Constructing database representing manifold array architecture instruction set for use in support tool code creation

Abstract

Details of a highly cost effective and efficient implementation of a manifold array (ManArray) architecture and instruction syntax for use therewith are described herein. Various aspects of this approach include the regularity of the syntax, the relative ease with which the instruction set can be represented in database form, the ready ability with which tools can be created, the ready generation of self-checking codes and parameterized testcases. Parameterizations can be fairly easily mapped and system maintenance is significantly simplified.

Inventors: **Pechanek; Gerald G.** (Cary, NC); **Strube; David Carl** (Raleigh, NC); **Barry; Edwin Frank** (Cary, NC); **Kurak, Jr.; Charles W.** (Durham, NC); **Busboom; Carl Donald** (Cary, NC); **Schneider; Dale Edward** (Durham, NC); **Pitsianis; Nikos P.** (Chapel Hill, NC); **Morris; Grayson** (Durham, NC); **Wolff; Edward A.** (Chapel Hill, NC); **Marchand; Patrick R.** (Apex, NC); **Rodriguez; Ricardo E.** (Raleigh, NC); **Jacobs; Marco C.** (Durham, NC)

Assignee: **PTS Corporation** (San Jose, CA)

Appl. No.: **599980**

Filed: **June 22, 2000**
RELATED APPLICATIONS

The present application claims the benefit of U.S. Provisional Application Ser. No. 60/140,425 entitled "Methods and Apparatus for Parallel Processing Utilizing a Manifold Array (ManArray) Architecture and Instruction Syntax" and filed Jun. 22, 1999 which is incorporated herein by reference in its entirety.

Claims

1. An array processor apparatus comprising:

- an array of processing elements executing a regular instruction set; and

- means for constructing a database for the instruction set, the database comprising a plurality of instruction records with each instruction record in the database associated with one of the instructions of the instruction set, each instruction record including entries defining conditional execution of the associated instruction, a target processing element of the associated instruction, an execution unit of the target processing element, data types of the associated instruction and operands for each data type.

2. The apparatus of claim 1 further comprising means for generating multiple test vectors to set up and check state information for packed data type instructions.

3. The apparatus of claim 1 further comprising:
means for parameterizing test vectors; and

means for generating self-checking codes from the parameterized test vectors.

4. The apparatus of claim 1 further comprising means for parameterizing test vectors to create a parameterization; and

means for mapping the parameterization.

5. The apparatus of claim 1 wherein the regular instruction set is further defined in that each instruction has four parts delineated by periods with the four parts always in the same order to facilitate easy parsing by automated tools.

6. The apparatus of claim 1 wherein, every instruction has an instruction name; instructions that support conditional execution forms may have a leading (T. or F.) flag; arithmetic instructions may set a conditional execution state based on one of four flags (C=carry, N=sign, V=overflow, Z=zero); instructions that can be executed on both an SP and a PE or PEs specify the target processor via (.S or .P) designations, instructions without an .S or .P designation are SP control instructions; arithmetic instructions always specify which unit or units that they execute on (A=ALU, M=MAU, D=DSU); load/store instructions do not specify which unit; arithmetic instructions (ALU,MAU,DSU) have data types to specify the number of parallel operations that the instruction performs, the size of the data type and optionally the sign of the operands (S=Signed, U=Unsigned); and load/store instructions have data types (D=doubleword, W=word, H=high, H0=tow halfword, B0=byte0).

7. The apparatus of claim 1 wherein each instruction record further includes the number of cycles the instruction takes to execute (CYCLES), encoding tables for each field in the instruction (ENCODING) and configuration information (CONFIG) for subsetting the instruction set.

8. The apparatus of claim 1 further comprising an instruction-description data structure for an instruction.

9. The apparatus of claim 8 further comprising a second data structure defining input and output state for the instruction.

10. An array processing method comprising the steps of:

establishing a regular instruction set;

executing the instructions of the instruction set by an array of processing elements; and

constructing a database for the instruction set, the database comprising a plurality of instruction records with each instruction record in the database associated with one of the instructions of the instruction set, each instruction record including entries defining conditional execution of the associated instruction, a target processing element of the associated instruction, an execution unit of the target processing.
element, data types of the associated instruction and operands for each data type.

11. The method of claim 10 further comprising the step of establishing an instruction-description data structure for an instruction.

12. The method of claim 11 further comprising the step of establishing a second data structure defining input and output state for the instruction.

13. The method of claim 10 further comprising the step of generating multiple test vectors to set up and check state information for packed data type instructions.

14. The method of claim 10 further comprising the steps of:
parameterizing test vectors; and

generating self-checking codes from the parameterized test vectors.

15. The method of claim 10 further comprising the steps of:
parameterizing test vectors to create a parameterization; and

mapping the parameterization.

16. The apparatus of claim 10 further comprising the step of defining each instruction in the regular instruction set as having four parts delineated by periods with the four parts always in the same order to facilitate easy parsing by automated tools.

17. The apparatus of claim 10 further comprising the step of defining every instruction as having an instruction name; instructions that support conditional execution forms as having a leading (T. or F.) flag; utilizing arithmetic instructions to set a conditional execution state based on one of four flags (C=carry, N=sign, V=overflow, Z=zero); specifying for instructions that can be executed on both an SP and a PE or PEs the target processor via (.S or .P) designations, and defining instructions without an .S or .P designation as SP control instructions; specifying for arithmetic instructions which unit or units that they execute on (A=ALU, M=MAU, D=DSU); not specifying for load/store instructions which unit; arithmetic instructions (ALU, MAU, DSU) having data types to specify the number of parallel operations that the instruction performs, the size of the data type and optionally the sign of the operands (S=Signed, U=Unsigned); and load/store instructions have data types (D=doubleword, W=word, H1=high halfword, H0=low halfword, B0=byte0).

18. The method of claim 10 further comprising the step of establishing each instruction record as further including the number of cycles the instruction takes to execute (CYCLES), encoding tables for each field in the instruction (ENCODING) and configuration information (CONFIG) for subsetting the instruction set.
FIELD OF THE INVENTION

The present invention relates generally to improvements to parallel processing, and more particularly to such processing in the framework of a ManArray architecture and instruction syntax.

BACKGROUND OF THE INVENTION

A wide variety of sequential and parallel processing architectures and instruction sets are presently existing. An ongoing need for faster and more efficient processing arrangements has been a driving force for design change in such prior art systems. One response to these needs have been the first implementations of the ManArray architecture. Even this revolutionary architecture faces ongoing demands for constant improvement.

SUMMARY OF THE INVENTION

To this end, the present invention addresses a host of improved aspects of this architecture and a presently preferred instruction set for a variety of implementations of this architecture as described in greater detail below. Among the advantages of the improved ManArray architecture and instruction set described herein are that the instruction syntax is regular. Because of this regularity, it is relatively easy to construct a database for the instruction set. With the regular syntax and with the instruction set represented in database form, developers can readily create tools, such as assemblers, disassemblers, simulators or test case generators using the instruction database. Another aspect of the present invention is that the syntax allows for the generation of self-checking codes from parameterized test vectors. As addressed further below, parameterized test case generation greatly simplifies maintenance. It is also advantageous that parameterization can be fairly easily mapped.

These and other features, aspects and advantages of the invention will be apparent to those skilled in the art from the following detailed description taken together with the accompanying drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 illustrates an exemplary ManArray 2.times.2 iVLIW processor showing the connections of a plurality of processing elements connected in an array topology for implementing the architecture and instruction syntax of the present invention;

FIG. 2 illustrates an exemplary test case generator program in accordance with the present invention;

FIG. 3 illustrates an entry from an instruction-description data structure for a multiply instruction (MPY); and

FIG. 4 illustrates an entry from an MAU-answer set for the MPY instruction.

DETAILED DESCRIPTION
Further details of a presently preferred ManArray core, architecture, and instructions for use in conjunction with the present invention are found in U.S. patent application Ser. No. 08/885,310 filed Jun. 30, 1997, now U.S. Pat. No. 6,023,753,

U.S. patent application Ser. No. 09/169,072, filed Oct. 9, 1998, now U.S. Pat. No. 6,219,776,
U.S. patent application Ser. No. 09/187,539 filed Nov. 6, 1998, now U.S. Pat. No. 6,151,668,
U.S. patent application Ser. No. 09/228,374 filed Jan. 12, 1999 now U.S. Pat. No. 6,216,223,
U.S. patent application Ser. No. 09/238,446 filed Jan. 28, 1999, now U.S. Pat. No. 6,366,999,
U.S. patent application Ser. No. 09/267,570 filed Mar. 12, 1999, now U.S. Pat. No. 6,446,190,
U.S. patent application Ser. No. 09/337,839 filed Jun. 22, 1999,
U.S. patent application Ser. No. 09/422,015 filed Oct. 21, 1999 now U.S. Pat. No. 6,408,382,
U.S. patent application Ser. No. 09/432,705 filed Nov. 2, 1999 entitled "Methods and Apparatus for Improved Motion Estimation for Video Encoding",
U.S. patent application Ser. No. 09/472,372 filed Dec. 23, 1999 now U.S. Pat. No. 6,256,683,
U.S. patent application Ser. No. 09/596,103 filed Jun. 16, 2000, now U.S. Pat. No. 6,397,324,
U.S. patent application Ser. No. 09/598,566 entitled "Methods and Apparatus for Generalized Event Detection and Action Specification in a Processor" filed Jun. 21, 2000, and
U.S. patent application Ser. No. 09/598,567 entitled "Methods and Apparatus for Improved Efficiency in Pipeline Simulation and Emulation" filed Jun. 21, 2000,
U.S. patent application Ser. No. 09/598,564 filed Jun. 21, 2000, now U.S. Pat. No. 6,622,234,

U.S. patent application Ser. No. 09/598,558 entitled "Methods and Apparatus for Providing Manifold Array (ManArray) Program Context Switch with Array Reconfiguration Control" filed Jun. 21, 2000, and

U.S. patent application Ser. No. 09/598,084 filed Jun. 21, 2000, now U.S. patent No. 6,654,870, as well as,

Provisional Application Ser. No. 60/113,637 entitled "Methods and Apparatus for Providing Direct Memory Access (DMA) Engine" filed Dec. 23, 1998,

Provisional Application Ser. No. 60/113,555 entitled "Methods and Apparatus Providing Transfer Control" filed Dec. 23, 1998,

Provisional Application Ser. No. 60/139,946 entitled "Methods and Apparatus for Data Dependent Address Operations and Efficient Variable Length Code Decoding in a VLIW Processor" filed Jun. 18, 1999,

Provisional Application Ser. No. 60/140,245 entitled "Methods and Apparatus for Generalized Event Detection and Action Specification in a Processor" filed Jun. 21, 1999,

Provisional Application Ser. No. 60/140,163 entitled "Methods and Apparatus for Improved Efficiency in Pipeline Simulation and Emulation" filed Jun. 21, 1999,

Provisional Application Ser. No. 60/140,162 entitled "Methods and Apparatus for Initiating and Re-Synchronizing Multi-Cycle SIMD Instructions" filed Jun. 21, 1999,

Provisional Application Ser. No. 60/140,244 entitled "Methods and Apparatus for Providing One-By-One Manifold Array (1.x.1 ManArray) Program Context Control" filed Jun. 21, 1999,

Provisional Application Ser. No. 60/140,325 entitled "Methods and Apparatus for Establishing Port Priority Function in a VLIW Processor" filed Jun. 21, 1999,

Provisional Application Ser. No. 60/140,425 entitled "Methods and Apparatus for Parallel Processing Utilizing a Manifold Array (ManArray) Architecture and Instruction Syntax" filed Jun. 22, 1999,

Provisional Application Ser. No. 60/165,337 entitled "Efficient Cosine Transform Implementations on the ManArray Architecture" filed Nov. 12, 1999, and

Provisional Application Ser. No. 60/171,911 entitled "Methods and Apparatus for DMA Loading of Very Long Instruction Word Memory" filed Dec. 23, 1999,

Provisional Application Ser. No. 60/184,668 entitled "Methods and Apparatus for Providing Bit-Reversal and Multicast Functions Utilizing DMA Controller" filed Feb. 24, 2000,
In a presently preferred embodiment of the present invention, a ManArray 2×2 iVLIW single instruction multiple data stream (SIMD) processor 100 shown in FIG. 1 contains a controller sequence processor (SP) combined with processing element-0 (PE0) SP/PE0101, as described in further detail in U.S. application Ser. No. 09/169,072 entitled "Methods and Apparatus for Dynamically Merging an Array Controller with an Array Processing Element". Three additional PEs 151, 153, and 155 are also utilized to demonstrate improved parallel array processing with a simple programming model in accordance with the present invention. It is noted that the PEs can be also labeled with their matrix positions as shown in parentheses for PE0 (PE00) 101, PE1 (PE01)151, PE2 (PE10) 153, and PE3 (PE11) 155. The SP/PE0101 contains a fetch controller 103 to allow the fetching of short instruction words (SIWs) from a B=32-bit instruction memory 105. The fetch controller 103 provides the typical functions needed in a programmable processor such as a program counter (PC), branch capability, digital signal processing eventpoint loop operations, support for interrupts, and also provides the instruction memory management control which could include an instruction cache if needed by an application. In addition, the SIW I-Fetch controller 103 dispatches 32-bit SIWs to the other PEs in the system by means of a 32-bit instruction bus 102.

In this exemplary system, common elements are used throughout to simplify the explanation, though actual implementations are not so limited. For example, the execution units 131 in the combined SP/PE0101 can be separated into a set of execution units optimized for the control function, e.g. fixed point execution units, and the PE0 as well as the other PEs 151, 153 and 155 can be optimized for a floating point application. For the purposes of this description, it is assumed that the execution units 131 are of the same type in the SP/PE0 and the other PEs. In a similar manner, SP/PE0 and the other PEs use a five instruction slot iVLIW architecture which contains a very long instruction word memory (VIM) memory 109 and an instruction decode and VIM controller function unit 107 which receives instructions as dispatched from the SP/PE0's I-Fetch unit 103 and generates the VIM addresses-and-control signals 108 required to access the iVLIWs stored in the VIM. These iVLIWs are identified by the letters SLAMD in VIM 109. The loading of the iVLIWs is described in further detail in U.S. patent application Ser. No. 09/187,539 entitled "Methods and Apparatus for Efficient Synchronous MIMD Operations with iVLIW
PE-to-PE Communication”. Also contained in the SP/PE0 and the other PEs is a common PE configurable register file 127 which is described in further detail in U.S. patent application Ser. No. 09/169,255 entitled "Methods and Apparatus for Dynamic Instruction Controlled Reconfiguration Register File with Extended Precision”.

Due to the combined nature of the SP/PE0, the data memory interface controller 125 must handle the data processing needs of both the SP controller, with SP data in memory 121, and PE0, with PE0 data in memory 123. The SP/PE0 controller 125 also is the source of the data that is sent over the 32-bit broadcast data bus 126. The other PEs 151, 153, and 155 contain common physical data memory units 123', 123", and 123” though the data stored in them is generally different as required by the local processing done on each PE. The interface to these PE data memories is also a common design in PEs 1, 2, and 3 and indicated by PE local memory and data bus interface logic 157, 157' and 157".

Interconnecting the PEs for data transfer communications is the cluster switch 171 more completely described in U.S. Pat. No. 6,023,753 entitled "Manifold Array Processor”, U.S. application Ser. No. 09/949,122 entitled "Methods and Apparatus for Manifold Array Processing", and U.S. application Ser. No. 09/169,256 entitled "Methods and Apparatus for ManArray PE-to-PE Switch Control”. The interface to a host processor, other peripheral devices, and/or external memory can be done in many ways. The primary mechanism shown for completeness is contained in a direct memory access (DMA) control unit 181 that provides a scalable ManArray data bus 183 that connects to devices and interface units external to the ManArray core. The DMA control unit 181 provides the data flow and bus arbitration mechanisms needed for these external devices to interface to the ManArray core memories via the multiplexed bus interface represented by line 185. A high level view of a ManArray Control Bus (MCB) 191 is also shown.

Turning now to specific details of the ManArray architecture and instruction syntax as adapted by the present invention, this approach advantageously provides a variety of benefits. Among the benefits of the ManArray instruction syntax, as further described herein, is that first the instruction syntax is regular. Every instruction can be deciphered in up to four parts delimited by periods. The four parts are always in the same order which lends itself to easy parsing for automated tools. An example for a conditional execution (CE) instruction is shown below:

(CE).(NAME).(PROCESSOR/UNIT).(DATATYPE)

Below is a brief summary of the four parts of a ManArray instruction as described herein:

(1) Every instruction has an instruction name.

(2A) Instructions that support conditional execution forms may have a leading (T. or F.) or . . .

(2B) Arithmetic instructions may set a conditional execution state based on one of four flags (C=carry, N=sign, V=overflow, Z=zero).

(3A) Instructions that can be executed on both an SP and a PE or PEs specify the target processor via (.S or .P) designations. Instructions without an .S or .P designation are SP control instructions.
(3B) Arithmetic instructions always specify which unit or units that they execute on (A=ALU, M=MAU, D=DSU).

(3C) Load/Store instructions do not specify which unit (all load instructions begin with the letter `L` and all stores with letter `S`).

(4A) Arithmetic instructions (ALU, MAU, DSU) have data types to specify the number of parallel operations that the instruction performs (e.g., 1, 2, 4 or 8), the size of the data type (D=64 bit doubleword, W=32 bit word, H=16 bit halfword, B=8 bit byte, or FW=32 bit floating point) and optionally the sign of the operands (S=Signed, U=Unsigned).

(4B) Load/Store instructions have single data types (D=doubleword, W=word, H1=high halfword, H0=low halfword, B0=byte0).

The above parts are illustrated for an exemplary instruction below: ##STR1##

Second, because the instruction set syntax is regular, it is relatively easy to construct a database for the instruction set. The database is organized as instructions with each instruction record containing entries for conditional execution (CE), target processor (PROCS), unit (UNITS), datatypes (DATATYPES) and operands needed for each datatype (FORMAT). The example below using TcLsyntax, as further described in J. Ousterhout, Tcl and the Tk Toolkit, Addison-Wesley, ISBN 0-201-63337-X, 1994, compactly represents all 196 variations of the ADD instruction.

The 196 variations come from (CE)*(PROCS)*(UNITS)*(DATATYPES)=7 *2*2*7=196. It is noted that the `e` in the CE entry below is for unconditional execution.

set instruction(ADD,CE) {e t. f. c n v z}

set instruction(ADD,PROCS) {s p}

set instruction(ADD,UNITS) {a m}

set instruction(ADD,DATATYPES) {1d 1w 2w 2h 4h 4b 8b}

set instruction(ADD,FORMAT,1d) {RTE RXE RYE}

set instruction(ADD,FORMAT,1w) {RT RX RY}

set instruction(ADD,FORMAT,2w) {RTE RXE RYE}

set instruction(ADD,FORMAT,2h) {RT RX RY}

set instruction(ADD,FORMAT,4h) {RTE RXE RYE}

set instruction(ADD,FORMAT,4b) {RT RX RY}
set instruction(ADD,FORMAT,8b) {RTE RXE RYE}

The example above only demonstrates the instruction syntax. Other entries in each instruction record include the number of cycles the instruction takes to execute (CYCLES), encoding tables for each field in the instruction (ENCODING) and configuration information (CONFIG) for subsetting the instruction set. Configuration information (1.times.1, 1.times.2, etc.) can be expressed with evaluations in the database entries:

```tcl
proc Manta {} {
    # are we generating for Manta?
    return 1

    # are we generating for ManArray?
    # return 0
}
```

Having the instruction set defined with a regular syntax and represented in database form allows developers to create tools using the instruction database. Examples of tools that have been based on this layout are:

- Assembler (drives off of instruction set syntax in database),
- Disassembler (table lookup of encoding in database),
- Simulator (used database to generate master decode table for each possible form of instruction), and
- Testcase Generators (used database to generate testcases for assembler and simulator).

Another aspect of the present invention is that the syntax of the instructions allows for the ready generation of self-checking code from test vectors parameterized over conditional execution/datatypes/sign-extension/etc. TCgen, a test case generator, and LSgen are exemplary programs that generate self-checking assembly programs that can be run through a Verilog simulator and C-simulator.

An outline of a TCgen program 200 in accordance with the present invention is shown in FIG. 2. Such programs can be used to test all instructions except for flow-control and iVLIW instructions. TCgen uses two data structures to accomplish this result. The first data structure defines instruction-set syntax (for which datatypes/ce[1,2,3]/sign extension/rounding/operands is the instruction defined) and semantics (how many cyles/does the instruction require to be executed, which operands are immediate operands, etc.). This data structure is called the instruction-description data structure.
An instruction-description data structure 300 for the multiply instruction (MPY) is shown in FIG. 3 which illustrates an actual entry out of the instruction-description for the multiply instruction (MPY) in which e stands for empty. The second data structure defines input and output state for each instruction. An actual entry out of the MAU-answer set for the MPY instruction 400 is shown in FIG. 4. State can contain functions which are context sensitive upon evaluation. For instance, when defining an MPY test vector, one can define: RX.sub.b (RX before)=maxint, RY.sub.b (RY before)=maxint, RT.sub.a =maxint*maxint. When TCgen is generating an unsigned word form of the MPY instruction, the maxint would evaluate to 0.times.ffffffff. When generating an unsigned halfword form, however, it would evaluate to 0.times.ff. This way the test vectors are parameterized over all possible instruction variations. Multiple test vectors are used to set up and check state for packed data type instructions.

The code examples of FIGS. 3 and 4 are in Tcl syntax, but are fairly easy to read. "Set" is an assignment, ( ) are used for array indices and the { } are used for defining lists. The only functions used in FIG. 4 are "maxint", "minint", "sign0unsil", "signlunsi0", and an arbitrary arithmetic expression evaluator (mpexpr). Many more such functions are described herein below.

TCgen generates about 80 tests for these 4 entries, which is equivalent to about 3000 lines of assembly code. It would take a long time to generate such code by hand. Also, parameterized testcase generation greatly simplifies maintenance. Instead of having to maintain 3000 lines of assembly code, one only needs to maintain the above defined vectors. If an instruction description changes, that change can be easily made in the instruction-description file. A configuration dependent instruction-set definition can be readily established. For instance, only having word instructions for the ManArray, or fixed point on an SP only, can be fairly easily specified.

Test generation over database entries can also be easily subset. Specifying "SUBSET(DATATYPES) {1sw 1sh}" would only generate testcases with one signed word and one signed halfword instruction forms. For the multiply instruction (MPY), this means that the unsigned word and unsigned halfword forms are not generated. The testcase generators TeIRita and TelRitaCorita are tools that generate streams of random (albeit with certain patterns and biases) instructions. These instruction streams are used for verification purposes in a co-verification environment where state between a C-simulator and a Verilog simulator is compared on a per-cycle basis.

Utilizing the present invention, it is also relatively easy to map the parameterization over the test vectors to the instruction set since the instruction set is very consistent.

Further aspects of the present invention are addressed in the documentation which follows below. This documentation is divided into the following principle sections:

Section I--Table of Contents;
Section II--Programmer's User's Guide (PUG);
Section III--Programmer's Reference (PREF).
The Programmer's User's Guide Section addresses the following major categories of material and provides extensive details thereon: (1) an architectural overview; (2) processor registers; (3) data types and alignment; (4) addressing modes; (5) scalable conditional execution (CE); (6) processing element (PE) masking; (7) indirect very long instruction words (iVLIWs); (8) looping; (9) data communication instructions; (10) instruction pipeline; and (11) extended precision accumulation operations.

The Programmer's Reference Section addresses the following major categories of material and provides extensive details thereof: (1) floating-point (FP) operations, saturation and overflow; (2) saturated arithmetic; (3) complex multiplication and rounding; (4) key to instruction set; (5) instruction set; (6) instruction formats, as well as, instruction field definitions.
While the present invention has been disclosed in the context of various aspects of presently preferred embodiments, it will be recognized that the invention may be suitably applied to other environments and applications consistent with the claims which follow.