butler, et al., "Single Instruction Stream Parallelism Is Greater than Two", 1991 ACM, pp. 276-286.
- Case et al., "DEC Enters Microprocessor Business With Alpha", *Microprocessor Report*, 6(3):1,6-14, (Mar. 4, 1992).
- Chang et al., "Comparing Static and Dynamic Code Scheduling for Multiple-Instruction-Issue Processors", *Proceedings of the 24th International Symposium on Microarchitectures—MICRO24*, pp. 1-9, 1991.
- Charlesworth, A.E., [1981]. "An approach to scientific processing: The architecture design of the AP-120B/FPS-164 family" *Computer* 14:9 (Sep.), pp. 18-27.
- Chen, "The Effect of Code Expanding Optimizations on Instruction Cache Design", *IEEE Transactions on Computers*, 42(9) pp. 1045-1057, Sep. 1993.
- Colwell, et al., "A VLIW Architecture for a Trace Scheduling Compiler", *IEEE Transactions on Computers*, 37(8) pp. 967-979, Aug. 1988.
- Colwell, R.P. et al. [1987]. "A VLIW architecture for a trace scheduling compiler." Proc. second Conf. on Architectural Support for Programming Languages and Operating Systems, *IEFF/ACM* (Mar.), pp. 180-192.
- Conte, "Trade-Offs in Processor/Memory Interfaces for Superscalar Processors", *MICRO-25, The 25th Annual International Symposium on Microarchitecture*, Dec. 1992.
- De Gloria et al., "A Programmable Instruction Format Extension to VLIW Architectures", *Proceedings Comp. Euro. 1992*, pp. 35-40 (May 4, 1992).
- Dehnert et al., "Compiling for the Cydra 5", *J. Supercomputing*, 7, pp. 181-227, May 1993.
- Dehnert, J.C. et al. [1989]. "Overlapped loop support on the Cydra 5." Proc. Third Conf. on Architectural Support for Programming Languages and Operating Systems (Apr.), *TEEE/ACM*. Boston, pp. 26-39.
- Dorozhevets et al., The El-Brus-3 and Mars-M: Recent Advances in Russian High-Performance Computing, *Journal of Supercomputing*, 6(1):5-48 (Mar. 1, 1992).
- Dutton, "The Design of the DEC 3000 Model 500 AXP Workstation", *IEEE*, 1063-6390/93, pp. 449-455 (1993).
- Fisher et al., "Instruction-Level Parallel Processing", *Science*, 253, pp. 1233-1241, Sep. 1991.
- Fisher et al., "Parallel Processing: A Smart Compiler and a Dumb Machine", *SIGPLAN Notices*, 19(6):37-47 (Jun. 1984).
- Fisher et al., "Parallel Processing: A Smart Compiler and a Dumb Machine", *ACM-Sigplan 84 Compiler Construction Conference*, 19(6), Jun. 1984.
- Fisher, "Trace Scheduling: A Technique for Global Microcode Compaction", *IEEE Transactions on Computers*, C-30(7):478-490 Jul. 1981.
- Fisher, "Very Long Instruction Word Architectures and the ELI-512", *Proceedings of the 10th Symposium on Computer Architecture*, ACM Press, pp. 140-150 (1983).
- Gee et al., "Cache Performance of the SPEC92 Benchmark Suite", *IEEE MICRO*, pp. 17-27, Aug. 1993.
- Gray et al., "Static Instruction Scheduling for the HARP Multiple-Instruction-Issue Architecture", *Technical Report 142*, University of Hertfordshire, Hatfield, Herts UK, Oct. 1992.
- Grove et al., "GEM Optimizing Compilers for Alpha AXP Systems", *IEEE*, 1063-6390/93, pp. 464-473 (1993).
- Gwennap, "Visionaries See Beyond Superscalar", *Microprocessor Report*, pp. 18-19, Dec. 6, 1993.
- Hennessy et al., "Computer Architecture; a Quantitative Approach", ISBN 1-55880-069-8, Morgan Kaufmann Publishers, Inc., San Mateo Calif. (1990) Table of Contents, pp. xi-xv.
- Hennessy et al., "Computer Technology and Architecture: An Evolving Interaction", *IEEE Computer*, pp. 18-29, Sep. 1991.
- Hsu et al., "Highly Concurrent Scalar Processing", *13th International Symposium on Computer Architecture*, pp. Tokyo, 1986, pp. 1-10.
- Johnson, "Superscalar Microprocessor Design", Prentice-Hall 1991 pp. 233-235.
- Johnson, "Superscalar Microprocessor Design", Prentice-Hall, Englewood Cliffs, New Jersey, 1991.
- Karl, "Some Design Aspects for VLIW Architectures Exploiting Fine-Grained Parallelism", *Proceedings of the 5th International PARLE Conference*, pp. 582-599, Jun. 1993.
- Kato et al., "Delayed Instruction Execution on a Long Instruction Word (LIW) Computer", *Systems & Computers in Japan*, 23(14):13-22 (Jan. 1, 1992).
- Lam et al., "Limits of Control Flow on Parallelism", *Computer Architecture News*, 20(2):46-57 (1992).

US 6,360,313 B1

Page 3

- Lam, "Software Pipelining: An Effective Scheduling Technique for VLIW Machines", *Proceedings of ACM SIGPLAN '88 Conference on Programming Language Design and Implementation*, pp. 318-328, Jun. 1988.
- Mahlke et al., "Effective Compiler Support for Predicated Execution Using the Hyperblock", MICRO 25, *Proceedings of the 25th Annual International Symposium on Microarchitectures*, IEEE Computer Society Press, pp. 45-54, Dec. 1992.
- Mahlke et al., "Sentinel Scheduling for VLIW and Superscalar Processors", *In Proceedings of ASPLOS V*, 27(9) pp. 238-247, Sep. 1992.
- Minagawa et al., "Pre-decoding Mechanism for Superscalar Architecture", *IEEE Pacific Rim Conference on Communications, Computers and Signal Processing*, pp. 21-24 (May 9, 1991).
- Moon et al., "An Efficient Resource-Constrained Global Scheduling Technique for Superscalar and VLIW Processors", MICRO 25, *Proceedings of the 25th Annual International Symposium on Microarchitectures*, pp. 55-71, Dec. 1992.
- Nicolau et al., "Measuring the Parallelism Available for Very Long Instruction Word Architectures", *IEEE Transactions on Computers*, C-33(11), pp. 968-976, Nov. 1984.
- Oyang et al., "A Cost Effective Approach to Implement A Long Instruction Word Microprocessor", *Computer Architecture News*, 18(1), Mar. 1990, pp. 59-72.
- Pan et al., "Improving the Accuracy of Dynamic Branch Prediction Using Branch Correlation", *Fifth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-V)*, vol. 27, pp. 76-84 (1992).
- Park et al., "On Predicated Execution", *Technical Report HPL-91-58*, Hewlett-Packard Laboratories May 1991.
- Rau et al., "Efficient Code Generation for Horizontal Architectures: Compiler Techniques and Architectural Support", *Proceedings of the Ninth Annual International Symposium on Computer Architecture*, pp. 131-139, 1982.
- Rau et al., "Instruction-Level Parallel Processing: History, Overview and Perspective", *J. Supercomputing*, vol. 7, pp. 9-50 (1993).
- Rau, "Cydra™ 5 Directed Dataflow Architecture", *Proceedings of COMPCON* 1988.
- Rau, "Dynamic Scheduling Techniques for VLIW Processors", *Technical Report HPL-93-52*, Hewlett-Packard Laboratories, Jun. 1993.
- Rau, B.R., et al. [1989]. "The Cydra 5 departmental supercomputer: Design philosophies, decisions, and tradeoffs," *IEEE Computers*, 22:1 (Jan.), pp. 12-34.
- Rau, et al. (Editors), "Instruction-Level Parallelism", reprint from *J. Supercomputing*, 7(1/2), 1993.
- Schuette et al., "Instruction-Level Experimental Evaluation of the Multiflow Trace 14/300 VLIW Computer", *J. Supercomputing*, vol. 7, pp. 249-271 (1993).
- Silberman et al., "An Architectural Framework for Supporting Heterogeneous Instruction-Set Architectures", *IEEE Computer*, 26(6), pp. 39-56, Jun. 1993.
- Sites (Editor), "Alpha Architecture Reference Manual", Digital Press 1992.
- Smith et al., "Boosting Beyond Static Scheduling in a Superscalar Processor", *IEEE Computer*, pp. 344-353, 1990.
- Smith, J.E. [1989]. "Dynamic instruction scheduling and the astronautics ZS-I" *Computer* 22:7 (Jul.), pp. 21-35.
- Smith, J.E. et al. [1987]. "The ZS-I central processors," *Proc. Second Conf. on Architectural Support for Programming Languages and Operating Systems*, IEEE/ACM (Mar.), pp. 199-204.
- Sohi and Vajapeyam [1989]. "Tradeoffs in instruction format design for horizontal architectures," *Proc. Second Conf. on Architectural Support for Programming Languages and Operating Systems*, IEEE/ACM (Apr.), pp. 15-25.
- Sohi, G.S. [1990]. "Instruction issue logic for high-performance, interruptible, multiple functional unit pipelined computers," *IEEE Trans. on Computers* 39:3 (Mar.), 349-359.
- Steven et al., "An Evaluation of the iHARP Multiple-Instruction-Issue Processor", Division of Computer Science, Univ. of Hertfordshire, Hatfield, Hertfordshire, pp. 1-8, Sep. 1995.
- Stevens et al., "iHARP: A Multiple Instruction Issue Processor", *Technical Report No. 125*, Hatfield Polytechnic, Nov. 1991.
- Stevens, "An introduction to the Hatfield Superscalar Scheduler", *Technical Report No. 316*, University of Hertfordshire, Hatfield, Herts UK, Spring 1998.
- Stone et al., "Computer Architecture in the 1990s", *IEEE Computer*, pp. 30-37, Sep. 1991.
- The SPARC Architecture Manual, Version 8, Prentice Hall, New Jersey, 1992.
- Tjaden et al., "Detection and Parallel Execution of Parallel Instructions", *IEEE Transactions On Computers*, C-19(10):889-895 Oct. 1970.
- Tomasulo, R.M. [1967]. "An efficient algorithm for exploiting multiple arithmetic units," *IBM J. Research and Development* 11:1 (Jan.), 25-33.
- Uht, "Extraction of Massive Instruction Level Parallelism", *Computer Architecture News*, 21(3):5-12, Jun. 1993.
- Wall, "Limits of Instruction Level Parallelism", *Proceedings of the Fourth International Conference on Architectural Support for Programming Languages and Operation Systems*, pp. 176-188, Apr. 1991.
- Warter et al., "Enhanced Modulo Scheduling for Loops With Conditional Branches", MICRO 25, *Proceedings of the 25th Annual International Symposium on Microarchitecture*, pp. 170-179 (1992).
- Warter et al., "The Benefit of Predicated Execution for Software Pipelining", *HICSS-26 Conference Proceedings*, vol. 1, pp. 497-506, Jan. 1993.
- Weaver et al. (Editors), "The SPARC Architecture Manual—Version 9", SPARC International Inc., PTR Prentice Hall, Englewood Cliffs, New Jersey, 1994.
- Weiss and Smith [1984]. "Instruction issue logic for pipelined supercomputers," *Proc. 11th Symposium on Computer Architecture* (Jun.) pp. 110-118.
- Horst et al., "Multiple Instruction issue in the Nonstop cyclone processor" *Proceedings of the 17th Annual International Symposium on Computer Architecture*, IEEE Computer Society Press, Washington, (May 28-31, 1990) pp. 216-226.
- Requa et al., "The Piecewise data flow architecture: Architectural concepts" *IEEE Transaction on Computers* (1983) C-32(5):425-438.
- Wang et al., "I-NET mechanism for issuing multiple instructions" *Proceedings of Supercomputing IEEE Computer Society* (Nov. 14-18, 1988) Orlando, FL., pp. 88-95.

* cited by examiner

U.S. Patent

Mar. 19, 2002

Sheet 1 of 5

US 6,360,313 B1

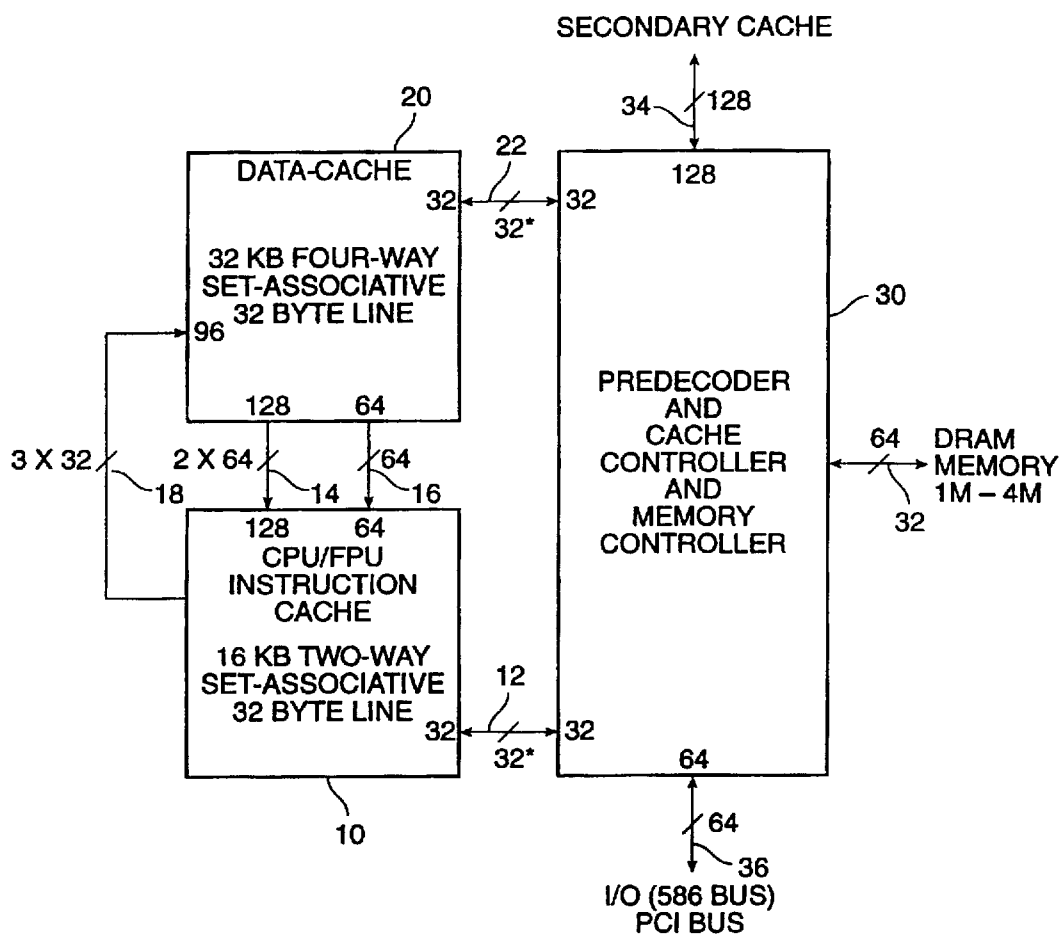


FIG. 1

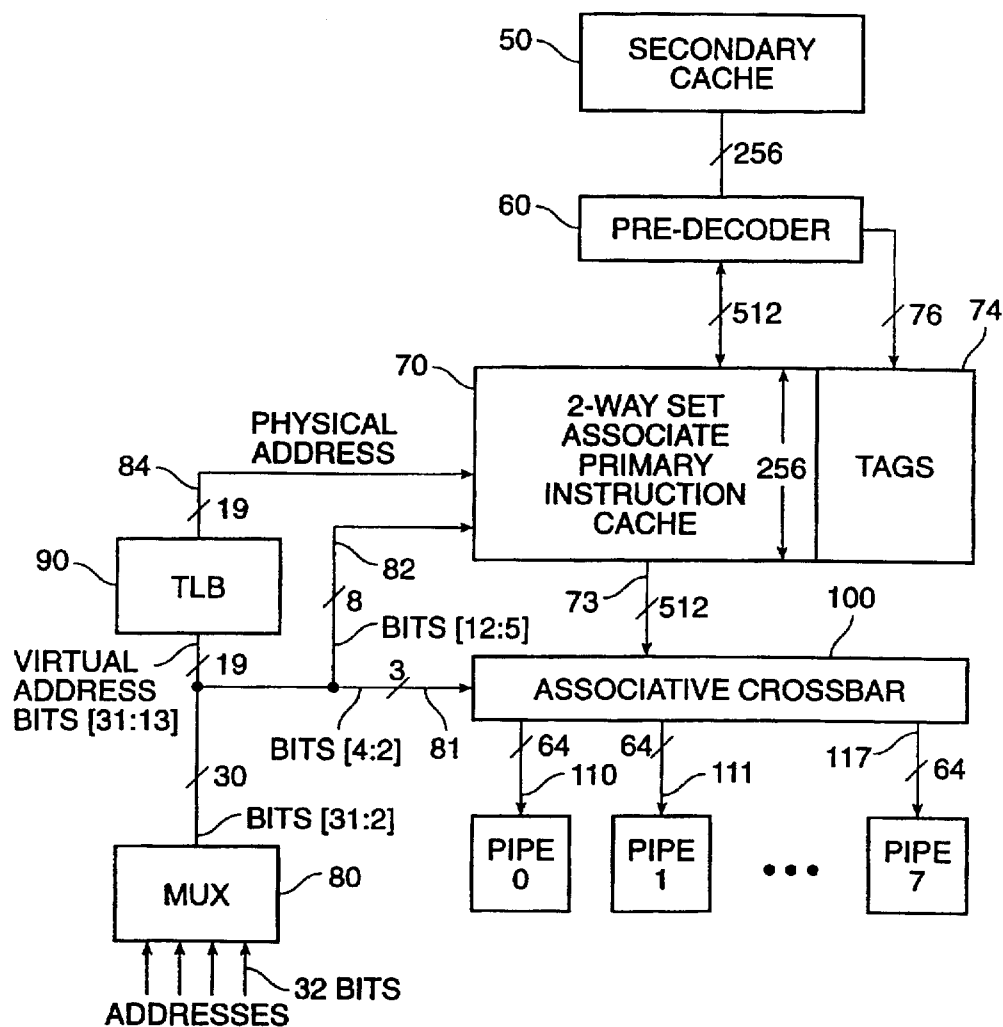


FIG. 2

U.S. Patent

Mar. 19, 2002

Sheet 3 of 5

US 6,360,313 B1

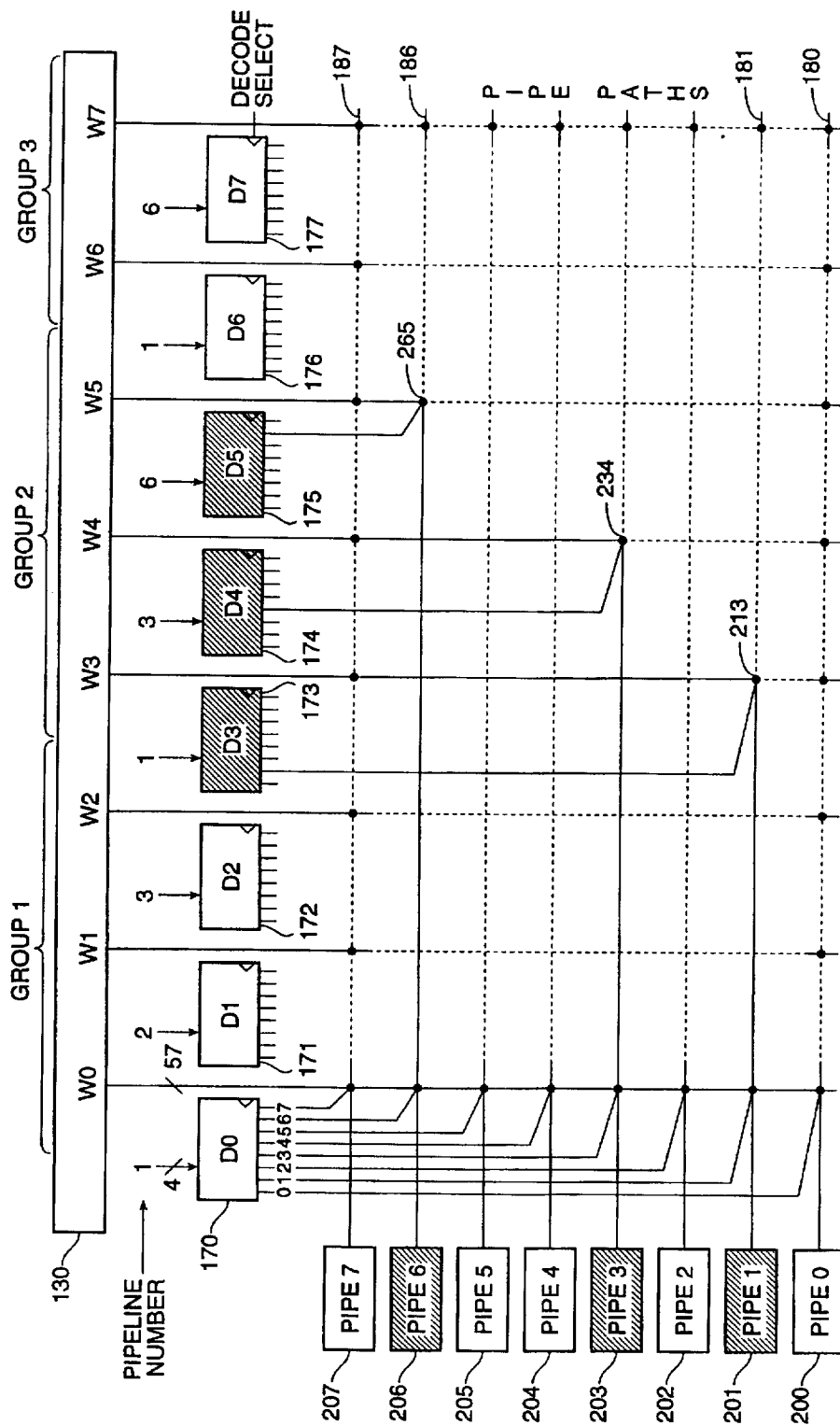


FIG. 3

U.S. Patent

Mar. 19, 2002

Sheet 4 of 5

US 6,360,313 B1

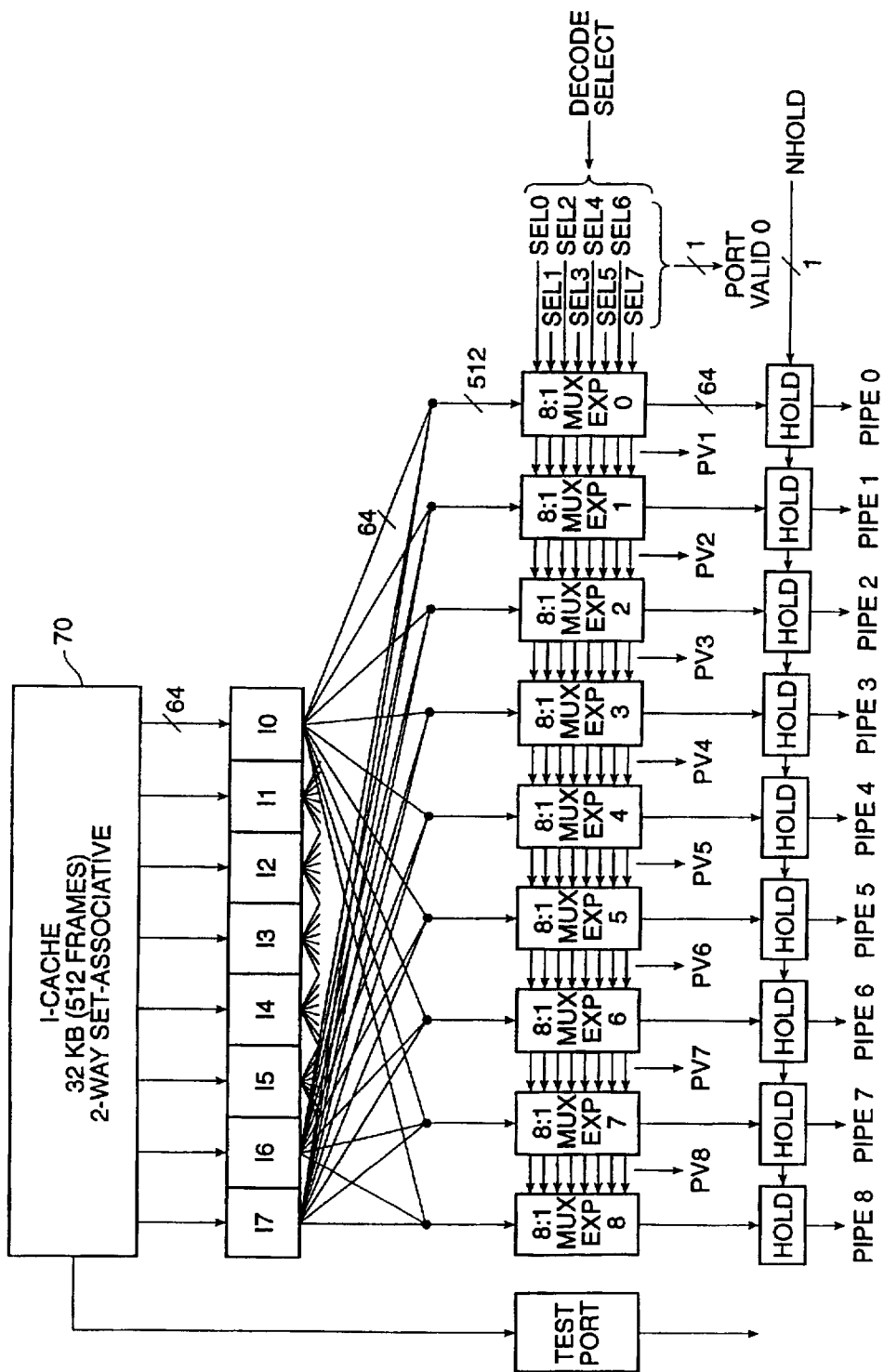


FIG. 4

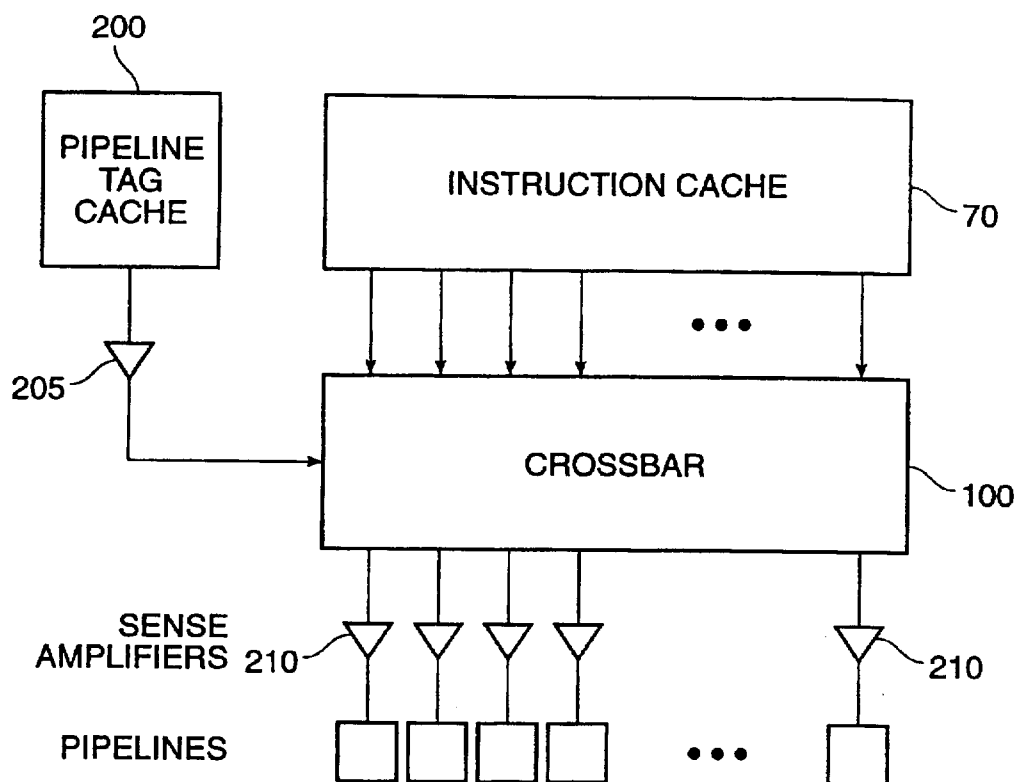


FIG. 5

US 6,360,313 B1

1

INSTRUCTION CACHE ASSOCIATIVE
CROSSBAR SWITCHCROSS-REFERENCES TO RELATED
APPLICATIONS

This application is a continuation of application Ser. No. 09/057,861 filed Apr. 9, 1998, now pending; which is a continuation of application Ser. No. 08/754,337 filed Nov. 22, 1996, which issued as U.S. Pat. No. 5,794,003 on Aug. 11, 1998; which is a continuation of application Ser. No. 08/498,135 filed Jul. 5, 1995, now abandoned; which is a continuation of application Ser. No. 08/147,797 filed Nov. 5, 1993, now abandoned; the disclosures of which are incorporated by reference for all purposes.

This application also incorporates by reference for all purposes application Ser. No. 08/422,753 filed Apr. 13, 1995, which issued as U.S. Pat. No. 5,560,028 on Sep. 24, 1996; which is a continuation of application Ser. No. 08/147,800 filed Nov. 5, 1993, now abandoned.

BACKGROUND OF THE INVENTION

This invention relates to the architecture of computing systems, and in particular to an architecture in which individual instructions may be executed in parallel, as well as to methods and apparatus for accomplishing that.

A common goal in the design of computer architectures is to increase the speed of execution of a given set of instructions. One approach to increasing instruction execution rates is to issue more than one instruction per clock cycle, in other words, to issue instructions in parallel. This allows the instruction execution rate to exceed the clock rate. Computing systems that issue multiple independent instructions during each clock cycle must solve the problem of routing the individual instructions that are dispatched in parallel to their respective execution units. One mechanism used to achieve this parallel routing of instructions is generally called a "crossbar switch."

In present state of the art computers, e.g. the Digital Equipment Alpha, the Sun Microsystems SuperSparc, and the Intel Pentium, the crossbar switch is implemented as part of the instruction pipeline. In these machines the crossbar is placed between the instruction decode and instruction execute stages. This is because the conventional approach requires the instructions to be decoded before it is possible to determine the pipeline to which they should be dispatched. Unfortunately, decoding in this manner slows system speed and requires extra surface area on the integrated circuit upon which the processor is formed. These disadvantages are explained further below.

SUMMARY OF THE INVENTION

We have developed a computing system architecture that enables instructions to be routed to an appropriate pipeline more quickly, at lower power, and with simpler circuitry than previously possible. This invention places the crossbar switch earlier in the pipeline, making it a part of the initial instruction fetch operation. This allows the crossbar to be a part of the cache itself, rather than a stage in the instruction pipeline. It also allows the crossbar to take advantage of circuit design parameters that are typical of regular memory structures rather than random logic. Such advantages include: lower switching voltages (200–300 millivolts rather than 3–5 volts); more compact design, and higher switching speeds. In addition, if the crossbar is placed in the cache, the need for many sense amplifiers is eliminated, reducing the circuitry required in the system as a whole.

2

To implement the crossbar switch, the instructions coming from the cache, or otherwise arriving at the switch, must be tagged or otherwise associated with a pipeline identifier to direct the instructions to the appropriate pipeline for execution. In other words, pipeline dispatch information must be available at the crossbar switch at instruction fetch time, before conventional instruction decode has occurred. There are several ways this capability can be satisfied: In one embodiment this system includes a mechanism that routes each instruction in a set of instructions to be executed in parallel to an appropriate pipeline, as determined by a pipeline tag applied to each instruction during compilation, or placed in a separate identifying instruction that accompanies the original instruction. Alternately the pipeline affiliation can be determined after compilation at the time that instructions are fetched from memory into the cache, using a special predecoder unit.

Thus, in one implementation, this system includes a register or other means, for example, the memory cells providing for storage of a line in the cache, for holding instructions to be executed in parallel. Each instruction has associated with it a pipeline identifier indicative of the pipeline to which that instruction is to be issued. A crossbar switch is provided which has a first set of connectors coupled to receive the instructions, and a second set of connectors coupled to the processing pipelines to which the instructions are to be dispatched for execution. Means are provided which are responsive to the pipeline identifiers of the individual instructions in the group supplied to the first set of connectors for routing those individual instructions onto appropriate paths of the second set of connectors, thereby supplying each instruction in the group to be executed in parallel to the appropriate pipeline.

In a preferred embodiment of this invention the associative crossbar is implemented in the instruction cache. By placing the crossbar in the cache all switching is done at low signal levels (approximately 200–300 millivolts). Switching at these low levels is substantially faster than switching at higher levels (5 volts) after the sense amplifiers. The lower power also eliminates the need for large driver circuits, and eliminates numerous sense amplifiers. Additionally by implementing the crossbar in the cache, the layout pitch of the crossbar lines matches the pitch of the layout of the cache.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is a block diagram illustrating a typical environment for a preferred implementation of this invention;

FIG. 2 is a diagram illustrating the overall structure of the instruction cache of FIG. 1;

FIG. 3 is a diagram illustrating one embodiment of the associative crossbar;

FIG. 4 is a diagram illustrating another embodiment of the associative crossbar; and

FIG. 5 is a diagram illustrating another embodiment of the associative crossbar.

DESCRIPTION OF THE SPECIFIC
EMBODIMENTS

FIG. 1 is a block diagram of a computer system incorporating the associative crossbar switch according to the preferred embodiment of this invention. The following briefly describes the overall preferred system environment within which the crossbar is incorporated. For additional information about the system, see application Ser. No.

US 6,360,313 B1

3

08/422,753 filed Apr. 13, 1995, which issued as U.S. Pat. No. 5,560,028 on Sep. 24, 1996; which is a continuation of application Ser. No. 08/147,800 filed Nov. 5, 1993, now abandoned, and entitled "Software Scheduled Superscaler Computer Architecture," which is incorporated by reference herein. FIG. 1 illustrates the organization of the integrated circuit chips by which the computing system is formed. As depicted, the system includes a first integrated circuit 10 that includes a central processing unit, a floating point unit, and an instruction cache.

In the preferred embodiment the instruction cache is a 16 kilobyte two-way set-associative 32 byte line cache. A set associative cache is one in which the lines (or blocks) can be placed only in a restricted set of locations. The line is first mapped into a set, but can be placed anywhere within that set. In a two-way set associative cache, two sets, or compartments, are provided, and each line can be placed in one compartment or the other.

The system also includes a data cache chip 20 that comprises a 32 kilobyte four-way set-associative 32 byte line cache. The third chip 30 of the system includes a predecoder, a cache controller, and a memory controller. The predecoder and instruction cache are explained further below. For the purposes of this invention, the CPU, FPU, data cache, cache controller and memory controller all may be considered of conventional design.

The communication paths among the chips are illustrated by arrows in FIG. 1. As shown, the CPU/FPU and instruction cache chip communicates over a 32 bit wide bus 12 with the predecoder chip 30. The asterisk is used to indicate that these communications are multiplexed so that a 64 bit word is communicated in two cycles. Chip 10 also receives information over 64 bit wide buses 14, 16 from the data cache 20, and supplies information to the data cache 20 over three 32 bit wide buses 18. The predecoder decodes a 32 bit instruction received from the secondary cache into a 64 bit word, and supplies that 64 bit word to the instruction cache on chip 10.

The cache controller on chip 30 is activated whenever a first level cache miss occurs. Then the cache controller either goes to main memory or to the secondary cache to fetch the needed information. In the preferred embodiment the secondary cache lines are 32 bytes and the cache has an 8 kilobyte page size.

The data cache chip 20 communicates with the cache controller chip 30 over another 32 bit wide bus. In addition, the cache controller chip 30 communicates over a 64 bit wide bus 32 with the DRAM memory, over a 128 bit wide bus 34 with a secondary cache, and over a 64 bit wide bus 36 to input/output devices.

As will be described further below, the system shown in FIG. 1 includes multiple pipelines able to operate in parallel on separate instructions which are dispatched to these parallel pipelines simultaneously. In one embodiment the parallel instructions have been identified by the compiler and tagged with a pipeline identification tag indicative of the specific pipeline to which that instruction should be dispatched.

In this system, an arbitrary number of instructions can be executed in parallel. In one embodiment of this system the central processing unit includes eight functional units and is capable of executing eight instructions in parallel. These pipelines are designated using the digits 0 to 7. Also, for this explanation each instruction word is assumed to be 32 bits (4 bytes) long.

As briefly mentioned above, in the preferred embodiment the pipeline identifiers are associated with individual

4

instructions in a set of instructions during compilation. In the preferred embodiment, this is achieved by compiling the instructions to be executed using a well-known compiler technology. During the compilation, the instructions are checked for data dependencies, dependence upon previous branch instructions, or other conditions that preclude their execution in parallel with other instructions. The result of the compilation is identification of a set or group of instructions which can be executed in parallel. In addition, in the preferred embodiment, the compiler determines the appropriate pipeline for execution of an individual instruction. This determination is essentially a determination of the type of instruction provided. For example, load instructions will be sent to the load pipeline, store instructions to the store pipeline, etc. The association of the instruction with the given pipeline can be achieved either by the compiler, or by later examination of the instruction itself, for example, during predecoding.

Referring again to FIG. 1, in normal operation the CPU will execute instructions from the instruction cache according to well-known principles. On an instruction cache miss, however, a set of instructions containing the instruction missed is transferred from the main memory into the secondary cache and then into the primary instruction cache, or from the secondary cache to the primary instruction cache, where it occupies one line of the instruction cache memory. Because instructions are only executed out of the instruction cache, all instructions ultimately undergo the following procedure.

At the time a group of instructions is transferred into the instruction cache, the instruction words are predecoded by the predecoder 30. As part of the predecoding process, a multiple bit field prefix is added to each instruction based upon a tag added to the instruction by the compiler. This prefix gives the explicit pipe number of the pipeline to which that instruction will be routed. Thus, at the time an instruction is supplied from the predecoder to the instruction cache, each instruction will have a pipeline identifier.

It may be desirable to implement the system of this invention on computer systems that already are in existence and therefore have instruction structures that have already been defined without available blank fields for the pipeline information. In this case, in another embodiment of this invention, the pipeline identifier information is supplied on a different clock cycle, then combined with the instructions in the cache or placed in a separate smaller cache. Such an approach can be achieved by adding a "no-op" instruction with fields that identify the pipeline for execution of the instruction, or by supplying the information relating to the parallel instructions in another manner. It therefore should be appreciated that the manner in which the instruction and pipeline identifier arrives at the crossbar to be processed is somewhat arbitrary. I use the word "associated" herein to designate the concept that the pipeline identifiers are not required to have a fixed relationship to the instruction words. That is, the pipeline identifiers need not be embedded within the instructions themselves by the compiler. Instead they may arrive from another means, or on a different cycle.

FIG. 2 is a simplified diagram illustrating the secondary cache, the predecoder, and the instruction cache. This figure, as well as FIGS. 3, 4 and 5, are used to explain the manner in which the instructions tagged with the pipeline identifier are routed to their designated instruction pipelines.

In FIG. 2, for illustration, assume that groups of instructions to be executed in parallel are fetched in a single transfer across a 256 bit (32 byte) wide path from a sec-

US 6,360,313 B1

5

ondary cache 50 into the predecoder 60. As explained above, the predecoder prefixes the pipeline "P" field to the instruction. After predecoding the resulting set of instructions is transferred into the primary instruction cache 70. At the same time, a tag is placed into the tag field 74 for that line.

In the preferred embodiment the instruction cache operates as a conventional physically-addressed instruction cache. In the example depicted in FIG. 2, the instruction cache will contain 512 bit sets of instructions of eight instructions each, organized in two compartments of 256 lines.

Address sources for the instruction cache arrive at a multiplexer 80 that selects the next address to be fetched. Because preferably instructions are always machine words, the low order two address bits <1:0> of the 32 bit address field supplied to multiplexer 80 are discarded. These two bits designate byte and half-word boundaries. Of the remaining 30 bits, the next three low order address bits <4:2>, which designate a particular instruction word in the set, are sent directly via bus 81 to the associative crossbar. The next low eight address bits <12:5> are supplied over bus 82 to the instruction cache 70 where they are used to select one of the 256 lines in the instruction cache. Finally, the remaining 19 bits of the virtual address <31:13> are sent to the translation lookaside buffer (TLB) 90. The TLB translates these bits into the high 19 bits of the physical address. The TLB then supplies them over bus 84 to the instruction cache. In the cache they are compared with the tag of the selected line, to determine if there is a "hit" or a "miss" in the instruction cache.

If there is a hit in the instruction cache, indicating that the addressed instruction is present in the cache, then the selected set of instructions is transferred across the 512 bit wide bus 73 into the associative crossbar 100. The associative crossbar 100 then dispatches the addressed instructions to the appropriate pipelines over buses 110, 111, . . . , 117. Preferably the bit lines from the memory cells storing the bits of the instruction are themselves coupled to the associative crossbar. This eliminates the need for numerous sense amplifiers, and allows the crossbar to operate on the lower voltage swing information from the cache line directly, without the normally intervening driver circuitry to slow system operation.

FIG. 3 illustrates in more detail one embodiment of the associative crossbar. A 512 bit wide register 130, which represents the memory cells in a line of the cache (or can be a physically separate register), contains at least the set of instructions capable of being issued. For the purposes of illustration, register 130 is shown as containing up to eight instruction words W0 to W7. Using means described in the copending application referred to above, the instructions have been sorted into groups for parallel execution. For illustration here, assume the instructions in Group 1 are to be dispatched to pipelines 1, 2 and 3; the instructions in Group 2 to pipelines 1, 3 and 6; and the instructions in Group 3 to pipelines 1 and 6. The decoder select signal enables only the appropriate set of instructions to be executed in parallel, essentially allowing register 130 to contain more than just one set of instructions. Of course, by only using register 130 only for one set of parallel instructions at a time, the decoder select signal is not needed.

As shown in FIG. 3, the crossbar switch itself consists of two sets of crossing pathways. In the horizontal direction are the pipeline pathways 180, 181, . . . , 187. In the vertical direction are the instruction word paths, 190, 191, . . . , 197. Each of these pipeline and instruction pathways is them-

6

selves a bus for transferring the instruction word. Each horizontal pipeline pathway is coupled to a pipeline execution unit 200, 201, 202, 207. Each of the vertical instruction word pathways 190, 191, . . . , 197 is coupled to an appropriate portion of register or cache line 130.

The decoders 170, 171, . . . , 177 associated with each instruction word pathway receive the 4 bit pipeline code from the instruction. Each decoder, for example decoder 170, provides eight 1 bit control lines as output. One of these control lines is associated with each pipeline pathway crossing of that instruction word pathway. Selection of a decoder as described with reference to FIG. 3 activates the output bit control line corresponding to that input pipe number. This signals the crossbar to close the switch between the word path associated with that decoder and the pipe path selected by that bit line. Establishing the cross connection between these two pathways causes a selected instruction word to flow into the selected pipeline. For example, decoder 173 has received the pipeline bits for word W3. Word W3 has associated with it pipeline path 1. The pipeline path 1 bits are decoded to activate switch 213 to supply instruction word W3 to pipeline execution unit 201 over pipeline path 181. In a similar manner, the identification of pipeline path 3 for decoder D4 activates switch 234 to supply instruction word W4 to pipeline path 3. Finally, the identification of pipeline 6 for word W5 in decoder D5 activates switch 265 to transfer instruction word W5 to pipeline execution unit 206 over pipeline pathway 186. Thus, instructions W3, W4 and W5 are executed by pipes 201, 203 and 206, respectively.

The pipeline processing units 200, 201, . . . , 207 shown in FIG. 3 can carry out desired operations. In a preferred embodiment of the invention, each of the eight pipelines first includes a sense amplifier to detect the state of the signals on the bit lines from the crossbar. In one embodiment the pipelines include first and second arithmetic logic units; first and second floating point units; first and second load units; a store unit and a control unit. The particular pipeline to which a given instruction word is dispatched will depend upon hardware constraints as well as data dependencies.

FIG. 4 is a diagram illustrating another embodiment of the associative crossbar. In FIG. 4 nine pipelines 0-8 are shown coupled to the crossbar. The decode select is used to enable a subset of the instructions in the register 130 for execution just as in the system of FIG. 3.

The execution ports that connect to the pipelines specified by the pipeline identification bits of the enabled instructions are then selected to multiplex out the appropriate instructions from the contents of the register. If one or more of the pipelines is not ready to receive a new instruction, a set of hold latches at the output of the execution ports prevents any of the enabled instructions from issuing until the "busy" pipeline is free. Otherwise the instructions pass transparently through the hold latches into their respective pipelines. Accompanying the output of each port is a "port valid" signal that indicates whether the port has valid information to issue to the hold latch. FIG. 5 illustrates an alternate embodiment for the invention where pipeline tags are not included with the instruction, but are supplied separately, or where the cache line itself is used as the register for the crossbar. In these situations, the pipeline tags may be placed into a high speed separate cache memory 200. The output from this memory can then control the crossbar in the same manner as described in conjunction with FIG. 3. This approach eliminates the need for sense amplifiers between the instruction cache and the crossbar. This enables the crossbar to switch very low voltage signals more quickly than higher level signals, and the need for hundreds of sense

US 6,360,313 B1

7

amplifiers is eliminated. To provide a higher level signal for control of the crossbar, sense amplifier 205 is placed between the pipeline tag cache 200 and the crossbar 100. Because the pipeline tag cache is a relatively small memory, however, it can operate more quickly than the instruction cache memory, and the tags therefore are available in time to control the crossbar despite the sense amplifier between the cache 200 and the crossbar 100. Once the switching occurs in the crossbar, then the signals are amplified by sense amplifiers 210 before being supplied to the various pipelines for execution.

The architecture described above provides many unique advantages to a system using this crossbar. The crossbar described is extremely flexible, enabling instructions to be executed sequentially or in parallel, depending entirely upon the "intelligence" of the compiler. Importantly, the associative crossbar relies upon the content of the message being decoded, not upon an external control circuit acting independently of the instructions being executed. In essence, the associative crossbar is self directed.

Another important advantage of this system is that it allows for more intelligent compilers. Two instructions which appear to a hardware decoder (such as in the prior art described above) to be dependent upon each other can be determined by the compiler not to be interdependent. For example, a hardware decoder would not permit two instructions $R1+R2=R3$ and $R3+R5=R6$ to be executed in parallel. A compiler, however, can be "intelligent" enough to determine that the second $R3$ is a previous value of $R3$, not the one calculated by $R1+R2$, and therefore allow both instructions to issue at the same time. This allows the software to be more flexible and faster.

Although the foregoing has been a description of the preferred embodiment of the invention, it will be apparent to those of skill in the art the numerous modifications and variations may be made to the invention without departing from the scope as described herein. For example, arbitrary numbers of pipelines, arbitrary numbers of decoders, and different architectures may be employed, yet rely upon the system we have developed.

What is claimed is:

1. In a computing system having a plurality of processing pipelines in which groups of individual instructions, within a very long instruction word, are executable by the plurality of processing pipelines, individual instructions in the very long instruction word to be executed having associated therewith group identifiers and pipeline identifiers, an apparatus for routing each individual instruction of a group of individual instructions to be executed in parallel to an appropriate processing pipeline of the plurality of processing pipelines, the apparatus comprising:

- a main memory for storing the very long instruction word;
- a very long instruction word storage coupled to the main memory, for receiving the very long instruction word from the main memory and for holding the very long instruction word the very long instruction word including groups of instructions to be executed in parallel, including pipeline identifiers and group identifiers;
- a selection circuit coupled to the very long instruction word storage for receiving the group identifiers included in the very long instruction word, for determining in response thereto a group of individual instructions to be executed in parallel, and for outputting a control signal;
- a decoder circuit coupled to the selection circuit and to the very long instruction word storage, for receiving the

8

control signal and the pipeline identifiers included in the very long instruction word, for determining in response thereto the appropriate processing pipeline for each individual instruction of the group, and for outputting switch control signals;

a switching circuit coupled to the decoder circuit, having a first set of connectors coupled to the very long instruction word storage for receiving the very long instruction word therefrom and a second set of connectors coupled to the plurality of processing pipelines, for coupling each individual instruction of the group to an appropriate processing pipeline in response to the switch control signals.

2. The apparatus of claim 1,

wherein the first set of connectors comprises a set of first communication buses, one first communication bus for each individual instruction held in the very long instruction word storage;

wherein the second set of connectors comprises a set of second communication buses, one second communication bus for each processing pipeline;

wherein the decoder circuit comprises a set of decoders coupled to receive as first input signals the pipeline identifiers and as second input signals the pipeline identifiers; and

wherein the switching circuit comprises a set of switches, one switch for every intersection between each of the first set of connectors and each of the second set of connectors, each switch for providing connections, in response to receiving the switch control signals, between each individual instruction in the group to be executed in parallel to the appropriate processing pipeline.

3. The apparatus of claim 2

wherein the selection means comprises a multiplexer coupled to receive the group identifiers for each individual instruction in the very long instruction word storage, and in response to the group identifiers, enable the decoder means to output switch control signals for each individual instructions of the group.

4. The apparatus of claim 3,

wherein the multiplexer supplies a switch control signal to the decoder means to enable the decoder means to output switch control signals for each individual instruction of the group of individual instructions from the very long instruction word.

5. In a computing system having a plurality of processing pipelines in which groups of individual instructions are executable, each individual instruction in a group executable in parallel by the plurality of processing pipelines, a method for transferring each individual instruction in a group to be executed through a switching unit having a first set of connectors coupled to a very long instruction word storage for receiving individual instructions therefrom, a second set of connectors coupled to the plurality of processing pipelines, and switches between the first set and the second set of connectors, the method comprising:

retrieving the very long instruction word from a main memory;

storing in the very long instruction word storage, the very long instruction word; the very long instruction word having a set of individual instructions including at least one group of individual instructions to be executed in parallel, individual instructions in the at least one group having associated therewith pipeline identifiers indicative of the processing pipeline which will execute that

US 6,360,313 B1

9

individual instruction, the very long instruction word storage also including at least one other individual instruction not in the at least one group of individual instructions, the at least one other individual instruction also having associated therewith the pipeline identifiers; and

using the pipeline identifiers to control the switches between the first set of connectors and the second set of connectors to thereby supply each individual instruction in the at least one group to be executed in parallel to an appropriate processing pipeline at a time different from when the at least one other individual instruction is supplied to an appropriate processing pipeline.

6. A method as in claim 5 wherein the step of using the pipeline identifiers comprises:

supplying the pipeline identifiers to individual decoders of a set of decoders, each decoder of which provides an output signal; and

using the output signals of the sets of decoders to control the switches between the first set of connectors and the second set of connectors to thereby supply each individual instruction in the at least one group to be executed in parallel to an appropriate processing pipeline.

7. A method as in claim 6 wherein individual instructions in the storage further includes group identifiers associated therewith to designate among the instructions present in the very long instruction word storage, which of the individual instructions may be simultaneously supplied to the plurality of processing pipelines, and the method further comprises:

supplying a group of instructions to be executed by the processing pipelines together with the group identifiers to a selector;

using the group identifiers to provide output determination signals; and

using both the output determination signals and the output signals to control the switches between the first set of connectors and the second set of connectors to thereby supply each instruction in the at least one group to be executed in parallel to the appropriate processing pipeline.

8. In a computing system having a plurality of processing pipelines in which groups of individual instructions are executable by the plurality of processing pipelines, a method for supplying each individual instruction in a group of individual instructions to be executed in parallel to an appropriate processing pipeline, the method comprising:

retrieving a very long instruction word from a main memory;

storing in a very long instruction word storage the very long instruction word retrieved from the main memory, the very long instruction word including groups of individual instructions to be executed in parallel, the individual instructions having associated therewith pipeline identifiers indicative of processing pipelines which will execute the individual instructions and having associated therewith group identifiers indicative of a group identification;

comparing the group identifiers to an execution group identifier of those instructions to be next executed in parallel; and

using the pipeline identifiers to control switches in a switch having a first set of connectors coupled to the very long instruction word storage for receiving the very long instruction word therefrom and a second set

10

of connectors coupled to the plurality of processing pipelines to thereby supply each individual instruction in the at least one group to be executed in parallel to the appropriate processing pipeline.

9. A computing system comprising:

a compiler for forming groups of instructions having opcodes including a first group of instructions and a second group of instructions, instructions in the first group of instructions executable in parallel, and instructions in the second group of instructions executable in parallel;

a first memory storage having at least a memory location, the memory location for storing the first group of instructions, for storing the second group of instructions comprising at least one instruction, and for storing group identifiers that indicate which instructions are included within the first group of instructions and which instructions are included within the second group of instructions;

a pre-decoder coupled to the first memory storage for decoding opcodes of instructions in the first group of instructions and opcodes of instructions in the second group of instructions, for forming a first group of expanded instructions, a second group of expanded instructions, and expanded group identifiers, and for determining processing pipeline identifiers associated with expanded instructions in the first group of expanded instructions and processing pipeline identifiers associated with expanded instructions in the second group of expanded instructions, in response thereto;

a second memory storage coupled to the predecoder having at least a memory location, the memory location for storing the first group of expanded instructions, the second group of expanded instructions, the expanded group identifiers, the processing pipeline identifiers associated with the expanded instructions in the first group of expanded instructions, and the processing pipeline identifiers associated with the expanded instructions in the second group of expanded instructions;

a decoder coupled to the second memory storage for receiving the first group of expanded instructions, the second group of expanded instructions, and the expanded group identifiers, and for issuing the first group of expanded instructions in response to the expanded group identifiers;

a plurality of processing pipelines;

a crossbar coupled to the decoder and to the plurality of processing pipelines, for issuing expanded instructions in the first group of expanded instructions to processing pipelines of the plurality of processing pipelines in response to the processing pipeline identifiers associated with the expanded instructions in the first group of expanded instructions.

10. The computing system of claim 9 wherein a first processing pipeline identifier from the processing pipeline identifiers associated with the expanded instructions in the first group of expanded instructions identifies a first floating point unit pipeline.

11. The computing system of claim 10 wherein a second processing pipeline identifier from the processing pipeline identifiers associated with the expanded instructions in the first group of expanded instructions identifies a first arithmetic logic unit pipeline.

12. The computing system of claim 10 wherein a second processing pipeline identifier from the processing pipeline

US 6,360,313 B1

11

identifiers associated with the expanded instructions in the first group of expanded instructions identifies a second floating point unit pipeline.

13. The computing system of claim 11 wherein a third processing pipeline identifier from the processing pipeline identifiers associated with the expanded instructions in the first group of expanded instructions identifies a second floating point unit pipeline.

14. The computing system of claim 9 wherein a first processing pipeline identifier from the processing pipeline identifiers associated with the expanded instructions in the second group of expanded instructions identifies a store unit pipeline.

15. The computing system of claim 14 wherein a second processing pipeline identifier from the processing pipeline identifiers associated with the expanded instructions in the second group of expanded instructions identifies a control unit pipeline.

16. The computing system of claim 9

wherein the decoder is also for issuing the second group of expanded instructions in response to the expanded group identifiers; and

wherein the crossbar is also for issuing expanded instructions in the second group of expanded instructions to processing pipelines of the plurality of processing pipelines in response to the processing pipeline identifiers associated with the expanded instructions in the second group of expanded instructions.

17. The computing system of claim 9 wherein the group identifiers are associated with instructions in the first group of instructions and with instructions in the second group of instruction.

18. The computing system of claim 17 wherein the group identifiers are embedded with instructions in the first group of instructions and with instructions in the second group of instruction.

19. The computing system of claim 9 wherein the first group of expanded instructions comprises at least one expanded instruction and the second group of expanded instructions comprises at least two expanded instructions.

20. The computing system of claim 19 wherein the second memory storage includes at least the two expanded instructions of the second group of expanded instructions and the one expanded instruction of the first group of expanded instructions.

21. The computing system of claim 9

wherein the compiler is also for forming a third group of instructions, the instructions in the third group of instructions executable in parallel;

wherein the memory location of the first memory storage is also for storing a third group of instructions in parallel with the first group of instructions and the second group of instructions; and

wherein the group identifiers indicate which instructions are included within the third group of instructions.

22. The computing system of claim 21 wherein the third group of instructions comprises at least one instruction.

23. The computing system of claim 9 wherein the first memory storage is a superscaler cache.

24. The computing system of claim 9 wherein one expanded instruction of the second group of expanded instructions is a branch instruction.

25. The computing system of claim 9 wherein the compiler explicitly identifies instructions that can be performed in parallel for the first group of instructions.

26. A method for issuing groups of individual software-scheduled instructions in parallel for processing comprises:

12

forming a first group of software-scheduled instructions, a second group of software-scheduled instructions comprising at least one instruction, and group identifiers indicating which software-scheduled instructions are included within the first group of software-scheduled instructions and which software-scheduled instructions are included within the second group of software-scheduled instructions, software-scheduled instructions in the first group of software-scheduled instructions having opcodes and executable in parallel, and software-scheduled instructions in the second group of software-scheduled instructions having opcodes and executable in parallel;

storing the first group of software-scheduled instructions, the second group of software-scheduled instructions, and the group identifiers in parallel in a first memory location;

forming a first group of expanded software-scheduled instructions, a second group of expanded software-scheduled instructions, and expanded group identifiers in response to opcodes of software-scheduled instructions in the first group of software-scheduled instructions and opcodes of software-scheduled instructions in the second group of software-scheduled instructions;

determining processing pipelines appropriate for expanded software-scheduled instructions in the first group of expanded software-scheduled instructions and processing pipelines appropriate for expanded software-scheduled instructions in the second group of expanded software-scheduled instructions also in response to the opcodes of software-scheduled instructions in the first group of software-scheduled instructions and the opcodes of software-scheduled instructions in the second group of software-scheduled instructions; and

issuing the first group of expanded software-scheduled instructions to the processing pipelines appropriate for expanded software-scheduled instructions in the first group of expanded software-scheduled instructions, in response to the expanded group identifiers.

27. The method of claim 26 further comprising:

issuing the second group of expanded software-scheduled instructions to the processing pipelines appropriate for expanded software-scheduled instructions in the second group of expanded software-scheduled instructions, in response to the expanded group identifiers.

28. The method of claim 26 wherein a first processing pipeline appropriate for an expanded software-scheduled instruction in the first group of expanded software-scheduled instructions is coupled to a first arithmetic logic processing unit.

29. The method of claim 28 wherein a second processing pipeline appropriate for an expanded software-scheduled instruction in the first group of expanded software-scheduled instructions is coupled to a first floating point processing unit.

30. The method of claim 28 wherein a second processing pipeline appropriate for an expanded software-scheduled instruction in the first group of expanded software-scheduled instructions is coupled to a second arithmetic logic processing unit.

31. The method of claim 29 wherein a third processing pipeline appropriate for an expanded software-scheduled instruction in the first group of expanded software-scheduled instructions is coupled to a second arithmetic logic processing unit.

US 6,360,313 B1

13

32. The method of claim 28 wherein a second processing pipeline appropriate for an expanded software-scheduled instruction in the first group of expanded software-scheduled instructions is coupled to a load unit.

33. The method of claim 26 wherein a first processing pipeline appropriate for an expanded software-scheduled instruction in the second group of expanded software-scheduled instructions is coupled to a store unit.

34. The method of claim 33 wherein a second processing pipeline appropriate for an expanded software-scheduled instruction in the second group of expanded software-scheduled instructions is coupled to a control unit.

35. The method of claim 26 further comprises:

storing the first group of expanded software-scheduled instructions, the second group of expanded software-scheduled instructions, and the expanded group identifiers, in a second memory location;

wherein the step of issuing the first group of expanded software-scheduled instructions comprises issuing the first group of expanded software-scheduled instructions and the second group of expanded software-scheduled instructions from the second memory location to a decoder; and

issuing the first group of expanded software-scheduled instructions to the processing pipelines appropriate for the expanded software-scheduled instructions in the first group of expanded software-scheduled instructions from the decoder in response to the expanded group identifiers stored in the second memory location.

36. The method of claim 35 further comprises:

issuing the second group of expanded software-scheduled instructions to the processing pipelines appropriate for the expanded software-scheduled instructions in the second group of expanded software-scheduled instructions from the decoder in response to the expanded group identifiers stored in the second memory location.

37. The method of claim 36 wherein the processing pipelines appropriate for the expanded software-scheduled instructions in the first group of expanded software-scheduled instructions are identified by processing pipeline identifiers.

38. The method of claim 37 wherein the step of storing the first group of expanded software-scheduled instructions further comprises storing the processing pipeline identifiers in the second memory location.

39. The method of claim 38 wherein the step of issuing the first group of expanded software-scheduled instructions to the processing pipelines comprises issuing the first group of expanded software-scheduled instructions to a crossbar.

40. The method of claim 39 wherein the crossbar is an associative crossbar responsive to the processing pipeline identifiers.

41. The method of claim 35 wherein the first group of expanded instructions comprises at least two expanded instructions.

42. The method of claim 41 wherein the second group of expanded instructions comprises at least one expanded instruction.

43. The method of claim 42 wherein the two expanded instructions of the first group of expanded instructions and the one expanded instruction of the second group of expanded instructions are stored in the second memory location.

44. The method of claim 26 wherein the step of forming the first group of software-scheduled instructions, the second group of software-scheduled instructions, and the group identifiers comprises using a compiler to form the first group

14

of software-scheduled instructions, the second group of software-scheduled instructions, and the group identifiers.

45. The method of claim 44 wherein the group identifiers are associated with instructions in the first group of instructions and with instructions in the second group of instructions.

46. The method of claim 44 wherein the compiler explicitly determines parallel executable instructions among a plurality of instructions to include in the first group of software-scheduled instructions.

47. The method of claim 26 wherein the step of forming the first group of software-scheduled instructions, the second group of software-scheduled instructions, and the group identifiers further comprises the step of forming a third group of software-scheduled instructions executable in parallel; and

wherein the group identifiers also indicate which software-scheduled instructions are included within the third group of software-scheduled instructions.

48. The method of claim 26 wherein the first memory location is a cache location in a superscaler cache.

49. A method for issuing a group of individual instructions in parallel for processing comprises:

storing in parallel a plurality of instructions and instruction grouping information in a location in a memory, the plurality of instructions and the instruction grouping information determined by a compiler, the instruction grouping information indicating which instructions of the plurality of instructions belong to a first group of instructions and can be issued in parallel, and indicating at least another instruction of the plurality of instructions that can be issued after the first group of instructions;

issuing the first group of instructions in response to the instruction grouping information; and

coupling instructions in the first group of instructions to instruction pipelines appropriate for the instructions in the first group of instructions.

50. The method of claim 49 further comprises

after coupling instructions in the first group of instructions, issuing the at least another instruction in response to the instruction grouping information; and coupling the at least another instruction to an instruction pipeline appropriate for the at least another instruction.

51. The method of claim 50 wherein the instruction grouping information also indicates which instructions of the plurality of instructions belong to a second group of instructions and can be issued in parallel after the at least another instruction.

52. The method of claim 49

wherein the instructions in the first group of instructions include instruction types; and

wherein coupling instructions in the first group of instructions further comprises determining the instruction pipelines appropriate for the instructions in the first group of instructions in response to the instruction types.

53. The method of claim 52 wherein the instruction types comprise opcodes.

54. The method of claim 50 wherein issuing the first group of instructions further comprises receiving the first group of instructions, the at least another instruction, and the instruction grouping information from the location in the memory.

55. The method of claim 50 wherein the first group of instructions comprises at least two instructions.

56. The method of claim 50 wherein the first group of instructions comprises at least one instruction.

US 6,360,313 B1

15

57. The method of claim 55 wherein an instruction frame comprises the plurality of instructions and instruction grouping information, and

wherein the instruction frame includes at least the two instructions of the first group of instructions and the at least another instruction.

58. The method of claim 50 wherein the compiler explicitly identifies parallel executable instructions from the plurality of instructions.

59. The method of claim 52 wherein the instruction types comprise pipeline identifiers that identify the instruction pipelines appropriate for the instructions in the first group of instructions.

60. The method of claim 59 wherein the pipeline identifiers are determined by the compiler.

61. The method of claim 60 wherein coupling instructions in the first group of instructions comprises using a crossbar switch to couple the instructions in the first group of instructions to the instruction pipelines appropriate for the instructions in the first group of instructions in response to the pipeline identifiers.

62. The method of claim 50 wherein the memory is a cache and the location is a cache entry.

63. A computing system comprising:

a cache including a plurality of cache entries, a cache entry of the plurality of cache entries configured to store in parallel a plurality of software-scheduled instructions and instruction grouping information, the instruction grouping information configured to identify a first group of software-scheduled instructions from the plurality of software-scheduled instructions and to identify at least another software-scheduled instruction from the plurality of software-scheduled instructions, the at least another software-scheduled instruction to be issued after instructions in the first group of software-scheduled instructions.

64. The computing system of claim 63 wherein the instruction grouping information is also configured to identify a second group of software-scheduled instructions from the plurality of software-scheduled instructions, instructions in the second group of software-scheduled instructions to be issued after at least another software-scheduled instruction.

65. The computing system of claim 63

wherein the cache is also configured to issue the first group of software-scheduled instructions, the at least another software-scheduled instruction, and the instruction grouping information;

the computing system further comprising a group decoder coupled to the cache and configured to receive the first group of software-scheduled instructions, the at least another software-scheduled instruction, and the instruction grouping information, and to issue the first group of software-scheduled instructions in response to the instruction grouping information.

66. The computing system of claim 65 wherein the group decoder is also configured to issue the at least another software-scheduled instruction, after the first group of software-scheduled instructions in response to the instruction grouping information.

67. The computing system of claim 65

wherein each instruction in the first group of software-scheduled instructions includes an instruction type,

the computing system further comprising an instruction decoder coupled to the cache and configured to receive the instruction types of the instructions in the first group of software-scheduled instructions and to deter-

16

mine instruction pipelines appropriate for each of the instructions in the first group of software-scheduled instructions.

68. The computing system of claim 67 wherein the instruction type comprises an opcode, and the instruction decoder comprises an opcode decoder.

69. The computing system of claim 67 further comprising: a pipeline coupler coupled to the group decoder and to the instruction decoder and configured to receive the first group of software-scheduled instructions and configured to couple each instruction in the first group of software-scheduled instructions to the instruction pipelines appropriate for the instructions in the first group of software-scheduled instructions.

70. The computing system of claim 69 wherein the pipeline coupler is a crossbar switch.

71. The computing system of claim 65 wherein the first group of software-scheduled instructions comprises at least two instructions.

72. The computing system of claim 71 wherein the first group of software-scheduled instructions comprises at least one instruction.

73. The computing system of claim 72 wherein an instruction frame comprises the plurality of software-scheduled instructions and instruction grouping information; and wherein the instruction frame includes at least the two instructions of the first group of software-scheduled instructions and the at least another software-scheduled instruction.

74. The computing system of claim 63 wherein the cache is a superscaler cache.

75. The computing system of claim 63 wherein a compiler explicitly identifies parallel executable instructions from the plurality of software-scheduled instructions that form the first group of software-scheduled instructions.

76. The computing system of claim 67 wherein the instruction types comprise pipeline identifiers indicative of instruction pipelines appropriate for the instructions in the first group of software-scheduled instructions.

77. The method of claim 49 wherein the instruction grouping information also indicates instruction pipelines appropriate for the instructions in the first group of instructions; and

wherein coupling instructions in the first group of instructions to the instruction pipelines appropriate for the instructions in the first group of instructions is in response to the instruction grouping information.

78. The method of claim 77 wherein the instruction grouping information also indicates instruction pipelines appropriate for the instructions in the second group of instructions.

79. The computing system of claim 65 further comprising an switching unit coupled to the cache and configured to receive the instructions in the first group of software-scheduled instructions and to couple the instructions in the first group of software-scheduled instructions to instruction pipelines appropriate for each of the instructions in the first group of software-scheduled instructions in response to the instruction grouping information.

80. The computing system of claim 79 wherein the switching unit is also configured to receive the at least another software-scheduled instruction and to couple the at least another software-scheduled instruction to an instruction pipeline appropriate the at least another software-scheduled instruction in response to the instruction grouping information.

81. A computing system in which instructions are issued in parallel to processing pipelines, the computing system comprising:

US 6,360,313 B1

17

a storage configured to store an instruction frame, the instruction frame including a plurality of instructions including a group of instructions and at least another instruction in parallel, instructions in the group of instructions to be issued in parallel, the at least another instruction to be issued at a time different from a time when the group of instructions is to be issued, the instruction frame also including data associated with the plurality of instructions, the data associated with the plurality of instructions indicative of which instructions in the plurality of instructions are included in the group of instructions, the data associated with the plurality of instructions also indicative of processing pipelines appropriate for the plurality of instructions, and the data associated with the plurality of instructions determined at a compile time; and

a switching circuit coupled to the storage, configured to issue the instructions in the group of instructions in parallel, to processing pipelines appropriate for the instructions in the group of instructions, in response to the data associated with the plurality of instructions.

82. The computing system of claim 81 wherein the group of instructions comprises at least two instructions.

83. The computing system of claim 81 wherein the group of instructions comprises one instruction.

84. The computing system of claim 81 wherein the switching circuit is also configured to issue the at least another instruction to a processing pipeline appropriate for the at least another instruction in response to the data associated with the plurality of instructions.

85. The computing system of claim 81 wherein the processing pipelines appropriate for the instructions in the group of instructions are respectively coupled to execution units appropriate for the instructions in the group of instructions.

86. The computing system of claim 85 wherein an execution unit appropriate for a first instruction in the group of instructions is a memory.

87. The computing system of claim 86 wherein an execution unit appropriate for a second instruction in the group of instructions is an arithmetic logic unit.

88. The computing system of claim 86 wherein an execution unit appropriate for a second instruction in the group of instructions is a floating point unit.

89. The computing system of claim 85 wherein an execution unit appropriate for one instruction in the group of instructions is a branch unit.

90. The computing system of claim 85 wherein a type of execution unit appropriate for a first instruction in the group of instructions and a type of execution unit appropriate for a second instruction in the one group of instructions are similar.

91. The computing system of claim 81 wherein the plurality of instructions in the instruction frame are determined at the compile time.

92. A method for issuing groups of instructions in parallel to processing pipelines, the method comprising:

storing in a storage, an instruction frame, the instruction frame including a plurality of instructions including a group of instructions and at least another instruction in parallel, instructions in the group of instructions to be issued in parallel, the at least another instruction to be issued at a time different from a time when the group of instructions is to be issued, the instruction frame also including data associated with the plurality of instructions, the data associated with the plurality of instructions indicative of which instructions are

18

included in the group of instructions, the data associated with the instructions also indicative of processing pipelines appropriate for the plurality of instructions, and the data associated with the plurality of instructions determined at compile time; and

issuing the instructions in the group of instructions in parallel to processing pipelines appropriate for the instructions in the group of instructions, in response to the data associated with the plurality of instructions.

93. The method of claim 92 further comprising:

during compile time, determining the plurality of instructions in the instruction frame.

94. The method of claim 92 wherein the group of instructions comprises at least two instructions.

95. The method of claim 94 further comprising, before issuing the group of instructions, issuing the at least another instruction to a processing pipeline appropriate for the at least another instruction in response to the data associated with the plurality of instructions.

96. The method of claim 92 wherein the processing pipelines appropriate for the instructions in the group of instructions are respectively coupled to execution units appropriate for the instructions in the group of instructions.

97. The method of claim 96 wherein a type of execution unit appropriate for a processing pipeline appropriate for a first instruction in the group of instructions is an arithmetic logic unit.

98. The method of claim 97 wherein a type of execution unit appropriate for a processing pipeline appropriate for a second instruction in the group of instructions is an arithmetic logic unit.

99. The method of claim 96 wherein a type of execution unit appropriate for a processing pipeline appropriate for a first instruction in the group of instructions is a floating point unit.

100. The method of claim 96 wherein a type of execution unit appropriate for a processing pipeline appropriate for a first instruction in the group of instructions is a memory unit and a type of execution unit appropriate for a processing pipeline appropriate for a second instruction in the group of instructions is a memory unit.

101. A method of operating a microprocessor comprises: compiling computer code to determine a frame of instructions;

storing in a memory storage the frame of instructions, the frame of instructions including a plurality of instructions and issue data, the plurality of instructions including at least a first instruction, a second instruction, and a third instruction, the issue data comprising data indicating that the first instruction is to be issued before the second instruction and the third instruction and that the second and third instructions are to be issued in parallel, and the issue data indicating respective processing units appropriate for the first instruction, the second instruction, and the third instruction; and

issuing the first instruction to a processing unit appropriate for the first instruction in response to the issue data; and

issuing the second instruction and the third instruction in parallel to respective processing units appropriate for the second instruction and the third instruction in response to the issue data.

102. The method of claim 101 wherein the processing unit appropriate for the first instruction is a memory unit.

103. The method of claim 102 wherein a processing unit appropriate for the second instruction is a memory unit.

US 6,360,313 B1

19

104. The method of claim 102 wherein a processing unit appropriate for the second instruction is an arithmetic logic unit.

105. The method of claim 101 wherein issuing the first instruction to a processing unit comprises using a switching unit to couple the first instruction to the processing unit appropriate for the first instruction in response to the issue data.

106. A computing system having a plurality of processing pipelines for executing groups of individual instructions, within very long instruction words, each individual instruction to be executed in each group being executed by different processing pipelines in parallel, the computing system comprising:

a main memory for storing a very long instruction word; a very long instruction word storage, coupled to the main memory, for receiving the very long instruction word from the main memory, and for holding the very long instruction word, the very long instruction word including a predetermined number N of individual instructions, and including at least one group of M individual instructions to be executed in parallel, where $M \leq N$, each individual instruction in the very long instruction word storage to be executed having an a pipeline identifier indicative of a processing pipeline for executing the individual instruction, and having a group identifier indicative of a group of individual instructions to which the individual instruction is assigned for execution in parallel;

instruction in a group decoder responsive to the group identifier for each individual instruction in the very long instruction word storage to be executed for enabling each individual instruction in the very long instruction word storage having a similar group identifier, to be executed in parallel by the plurality of processing pipelines; and

a pipeline decoder responsive to the pipeline identifier of each individual instructions in the very long instruction word storage to be executed for causing each individual instruction in a group of individual instructions having the similar group identifier to be supplied to the different processing pipelines.

107. The computing system of claim 106 wherein M is greater than or equal to 1.

108. The computing system of claim 106 wherein M is greater than 1.

109. The computing system in claim 106, wherein the very long instruction word storage includes the at least one group of M individual instructions, and also includes group identifiers and pipeline identifiers for each individual instruction in the at least one group of M individual instructions.

110. The computing system in claim 107, wherein each individual instruction in the at least one group of M individual instructions has associated therewith a different pipeline identifier.

111. The computing system of claim 106, wherein the very long instruction word storage holds a first group of individual instructions to be executed in parallel and a second group of individual instructions to be executed in parallel after the first group, each individual instruction in the first group having associated therewith a first group identifier different from a second group identifier associated with each individual instruction in the second group, the first group and the second group being placed adjacent to each other in the very long instruction word storage.

20

112. The computing system of claim 111 wherein:

the very long instruction word storage comprises a line in a cache memory having a fixed number of storage locations; and

the first group of individual instructions is placed at one end of the line in the cache memory, and the second group of individual instructions is placed next to the first group of individual instructions.

113. A method of executing in a plurality of processing pipelines arbitrary numbers of instructions in a stream of instructions in parallel which have been compiled to determine which instructions can be executed in parallel, the method comprising:

in response to the compilation, assigning a common group identifier to a group of instructions which can be executed in parallel;

determining a processing pipeline for execution of each instruction in the group of instructions to be executed;

assigning a pipeline identifier to each instruction in the group;

associating the common group identifier and the pipeline identifier with the group of instructions;

forming a very long instruction word with a fixed number of the instructions including at least the group of instructions and the common group identifier as well as at least one other instruction having a different group identifier, the at least one other instruction to be issued at a time different from instructions in the group of instructions; and

storing the very long instruction word in a main memory.

114. A method as in claim 113 further comprising:

placing the very long instruction word retrieved from the main memory into a very long instruction word register; and

executing the group of instructions in the plurality of processing pipelines in parallel.

115. A method as in claim 114,

wherein the very long instruction word register holds at least two groups of instructions; and

wherein placing the instructions in the very long instruction word register comprises placing the group of instructions adjacent to the at least one other instruction having the different group identifier in the very long instruction word register.

116. A method as in claim 115 wherein executing the group of instructions in parallel comprises:

coupling the very long instruction word register to a detection means to receive group identifiers associated with each instruction to be executed in the very long instruction word; and

supplying only instructions in the group of instructions to the processing pipelines in response to the group identifiers.

117. In a computing system having a plurality of processing pipelines in which groups of individual instructions, within very long instruction words, are executable in parallel by processing pipelines, a method for supplying each individual instruction in a group to be executed in parallel to corresponding appropriate processing pipelines, the method comprising:

retrieving a very long instruction word from a main memory;

storing in a very long instruction word storage the very long instruction word, the very long instruction word

US 6,360,313 B1

21

including groups of individual instructions to be executed in parallel, the groups of individual instruction to be executed in the very long instruction word having associated therewith pipeline identifiers indicative of the corresponding appropriate processing pipeline which will execute the instructions and group identifiers indicative of groups of instructions;

using the group identifiers in the very long instruction word to identify an execution group; and

using the pipeline identifiers to execute each individual instruction in the execution group in the corresponding appropriate processing pipelines.

118. In a computing system having a plurality of processing pipelines in which groups of individual instructions, from a very long instruction word, are executable in parallel by the plurality of processing pipelines, an apparatus for routing each individual instruction in a particular group to be executed in parallel to an appropriate processing pipeline, the apparatus comprising:

a main memory for storing the very long instruction word;

a very long instruction word storage coupled to the main memory, for receiving the very long instruction word from the main memory and for holding the very long instruction word, the very long instruction word including groups of individual instructions, individual instructions to be executed in the very long instruction word storage having associated therewith pipeline identifiers indicative of processing pipelines for executing the individual instructions and also having associated therewith group identifiers to designate groups of individual instructions to which individual instructions are assigned, the groups of individual instructions including at least a first group of individual instructions and at least another individual instruction, the pipeline identifiers and the group identifiers included in the very long instruction word;

a switching circuit having a first set of connectors coupled to the very long instruction word storage and a second set of connectors coupled to the plurality of processing pipelines; and

a router coupled to the very long instruction word storage and the switching circuit, responsive to the pipeline identifiers for routing each individual instruction in the first group of individual instructions from connectors of the first set of connectors onto appropriate connectors of the second set of connectors, to thereby supply each individual instruction in the first group of individual instructions to be executed in parallel to the appropriate processing pipeline at a time different from when the at least another individual instruction is supplied to an appropriate processing pipeline.

22

119. The apparatus of claim 118,

wherein the first set of connectors includes a set of first communication buses, one first communication bus for each individual instruction to be executed in the very long instruction word storage;

wherein the second set of connectors includes a set of second communication buses, one second communication bus for each processing pipeline; and

wherein the router comprises:

a set of decoders coupled to the very long instruction word storage, the decoders in the set for receiving as input signals the pipeline identifiers included in the very long instruction word storage and in response thereto for supplying as output signals switch control signals corresponding to each individual instruction in the very long instruction word storage; and

a set of switches coupled to the set of decoders and to the switching circuit, one switch of the set of switches at each intersection of each of the first set of communication buses with each of the second set of communication buses, each switch for receiving the switch control signals and for providing connections in response to receiving a corresponding switch control signal to thereby supply each individual instruction in the group to be executed in parallel to the appropriate processing pipeline.

120. The apparatus of claim 119 further comprising:

a detection circuit coupled to the very long instruction word storage, for receiving the group identifiers included in the very long instruction word storage to be executed and in response thereto supply a group control signal; and

wherein the set of decoders are also coupled to the detection circuit for receiving the group control signal and in response thereto supply the switch control signal for only those individual instructions in the group to be supplied to the plurality of processing pipelines.

121. The apparatus of claim 120,

wherein the detection circuit comprises a multiplexer coupled to receive the group identifiers included in the very long instruction word storage and in response thereto allow the group of individual instructions to be supplied to the plurality of processing pipelines.

122. Apparatus as in claim 121 wherein the multiplexer supplies output signals to the set of decoders to indicate the group of individual instructions to be next supplied to the plurality of processing pipelines.

* * * * *