EPIC Architecture Data Flow Processors and their Execution

Danijela Jakimovska, Aristotel Tentov, Goran Jakimovski, Sashka Gjorgjievska and Maja Malenko
The Faculty of Electrical Engineering and Information Technologies
Dept. of Computer Science and Engineering, Karposh 2, Skopje
Republic of Macedonia
{toto@feit.ukim.edu.mk}
{goranj@feit.ukim.edu.mk}
{sashka@feit.ukim.edu.mk}
{majam@feit.ukim.edu.mk}
{danijela@feit.ukim.edu.mk}

ABSTRACT: *The complexity of modern processors and the constant competition between different computer technologies led to the development of many different computer architectures, all with their pros and cons, but all with the same goal: to improve the overall performance of the computer systems. In this paper, we'll look at the following computer architectures: RISC architecture, CISC architecture, Superscalar architecture, VLIW architecture, EPIC architecture, and Data flow processor architectures. All of these architectures have their pros and cons. We'll also look at the advantages and disadvantages of each of them. We'll also briefly explain the idea of parallelizing instructions' execution, so that we can focus on the goal of executing more than 1 instruction in one clock cycle to improve the overall system performance. Let's take a look at some commercial implementations of modern computer architectures.*

Keywords: Processor Architectures, RISC, CISC, ILP

Received: 6 May 2023, Revised 21 July 2023, Accepted 30 July 2023

DOI: 10.6025/jio/2023/13/4/112-117

1. Introduction

Modern microprocessors are one of the most complicated systems that have been ever created by the human beings. Microprocessors have the main role in each system, since they handle the instruction and data flow, control the communication with memory and external devices and thus coordinate the whole system operation. Computer architects face with the challenge to maximize computer performance, while retaining the cost, power and functional requirements. Regarding this, they should consider three aspects of computer architecture design, including: instruction set architecture, organization (memory system, memory interconnect, internal processor), and hardware logic design. Therefore, in order to optimize the architecture design, an architect must be familiar with many technologies, such as compilers, operating systems, logic design and packaging.

One of the first computer architectures, such as the Intel IA-32, belongs to the Complex Instruction Set Computer (*CISC*) design which takes advantage of the microcode and is consisted of a wider range of variable-length instructions. In order to reduce the complexity of the instructions, Reduced Instructions Set Computing (*RISC*) was introduced, speeding up the process of decoding the instruction.

Superscalar processor and Very Long Instruction Word (*VLIW*) architectures were introduced in order to achieve Instruction Level Parallelism (*ILP*) exploiting the pipeline mechanism. The main issue in *ILP* is detecting and overcoming data dependencies, which Superscalar processors provide with the hardware, while *VLIW* processors with software support.

Architectures like Explicit Parallel Instruction Computing (*EPIC*) and Data Flow were designed to solve some of the problems detected in the previous parallel architectures. *EPIC* processors overcomes the hardware dependencies and furthermore, Data Flow machines provide concurrency in program execution.

2. RISC, CISC and Superscalar

2.1. CISC Architecture

CISC is designed to use complex instruction set which makes the assembly languages closer to the operations and data structures of the High Level Languages, [1]. *CISC* instruction set is proposed to take advantage of microcode, and is consisted of many variable-length instructions, which can specify a sequence of operations. The *CISC* instructions are characterized with complexity, in terms of instruction formats and addressing modes, and therefore require serial (slow) decoding algorithms, [2]. The memory references are usually combined with other operations, such as add memory data to register. *CISC* processors generally have few registers, and some of them may be special-purpose, which restricts the ways how they can be used, [3].000 Accordingly, the *CISC* architecture complicates the instruction's decoding and scheduling, and therefore is not very suitable for pipelining [4].

The family line of x86 processors is the leader in *CISC* computing, used both as general purpose processors and for embedded systems. Using the x86 processors in embedded design is relatively novel approach and is mainly purposed for the 32- and 64-bit designs such as Intel's Atom, *VIA's* Nano, Athlon's Neo, and VIA's *C*7.

2.2. RISC Architecture

The researches intended to improve the existing (mainly *CISC*) architectures led to reducing the set of instructions and their complexity, and thus *RISC* was designed. *RISC* instruction set consists of simple fix-sized operations, which are easy (quick) for decoding, and therefore suitable for pipelining. Contrary to *CISC*, *RISC* instructions are less complex, support simple addressing modes, and do not require a microcode for their implementation, [2], [4]. *RISC* processors have a relatively large number of general-purpose registers. *RISC* instructions reference to the main memory only via simple load and store operations. This is the main reason why the *RISC* processors are usually referred as loadstore architecture designs, [5].

Widely known success stories behind the *RISC* architecture are *AMD* 29*K*, *ARM*, *SPARC*, Power/*PC*, and *MIPS*. *ARM* takes the lead in embedded systems, such as smart phones and tablet computers, due to its low cost and low power consumption.

2.3. Instruction Level Parallelism

Further research led to an idea, which explains that splitting the work of a single processor to multiple processors will increase the productivity and will speed up the execution of the instructions. The main setback here is that programs were written and were meant to be executed in sequential manner, that is, one instruction at a cycle. This restriction of the sequential execution came from the data dependencies between the variables in a program.

Pipelining is a mechanism which enables instruction level parallelism, since parallel instructions are executed in parallel over multiple cycles. The theoretical increase of the instruction execution speed is proportional to the pipeline length, [2]. However, there are three potential hazard problems that can occur, during the pipeline execution of parallel instructions. Data hazards appear when an instruction result depends on the previous instruction; structural hazards happen when there is not enough hardware space for the parallel instructions execution and control hazards are result of the unexpected program counter change, [1]. There are several mechanisms targeting these problems, and the simplest solution is to stall the pipeline.

2.4. Superscalar architecture

RISC architecture is very suitable for exploiting parallelism through pipelining, since all the *RISC* instructions are simple and take roughly the same time to finish. The instruction execute stage is usually the most time consuming pipeline operation, so if the processor architecture employs multiple execution units, it would always have a busy one, while the others would be idle, [2]. As a solution to this problem, computer architects proposed a superscalar architecture, characterized with parallel instructions execution on multiple executing units. Although this architecture utilizes more execution units simultaneously, the number of hazards is increased and the processor has to retire the instructions in program order if they are re-ordered (dynamic scheduling). Besides that, handling branch operations becomes very problematic, since a typical program executes a branch after each six or seven instructions. A possible solution would be to utilize a special hardware branch predictor, which would predict the target with some probability, on behalf of the previous branch results. However, this approach has still some disadvantages, such as speculatively wrong executed instructions. The processor architects overcome this difficulty, by extending the processor architecture with reorder buffer. This buffer is intended to store the results of speculatively executed instructions, and to update the real state after the correct instructions has completed, [1].

3. VLIW and Epic Architectures

3.1. Very Long Instruction Word

Unlike the Superscalar architecture, which relies on hardware to detect and overcome data dependencies, Very Long Instruction Word (*VLIW*) processors, [3] use software solutions (compilers) to mark independent instructions. Usually compilers translate the code into intermediate language, optionally optimize it, and as well generate machine code for the specific architecture, [6]. *VLIW* processors use compilers that inspect the source code and thus concentrate on scheduling and optimizing the raw source code before translating it.

Scheduling is done using two structures: control flow and data flow graphs, [6]. Control flow graphs divide the source code into basic blocks that must be sequentially executed (usually the delimiter is a branch or a label), [6]. The data flow graph shows the dependencies between registers within a basic block and those which are independent can be executed in parallel, [6]. Here, the disadvantage is that a single basic block usually contains four to six operations and therefore, limits the amount of parallelism that can be achieved. To maximize parallelism, a technique called global scheduling is used, where instructions are moved from one block to another, [6].

Transmeta's Crusoe ® and Texas Instruments 320C6x line of processors are the commercial breakthrough behind the *VLIW* architecture. Transmeta's Crusoe processor uses hybrid hardware-software implementation of the *VLIW* architecture. It is a 128 bit architecture that uses Code Morphing Software that detects and resolves data dependencies. The Code Morphing software also implements routines for power management and thermal dissipation, which makes this processor ideal for mobile devices. The Texas Instruments 320C6x line of processors is a general purpose *DSP* processor using the *VLIW* architecture, which is mainly used as a research processor. It has wide range of debugging tools and compilers available for research.

3.2. Explicitly Parallel Instruction Computing

The explicitly parallel architecture was designed to overcome some essential limitations of the *VLIW* architecture, such as hardware dependence. The *EPIC* architecture solves the hardware dependence problem, by defining several mini-instructions which can be combined in groups, depending on the template type field, [8]. Therefore, the processors which are characterized with greater parallelism capabilities will simply exploit more bundles in parallel. *EPIC* processors utilize dispersal technique to issue two bundles at a time, and split-issue mechanism if the miniinstruction cannot be executed. Furthermore, the instructions can be predicted, reducing the cost of a branch operation. However, the cost for wrong prediction is very high, because branches appear very often. *EPIC* processors provide hardware support for the control speculation of loads and they allow parallel issuing of multiple prioritized branch operations. Speculative load failures are resolved with poison bits utilization, [1], [2].

The *EPIC* architecture doesn't solve all the problems of the *VLIW* architecture. The difficulty with the increased code size (because of the empty slots in the bundles) still remains unsolved. This has a negative impact on cache performance and bus bandwidth utilization. Other very important feature of each *EPIC* processor is the good compiler support, [1].

This architecture has only one implementation, as part of the IA-64 processor architecture of Itanium family processors, [1]. Intel Itanium architecture processors have been designed from the ground up to meet the increasing demands for high availability, scalability and performance needed for high-end enterprise and technical computing applications. In its core, Itanium was designed to address a number of performance bottlenecks in computers, such as memory latency, control flow dependencies and memory address disambiguation. It enables the hardware to take advantage of the available Instruction Level Parallelism and to provide the necessary resources, while focusing on dynamic runtime optimizations.

Precision Architecture – Reduced Instruction Set Computer (*PA-RISC*) [7] was originally designed as a 32-bit architecture, intended to be easily scalable across a broad performance range, while providing for straightforward migration of applications from existing systems. It was rather conservative RISC design, but still competitive in terms of speed, especially for simultaneous multiprocessing and floating-point operations.

4. Dataflow Architectures

Concurrency is a major step in increasing computational performance, especially with today's technological limitations. Dataflow architecture offers an attractive alternative to the conventional control flow architecture in providing concurrency in execution of programs. Execution of each dataflow instruction depends only on the availability of its operands, which implies implicit synchronization of parallel activities. There are no constraints in sequencing of dataflow instructions, except for the conventional data dependencies in a program.

Data flow architecture differs from control-flow architecture, by two basic principles: asynchronous operations and functionality. Dataflow instructions are executed only when all input operands are available (assuming hardware resources are also available), in contrast to control-flow model which uses program counter for sequential ordering of instruction execution. The functionality rule implies that any two enabled instructions can be executed in either order or concurrently, only if they don't interfere with each other (don't have data dependences), which implies parallel processing.

A dataflow program is represented as a directed graph, where named nodes represent instructions and links represent data dependencies among instructions, [9] [10]. Dataflow graphs can be described as machine language for dataflow computers. Data is conveyed from one node to another in data packets called tokens. This flow of tokens enables nodes (instructions)

Architecture/ Characteristics	**ILP**	**Instructions per cycle**	**Instruction format**
RISC	Yes, pipeline	Depends on the pipeline depth (usually 4 or 5)	Fixed length, Usually 16,32,64 etc
CISC	Yes, similar to superscalar	Depends on the operation's complexity	Variable length, Complex operations
Superscalar	Yes, Multiple pipelines	Depends on the pipeline depth (usually 8 or 10)	Fixed length, Usually 16,32,64 etc
VLIW	Yes, Fixed number of instructions in the word, Implicit parallelism	Depends on the number of execution units (usually 4 or 8)	Fixed length, Multiple instructions in one word
EPIC	Yes, variable, but limited number of instructions in the word, Explicit parallelism	Depends on the number of execution units (6 to 8)	Variable length, Multiple microinstructions in the instruction word
Dataflow	Yes, variable, but limited number of instructions (due to operands availability and hardware resources)	Depends on the number of execution/functional units	Packet format of instructions (PISC)

Table 1. Comparison of presented architectures

which depend on them and fires them.

Dataflow architecture can generally be divided into pure dataflow architecture and hybrid dataflow architecture. Pure dataflow firing rule says that an instruction can be executed as soon as all input operands are available to it. It gives the dataflow model asynchronous behavior and self-scheduling of instructions.

Pure dataflow architecture is subsequently divided into static, dynamic and explicit token store architectures, while the hybrid architecture utilizes some known control flow mechanisms, [11]. Pure dataflow architecture executes a program by receiving tokens, each containing data and tag, processing instructions and sending out newly formed tokens, [9]. When a set of matched tokens (tokens with same tag) is available at the execution unit, processing starts by fetching the appropriate instruction (with the same tag) from the instruction store. The instruction is executed and the result is generated, containing data and tag of the subsequent instructions which depend on it.

Pure dataflow architectures have some serious drawbacks, the major being is bad single thread performance. Other problems are: the overhead produced by token matching, as well as implementing efficient unit for matching tokens (resolved to some extent in explicit token store architecture, [12]).

Unfortunately drawbacks made it difficult to achieve direct implementation of computers based on a pure dataflow model. For this reason, possibilities of converging dataflow and control-flow models were investigated and broad spectrum of hybrids (techniques and machines based on them) were developed: threaded dataflow, course-grain dataflow, *RISC* dataflow, dataflow with complex machine operations, [13], [14].

5. Conclusion

This stream line of processor architectures is highly unlikely to die off. Other architectures are about to emerge, some similar to the previous architectures and other completely different. The advance of technology requires new and better processor architectures and exploitation of the concept of parallel computing. Overcoming the data dependencies issue has to be done in order to achieve better results in executing a program in parallel and better overall performance. Also, we need to further investigate the possibilities to narrow the gap between memory response time and processors working frequencies. However, in order to be able to go forward and toward developing completely new different processor architecture one must make thorough investigation of existing processor architectures. That was the main reason for leading this investigation, and presenting the results of it within this paper.

References

[1] Hennessy, John L., Patterson, David A. (2007). *Computer Architecture: A Quantitative Approach.*

[2] FitzRoy-Dale, Nicholas. (2005). *The VLIW and EPIC Processor Architectures.*

[3] Stallings, William. (2009). *Computer Organization and Architecture: Designing for Performance (8th ed.).* Prentice Hall.

[4] Dandamudi, Sivarama P. (2005). *Guide to RISC Processors: For Programmers and Engineers.*

[5] Oklobdzija, Vojin G. (1999). *Reduced Instruction Set Computers.*

[6] Zomaya, Albert Y. H. (Ed.). (1996). *Parallel and Distributed Computing Handbook.* McGraw-Hill. (Conte, Thomas M., “Superscalar and VLIW Processors.”)

[7] Hewlett-Packard. (1986). *Precision Architecture: The Processor.* HP Journal, August.

[8] Smotherman, Mark. (2011). *Historical Background for EPIC.*

[9] Silc, J., Robic, B., Ungerer, T. (1999). *Processor Architecture: From Dataflow to Superscalar and Beyond*.

[10] Davis, A. L., Keller, R. M. (1982). Data Flow Program Graphs. *IEEE Transactions on Computers*, February.

[11] Iannucci, R. A. (1988). Toward a Dataflow/Von Neumann Hybrid Architecture. *Proceedings of the 15th International Symposium on Computer Architecture (ISCA)*, May.

[12] Papadopoulos, G. M. (1988). Implementation of a General-Purpose Dataflow Multiprocessor. Technical Report TR-432, MIT Laboratory of Computer Science, Cambridge, August.

[13] Silc, J., Robic, B., Ungerer, T. (1998). Asynchrony in Parallel Computing: From Dataflow to Multithreading. *Parallel and Distributed Computing Practices*.

[14] Buehrer, R., Ekanadham, K. (1989). Incorporating Dataflow Ideas into Von Neumann Processors for Parallel Execution. *IEEE Transactions on Computers*, December.