Very Long Instruction Word: A Higher Performance Processor

Adedoyin Odumabo
PhD Student
Computer Science Dept.,
Lagos State University,
Nigeria

Ademola Adedokun
PhD Student
Computer Science Dept.,
Lagos State University,
Nigeria

Akinlolu Adekotujo
PhD Student
Computer Science Dept.,
Lagos State University,
Nigeria

Oluwatosin Ogunbodede
PhD Student
Computer Science Dept.,
Lagos State University,
Nigeria

ABSTRACT
The quest for improvements in the execution of system instructions whilst upholding processor efficiency begot different discoveries. Hardware solutions, such as having multiprocessors seem to be more expensive and had some challenges with it. Over the years, different software approaches have been discovered and explored. Unfortunately, some of these approaches still hugely depend on the permissibility of the hardware facility which could hinder productivity. Very Long Instruction Word (VLIW) processor adopts an instruction level parallelism (ILP) method that is fully software based. This paper provides an overview, performs a comparison with other related architectures such as CISC, RISC and highlights VLIW's development over the years as well as reasons to adopt this technology. The methodology and techniques used in this work include analysis of available documents, and content analysis techniques. This Analysis shows that VLIW machine performed better compared to other traditional machines using instruction's size, format, semantics as well as memory, registers and hardware design as metrics for comparison. VLIW is indeed a step ahead of the others and allows higher performance without the complexity inherent in other designs.

General Terms
Microprocessor Architecture

Keywords
Very long instruction word

1. INTRODUCTION
There has been much development in microprocessor architecture such as Complex Instructional Set Computing (CISC) and Reduced Instructional Set Computing (RISC). The advancement in integrated circuit (IC) technology and high-level language compiler technology, will force vendors to move with this new trend of VLIW in microprocessor instruction sets. A processor traditionally executes every instruction one after the other. This makes for inefficient and ineffective use of processor resources, yielding potentially poor performance [1]. The efforts towards improving the performance have yielded different hardware and software solutions. Most developments that are software based have hardware issues. This led to VLIW architectures which are purely software solutions and rely on compile-time detection of parallelism [2]. The compiler examines the program and identifies operations to be executed in parallel. After one instruction has been fetched all the corresponding operations are issued in parallel. No hardware component is needed for run-time detection of parallelism [2]. The compiler can potentially analyze the program to detect parallel operations unlike the hardware complexity encountered in pipelining, super scalar architecture and out of order execution.

VLIW executes operations in parallel, based on a fixed schedule, determining when programs are compiled. Since determining the order of execution of operations is handled by the compiler, the processor does not need the scheduling hardware as required by other methods [3]. Thus, VLIW CPUs offer more computing prowess with less hardware complexity than most superscalar CPUs. VLIW architectures are simpler and cheaper than RISC because of enhanced hardware modifications and simplifications. They however, require more compiler support. It is different from a vector processor, in that no high level regularity in the user's code is needed to make effective and efficient use of the hardware support. So also unlike a multiprocessor, there is no penalty for communication and synchronization between processes [1].

This paper provides an overview, performs a comparison with other related architectures such as CISC, RISC and highlights VLIW's development over the years as well as reasons to adopt this technology.

The paper is structured as follows: Section 2 presents the overview of VLIW while Section 3 presents the comparative analysis of CISC, RISC and VLIW. Section 4 presents advantages and disadvantages of VLIW. VLIW scheduling Algorithm is discussed in section 5 and Conclusion is presented in Section 6.

2. OVERVIEW OF VLIW
The idea was firstly conceptualized by Josh Fisher in his research at Yale University (1983). His development of Trace Scheduling algorithm in the practice of horizontal microcode compaction could exploit more ILP than any existing processor could provide. Before VLIW, the notion of pre-scheduling execution units and instruction-level parallelism in software was well established in the practice of developing horizontal microcode[4]. John Ellis describes the first VLIW compiler in his Ph.D. thesis under the supervision of Fisher. The compiler was named Bulldog, after Yale's mascot. Fisher left Yale University in 1984 to startup a company named Multiflow, along with cofounders John O'Donnell and John Ruttenberg. Multiflow produced the TRACE series of VLIW system in 1987[4]

The Very Long Instruction Word (VLIW) refers to a type of instruction set architecture designed to exploit instruction level parallelism and performance. VLIW is sometimes viewed as the next step beyond the Reduced Instruction Set Computing (RISC) architecture. In a VLIW processor, one instruction encodes multiple operations, taking one operation

for each execution unit of a device at the least [3]. The whole operation will be broken down into multiple functioning units for concurrent and seamless execution [5]. The main objective of VLIW is to eradicate the complicated or complex instruction scheduling and parallel dispatch that ensues in most common modern microprocessors.

Very-long-instruction-word (VLIW) is a processor that describes computer-system architecture where a language compiler breaks down program instruction into basic and distinct actions that can be accomplished by the processor in parallel, performing all these actions at same time [6, 7] It is a technique that makes use of and combines multiple standard instructions into one long instruction word. The instructions are very intuitive and make use small variants of logic instead of large unreadable chunks.

Central processing unit (CPU) allows programs to state how its instructions will be executed and this is mostly done sequentially. Whereas, Very Long Instruction Word (VLIW) processors also specifically allow written programs to clearly specify how its instructions should be executed and this is often done concurrently albeit in parallel. This is a design designated to increase performance of a computer system in order to avoid complexity that is frequently found in other designs.

However, since Instruction level processors try to increase the speed at which a program is executed in parallel rather than the conventional sequential programs, it belongs to a processor family that enhances the speed of execution of programs or instructions with its design, allowing individual systems or machines to execute instructions in parallel. ILP has powered the highest performance uniprocessors for decades with several systems built on it, and this makes ILP alongside its new techniques a popular approach [8].

One of the major keys to higher performance in microprocessors is that it allows large range of applications [9]. This is because VLIW employed fine-grain, instruction-level parallelism which includes the followings:

a. Pipelining: It is now used in the design of high performance processors.

b. Multiple processors: The use of multiple processors improves performance for only a restricted set of applications.

c. Superscalar implementation: It uses improved performance for all types of applications. Superscalar simply means the ability to fetch and complete more than one instruction at a time. The implementation of superscalar is an important requirement if architectural compatibility software is to be preserved which will ensure software legacy.

d. Specifying multiple operations per instruction: It creates very long instruction word architecture. VLIW and superscalar are similar in the sense that they issue and complete more than one operation at a time. For the VLIW implementation, the long instruction word encodes the concurrent operations. This encoding leads to great reduction in hardware complexity compared to a high-degree superscalar implementation in RISC or CISC.

2.1 Basic Structure of Vliw Architecture

Very Long Instruction Words (VLIW) is fetched directly from the memory, via a common register file that has multiple ports for collecting the operands and safeguarding results. Parallel random-access to the register-file connected to the memory, is only possible through the read and write cross bar. Execution in the various functional units is carried out concurrently and in parallel with the load and store operation of data between Random Access Memory (RAM) and the register file.

VLIW's are characterized by a single stream of execution (One program counter, and one control unit); a very long instruction format that provides adequate control bits to independently and directly control the actions of every functional unit in every cycle; and very large numbers of functional units and data paths, in which the control is planned at specified compile times.

VLIWs use multiple, independent functional units separately grouped. Instead of attempting to issue multiple, independent instructions to the aforementioned units, a VLIW bundles the multiple operations into one very long instruction [8]**.** Multiple functional units execute all the operations in an instruction concurrently, providing fine-grain parallelism within each instruction. Also, instructions directly control the hardware with no interpretation and minimal decoding. A powerful optimizing compiler is responsible for locating and extracting ILP from the program and for scheduling operations to exploit the available parallel resources.

The differences in these architectures however, have direct effects on the implementations. These architectures use the traditional-state machine-model of computation. Each instruction effects an increased change in the memory and register state of the computer, and the hardware fetches and executes instructions sequentially until a branch instruction generates a change control flow direction [7].

2.2 Vliw Processors

Early VLIW processors were established to be scientific super computers; newer processors were used mainly for stream, image and digital signal processing, low power mobile computers, multimedia codec hardware, etc. Recently, some of these processors enjoyed moderate commercial success in Philips Trimedia, TI TMS320C62x DSPs, and Intel Itanium than Transmeta Crusoe model. Also, Fu et. al. implemented an Executable Quantum Instruction Set Architecture (eQASM) which combined efficient timing specification, single-operation-multiple-qubit (SOMQ) execution, and VLIW architecture, relieves the quantum operation issue rate problem, by presenting better scalability than Quantum microarchitecture set (QuMIS) [10]. Moreover, VLIW compiler technologies are equally applicable to super scalar processors. Stream and media processing applications are used with predictable branch behavior and large amounts of ILP [9]. Examples of VLIW processors are Defoe, Transmeta Crusoe, Intel IA-64, Multiflow TRACE, Cydrome Cydra etc.

2.2.1 Defoe

Defoe is a 64-bit VLIW architecture and it is compressed. It has a set of 64 programmer visible general purpose registers that are 64 bits wide. The predicate registers are special 1-bit registers, which require a true or false value and register R0 always contains zero. There are 16 programmer visible predicate registers called PR0 to PR15. All operations are predicated, i.e. each instruction contains a predicate register field (PR) that contains a predicate register number. In an uncompressed VLIW architecture, MultiOps have a fixed length i.e. operations are not accessible within a MultiOp, NOPs are inserted into those slots, while a compressed VLIW architecture uses variable length MultiOps to get rid of those NOPs and achieve high code density. Also, operations are

encoded as 32-bit words. A special stop bit in the 32-bit word indicates the end of a MultiOp.

2.2.2 The Intel Itanium Processor

The Itanium-1 processor is Intel's major implementation of the IA-64 ISA series, which is developed both by Intel and HP. It is an Explicitly Parallel Instruction Computing (EPIC) form of VLIW which consist a 64-bit, 6 issue VLIW processor with four integer units, four multimedia units, two load/store units, two extended precision floating point units and two single precision floating point units. This processor running at 800 MHz on a 0.18 micron process and has a 10 stage pipeline. The IA-64 architecture used a fixed-width bundled instruction format with dependent instructions. MultiOp is called instruction group, where each MultiOp consists of one or more 128 bit bundles. However, each 128 bit bundle consists of three operations and a template. The opcode in each operation specifies a type field; the template encodes commonly used combinations of operation types. Defoe and IA-64 uses a decoupling buffer to improve its issue rate. Though the IA-64 registers are nominally 64 bits wide, there is a hidden 65th bit called NaT (Not a Thing). Also, all operations are predicated with 64 predicate registers.

2.2.3 The Transmeta Crusoe Processor

The Crusoe processor was developed by Transmeta Corporation. The Crusoe was designed to build a processor with modest performance and efficiently outdo the ISA of other processors, mainly the 80x86 and the Java virtual machine. In the design, complex mechanisms of achieving ILP were replaced with simpler and more power efficient alternatives; and whereas out of order issue and dynamic scheduling were not considered. In the Crusoe, long instructions are either 64 or 128 bits; a 128-bit instruction word is called a molecule which encodes 4 operations called atoms. The molecule format determines how operations are sent to functional units. The Crusoe consist of two integer units, a floating point unit, a load/store unit and a branch unit. It has 64 general purpose registers and uses condition flags compared to predication used by Defoe.

Table1: Architectural comparison of CISC, RISC and VLIW

ARCHITECTURE CHARACTERISTIC	CISC	RISC	VLIW
Instruction size	Varies	One size, usually 32 bits	One size, usually 64 or 128 bits
Instruction format	Field placement varies	Regular, consistent placement of fields	Many simple, independent operations
Instruction semantics	Varies from simple to complex	Almost always one simple operation	Many simple, independent operation
Registers	Few, sometimes special	Many, general purpose	Many, general purpose
Memory references	Bundled with operations in many different types of instructions	Not bundled with operations, i.e. load/store architecture	Not bundled with operations, i.e. load/store architecture
Hardware design focus	Exploit microcode implementations	Exploit implementations with one pipeline and no microcode	Exploit implementations with multiple pipeline, no microcode & no complex dispatch logic

3. COMPARISON OF CISC, RISC, AND VLIW

The differences between CISC, RISC and VLIW are in the formats and semantics of the instructions. Table 1 compares architecture characteristics.

CISC instructions vary in size, often specify a sequence of operations, and can require serial (slow) decoding algorithms. CISCs tend to have few registers, and the registers may be special-purpose, which restricts the ways in which they can be used. Memory references are typically combined with other operations (such as add memory to register). CISC instruction sets are designed to take advantage of microcode.

RISC instructions specify simple operations, are fixed in size, and are easy (quick) to decode. RISC architectures have a relatively large number of general-purpose registers. Instructions can reference main memory only through simple load-register-from-memory and store-register-to -memory operations. RISC instruction sets do not need microcode and are designed to simplify pipelining.

VLIW instructions are like RISC instructions except that they are longer to allow them to specify multiple, independent simple operations. A VLIW instruction can be thought of as several RISC instructions joined together. VLIW architectures tend to be RISC-like in most attributes.

4. SCHEDULING ALGORITHMS OF VLIW

Instruction scheduling algorithms are important to the performance of a VLIW processor. The writing of code for VLIW processors is difficulty even more than super scalar processor. Why? The super scalar processor program is

fundamentally sequential and depends on the hardware to extract parallelism from the sequential program, while during VLIW processor code generation; the compiler is met with the task of concurrently extracting parallelism from a sequential algorithm and scheduling independent operations [9]. Below are the three main scheduling algorithms:

4.1 Trace Scheduling

The first-generation ILP processors/compilers used a three phase method to generate code which failed such as create a sequential program and analyze each block for independent operations; if sufficient hardware resources are available, list independent operations within the same block in parallel; and when possible, move operations between blocks. The failure is because operations in a basic block are dependent on each other and choices made during scheduling basic blocks may make it difficult to move operations between blocks. Therefore, trace scheduling algorithm was introduction to solve the above problems by Joseph Fisher. In trace scheduling, a set of commonly executed sequence of blocks is gathered together into a trace and the whole trace is scheduled together. Trace scheduling operates on a linear sequence of blocks i.e. it allow operations from one side of conditional branch. It typically misses code motions that move operations from one trace to another. Trace scheduling operates such that extracting ILP from sequential programs will require code motion across multiple basic blocks. It is driven by profile data not static branch analysis. The trace scheduling algorithm can be seen in B Mathew's work [9].

4.2 Trace Scheduling-2

Trace scheduling - 2 is an advancement on trace scheduling which allows nonlinear code motion, i.e. it allows operations from both sides of a conditional branch. It uses an expected value function called speculative yield to consider the cost of speculative execution and resolve whether or not to move operations from one block to another. This algorithm works by picking clusters of operations where each cluster is a maximal set of operations that are connected without back edges in the flow graph of the program.

4.3 Super Block Scheduling

Super block scheduling developed partnership with Impact compiler at the University of Illinois. It operates like trace scheduling, by extracting ILP from sequential programs which requires code motion across multiple basic blocks. Super block scheduling is driven by static branch analysis. It is a set of regular basic blocks that control, may arrive at the top only, but may depart at more than one point. Super blocks uses a process called tail duplication which identify traces first and then eliminate side entries into a trace. Static analysis based super block scheduling achieved more results than profile based methods.

5. MERITS AND DEMERITS OF VLIW

In every development especially Microprocessor Architecture has both benefits and drawbacks. The merits and demerits of VLIW are below:

5.1 Merits of VLIW

There are many advantages of VLIW over CISC and RISC. Besides VLIW being easier, simpler and cheaper to assemble, below are its other advantages:

- Increased performance
- It removes complicated instruction scheduling and parallel problems that are found in super scalar approaches
- It is compiler friendly and improves hardware performance
- Handles Interrupts and exceptions relatively easily
- It can highly predict run-time behavior, hence allowing real-time applications and greater result for code optimization
- Simple hardware is required, which is normal and straightforward
- Pure VLIW machines do not need complicated logic to check for dependencies
- Potentially easier to program
- Can add more execution unit compartments and allow more instructions to be packed into the VLIW instruction set.
- Parallelism will be exploited at the instruction level

5.2 Demerits of VLIW

- In some algorithms, parallelism is underutilized
- High power consumption
- Increased memory usage because of high memory bandwidth requirement
- There is no object code compatibility between generations
- Increased code size due to empty "slots"
- Program size is large
- Compiler complexity

6. CONCLUSION

This work compares CISC, RISC AND VLIW, and highlights how VLIW machines perform better than other traditional machine. It buttresses the fact that VLIW architectures are the new trend in microprocessor architecture and were designed to leverage on instruction level parallelism.

Recent and known high performance processors depend on Instruction-Level-Parallelism to achieve high execution speed and performance. ILP processors achieve their high performance ratings by allowing multiple operations to execute in parallel and concurrently, using combined compiler and hardware techniques. Indeed, VLIW is a step ahead of the others and allows higher performance without the complexity inherent in these other designs. The processor will not make any run-time control decision below the program level which helps reduce complexity and increase turnaround time.

7. REFERENCES

[1] R. P. Colwell, R. P. Nix, J. J. O'donnell, D. B. Papworth, and P. K. Rodman, "A VLIW architecture for a trace scheduling compiler." pp. 180-192.

[2] J. A. Fisher, P. Faraboschi, and C. Young, Embedded computing: a VLIW approach to architecture, compilers and tools: Elsevier, 2005.

[3] L. H. John, and A. P. David, Computer Architecture; A Quantitative Approach, fourth ed.: MK Morgan Kaufmann Publishers, 2007.

[4] R. P. Colwell, W. E. Hall, C. S. Joshi, D. B. Papworth, P. K. Rodman, and J. E. Tornes, "Architecture and implementation of a VLIW supercomputer." pp. 910-919.

[5] M. Lam, "Software pipelining: An effective scheduling technique for VLIW machines." pp. 318-328.

[6] A. Aiken, and A. Nicolau, " Perfect pipelining: A new loop parallelization technique ". pp. 221–235.

[7] Schaum's, Outline of Theory and Problems of Computer Architecture The McGraw-Hill Companies Inc. Indian Special Edition, 2009.

[8] B. R. Rau, and J. A. Fisher, "Instruction-level parallel processing: history, overview, and perspective," The journal of Supercomputing, vol. 7, no. 1-2, pp. 9-50, 1993.

[9] B. Mathew, "very long instruction word architecture (VLIW processors and trace scheduling)", 2006.

[10] X. Fu, L. Riesebos, M. A. Rol, J. van Straten, J. van Someren, N. Khammassi, I. Ashraf, R. F. L. Vermeulen, V. Newsum, K. K. L. Loh, J. C. de Sterke, W. J. Vlothuizen, R. N. Schouten, C. G. Almudever, L. DiCarlo, and K. Bertels, "eQASM: An Executable *f* antum Instruction Set Architecture."