On the Classification of Computer Architecture

Article · July 2003

CITATIONS
6

READS
24,549

2 authors:

Adnan Shaout
University of Michigan-Dearborn
236 PUBLICATIONS **529** CITATIONS
SEE PROFILE

Taisir Eldos
Jordan University of Science and Technology
23 PUBLICATIONS **95** CITATIONS
SEE PROFILE

On the Classification of Computer Architecture

Adnan Shaout*, and Taisir Eldos**

*University of Michigan – Dearborn, Department of Electrical and Computer Engineering, Dearborn, MI 48128
**Jordan University of Science & Technology, Department of Computer Engineering, Irbid, Jordan 22110

ABSTRACT

There are many computer architecture classification methods based on different criteria such as cost, capacity (memory size, data word length and size of the secondary storage), performance, instruction set, component base and others. The purpose of this paper is to review existing computer architecture classification methods. A brief description of their philosophy, comparative analysis and applications will be presented. New classification methods are introduced based on classification criteria such as number of storage hierarchy levels, number of addressable fields, fault tolerance, processor identity, code morphing, and reconfigurability.

1. INTRODUCTION

There are many sources for computer classifications. The Association of Computing Machinery (ACM) has one of the most detailed computer classifications; where computers are classified based on control structures, arithmetic and logic structures, memory structures, I/O and data communications and integrated circuits used.

In this paper, classifications based on various criteria will be investigated and new classification methods will be proposed. Sections 2 through 9 will describe the existing computer architectures, which are based on the instruction sets of the computer system; CISC, RISC, MISC, HISC, WISC, ZISC and VLIW, and provide analysis of advantages and disadvantages of each class. In sections 10, beside the classification of computers by their instruction sets, other criteria have been used such as cost, capacity, performance, component density, and many others. This section looks at some of these classification criteria that have been used to classify computer systems. In section 11, new classification methods are introduced based on classification criteria such as number of storage hierarchy levels, number of addressable fields, fault tolerance, processor identity, code morphing, and reconfigurability. In section 12 we conclude.

2. COMPLEX INSTRUCTION SET COMPUTER (CISC)

The earliest processor designs used dedicated hardwired logic to decode and execute each instruction. That was appropriate for simple designs with few registers, but made architectures more complex and hard to build. Developers of computer systems took another approach; they built simple logic to control the data paths between the various elements of the processor, and used microcode instruction set to control the data path logic. In those systems, the main processor has some built-in ROM, which contains groups of microcode instructions, corresponding to each machine-language instruction (a macrocode instruction).

Because instructions could be retrieved much faster from a local ROM than from main memory, designers put as many instructions as possible into microcode. Microcode implementation allows using the same programming model among different hardware configurations, beside the advantage of easily modifying the instruction set. Some machines were optimized for scientific computing, others were optimized for business computing; however, since they all shared the same instruction set, programs could be moved from one machine to another without re-compilation (but with a possible increase or decrease in performance depending on the underlying hardware.) This kind of flexibility and power made microcoding the preferred way to build new computers for some time. Assembly language programming and the low memory size of the early days promoted the CISC style, leading to common features including 2-operand format, register to memory and memory to register instructions, multiple addressing modes for memory, variable length instructions and many clock cycles per instruction. The hardware architectures had typically complex instruction-decoding logic, small number of general-purpose registers and several special-purpose registers.

3. REDUCED INSTRUCTION SET COMPUTER (RISC)

RISC processors implement small instruction sets capable of running faster than CISC instructions. Developers set the goal of achieving one cycle

execution time. This objective could only be achieved if the instruction set was pipelined. Modern RISC processors feature small instruction set, load/store architecture, fixed length coding and hardware decoding, large register set, delayed branching and one clock per instruction.

The phases in the execution path are typically: instruction fetch, decoding, operand fetch, execution, memory access and write back of the operation results. Pipelining is just the overlapped execution of the different phases of the execution path. The penalty for the disruptions is paid in the form of lost or stall pipeline cycles. The effective instruction level parallelism exploited by traditional CISC microprocessors is nearly 2.5. Typical RISC processors go beyond the classical 3 level pipeline (Fetch, decode, and execute) and use pipelines with 4, 5, or 6 levels, and up to 20 in modern processors. Deeper pipeline means more parallelism but also more coordination problems.

Statistics of real programs have shown that 15% of the instructions are branches and around 50% of the forward going branches and 90% of the backward going branches are taken. To minimize the flushing effect on the pipeline, the branching decision is made in the decoding stage. This can be done only if the branching condition tests are very simple, like for example a register compare with zero or a condition flag test. At the end of the decoding phase the processor can start fetching instructions from the new target. But in this decode cycle the next instruction after the branch has already been fetched and in order to avoid stall cycles this instruction can be executed. In this case the branch is a delayed branch. From the programmer's point of view, the branch is postponed until after the next instruction is executed. The compiler tries to schedule a useful instruction in the location after the branch, which is called the delay slot. Processors with very deep pipelines schedule up to two delay slots and fill the rest with NOPs. Another technique is to predict the target of the branch.

The whole benefit of a RISC architecture can be defeated if the compiler is not sophisticated enough to rearrange instructions in the optimal order [1]. RISC architectures try to maximize the cooperation between hardware and software. Optimizing compilers are one of the essential components of RISC systems. This act of shifting the burden of code optimization from the hardware to the compiler was one of the key advances of the RISC revolution. Since the hardware was now simpler, this meant that the software had to absorb some of the complexity by aggressively profiling the code and making judicious use of minimal instruction set and expanded register count. Thus, RISC machines devoted their limited transistor resources to provide an environment in which code could be executed as quickly as possible, trusting that the compiler had made the code compact and optimal [2].

RISC processors depend on a complex memory hierarchy in order to work at full speed. In most of them, separate data and instruction caches try to avoid contention for the system bus when a fetch is overlapped with a register load or store and that explains the necessity of proper management of a memory hierarchy. Those features are part of a common strategy to guarantee an uninterrupted pipeline flow, a high level of parallel execution of sequentially coded programs. Fixed word encoding, hardwired decoding, delayed loads, delayed branches, etc., are just ways to achieve a regular pipeline flow.

RISC Processors can be classified according to many measures that affect the performance, like the word size, datapath width, pipeline depth, cache structure as split versus common or on-chip versus off-chip, bus structure as Harvard versus Princeton, prefetch buffers and write buffers, register files as common versus private, register management as scoreboarding versus register renaming and units chaining capability.

4. RISC VERSUS CISC

Many of the techniques used in RISC processors can be implemented in CISC designs. It is possible to rewire the processor in order to execute most of the instructions in one cycle, or it is possible to use a pipelined microengine in order to speed up execution. The microengine could be a RISC kernel giving all the advantages of RISC without its disadvantages. However, RISC features can be introduced in CISC processors only at the expense of much more hardware. It is possible to program the pipeline of a CISC processor to use the dead time between the load and store of one instruction argument in memory. The microengine works in this case following a load/store model, and it dynamically reschedules the operations needed by the macrocode. This dynamical rescheduling is too expensive compared to the software scheduling used in RISC processors. Software scheduling must be done only once and then it runs without complex hardware. Dynamic scheduling needs much more hardware logic.

CISC designers move complexity from software to hardware, making tradeoffs in favor of decreased code size, at the expense of a higher cycle per seconds (CPI). While RISC designers move complexity from hardware to software, making tradeoffs in favor of a lower CPI, at the expense of increased code size. CISC processors can still be made competitive to RISC processors if the cycle time is reduced, but RISC processors are better positioned to achieve greater reductions in the clock cycle time in the long run. The cycle time is determined by the following factors: pipelining depth, amount of logic in each stage and the VLSI technology used. RISC processors can achieve larger reductions in the clock cycle time with a lower investment in design time. Reducing the clock cycle time of CISC processors is possible, but much more difficult. The question is which design philosophy will be capable of climbing

the performance ladder faster in the next few years. RISC designs appear as potentially much faster than CISC processors, but numbers show that both RISC and CISC are improving at 50-55% per year [3].

5. MINIMUM INSTRUCTION SET COMPUTER (MISC)

Increasing speed in the RISC processor creates a large bottleneck between the processor and the slower memory. To increase the memory accessing speed, it is necessary to use cache memory to buffer instruction and data streams. The cache memory brings in a whole set of problems which can complicate the system design and make the system more expensive. RISC processors are relatively inefficient in handling subroutine calls and returns. Efficient subroutine mechanism is critical to the performance of a processor in supporting high-level languages. Many RISC processors use a large register file, which is windowed to assist in subroutine calls and returns. However, the register window must be big enough to handle a large set of input, output, and local parameters. The large register window wastes the most valuable resource in the RISC processor, and slows down the computer system during context switching. The principle of simplicity was not enforced enough to realize the full benefit from this principle. MISC architectures explore simplicity to its limit, by assuming 32 instructions.

The MuP21 chip a MISC implementation with the four instructions groups [4, 5]; transfer instructions, memory instructions, and arithmetic and register instructions. So far, only 24 instructions were implemented in MISC, leaving some room for future expansion. Subtraction can be synthesized by complement and addition. OR can be synthesized by complement, AND, and XOR.

Potential applications for MuP21 include advanced video games, Video test pattern generators, CAD design system, and telephone switching system, handheld computers, high-speed communications systems, intelligent hard disk controllers, and robotic controllers.

6. HIGH-LEVEL INSTRUCTION SET COMPUTER (HISC)

A High-level Instruction Set Computer (HISC) is a general-purpose architecture proposed by Anthony Fong [6], targeted on high performance, implementation flexibility, expandability, better access control and system dependent features for nowadays demand for high computing power and multimedia applications.

HISC is a 64-bit architecture, which involves simple instructions of fixed length, entries of operand descriptors and application oriented data types. The operands of an instruction are described by Operand Descriptors, which are records, which consist of virtual addresses, data types, operand sizes, vector information, operand access codes and design and system dependent information for the operand. The data types of the operands include integer, floating-point number, BCD, character and string. The vector information includes number of elements in the vector and element spacing for vector operands.

HISC reduced the demand for conditional branching as in RISC by eliminating the looping count for operands of variable lengths and large size, as well as vectors. On the other hand, HISC will operate super-scalar on a higher level. The inter-dependency of operands will be much less, while it is much likely to operate super-scalar for two or more function units. HISC also keeps the vector information so that vector operations will be done by hardware.

HISC processors provide better encapsulation and potential performance gain. In a typical object-oriented system, an object is defined as a software bundle of persistent variables and related methods. The persistent variables record the persistent state of the object, while the methods implement the behavior of the object. The basic operations that can be applied to an object are object creation, object removal, method invocation and persistent variables invocation. Object creation is to create an instance of a class or a prototype object, in which a new set of persistent variables are created while the methods are shared among objects with the same class or prototype. Object removal is done by a process called garbage collection. Any object, which is not referenced, by any other objects in the system will be removed. Method invocation passes the control from the current executing method to the invoked method.

In a typical object-oriented system, every object is a software collection of persistent variables and methods. In many software implementations, there is an object table for an object, which defines the name, access rights and the value of the persistent variables, or methods resided in the object. This object table has a structure very similar to the Operand Descriptor Table (ODT) in HISC. The logical relationship between the HISC architecture and the object-oriented concept provides for easy mapping of an object representation to the HISC architecture.

7. WRITABLE INSTRUCTION SET COMPUTER (WISC)

Writable Instruction Set Computers (WISC) is stack-based architecture. New stack machine design based on VLSI design technology that provides additional benefits not found on previous stack machines. These new stack computers use the union of their features to achieve combination of speed, flexibility, and simplicity.

Stack machines offer processor complexity that is much lower than that of CISC machines, and overall system complexity that is lower than that of either RISC or CISC machines [7]. They do this without requiring

complicated compilers or cache control hardware for good performance. They also accomplish competitive raw performance, and superior performance in most programming environments. Their first successful application area has been in real time embedded control environments, where they outperform other system design approaches. Stack machines also show great promise in executing logic programming languages such as Prolog, functional programming languages such as Miranda and Scheme, and artificial intelligence research languages such as OPS-5 and Lisp.

The major difference between this new breed of stack machine and the older stack machines is that large, high speed dedicated stack memories are now cost effective. Previously, the stacks were kept mostly in program memory; newer stack machines maintain separate memory chips or even on-chip memory for the stacks. These stack machines provide extremely fast subroutine calling capability and superior performance for interrupt handling and task switching. When put together, these qualities create computer systems that are fast and compact.

Both hardware and software stacks have been used to support four major areas in computing requirements: expression evaluation, subroutine return address storage, dynamically allocated local variable storage, and subroutine parameter passing. Types included Expression Evaluation Stack, Return Address Stack, Local Variable Stack, Parameter Stack and Combination Stacks

New generation of stack computers is based on the rich history of stack machine design and the new opportunities offered by VLSI fabrication technology. This combination produces a unique blend of simplicity and efficiency that has in the past been lacking in computers of all kinds. The features that produce these results and distinguish these machines from conventional designs are: multiple stacks with hardware stack buffers, zero-operand stack oriented instruction sets, and the capability for fast procedure calls. These design characteristics lead to a number of features in the new stack machines. Among these features are high performance without pipelining, very simple processor logic, very low system complexity, small program size, fast program execution, low interrupt response overhead, consistent program execution speeds across all time scales, and a low cost for context switching.

Many of the designs for these stack computers have their roots in the Forth programming language [7], because Forth forms both a high level and assembly language for a stack machine that has two hardware stacks: one for expression evaluation/parameter passing, and one for return addresses. Forth language actually defines a stack based computer architecture, which is emulated by the host processor while executing Forth programs. Although some of stack machines are designed primarily to run Forth, they are quite practical to use in many applications programmed in conventional languages. Of special interest are those applications that require stack machines' special advantages: small system size, good response to external events, and efficient use of limited hardware resources.

8. ZERO INSTRUCTION SET COMPUTERS (ZISC)

ZISC is the integrated circuit based on neural network designed for applications, which usually require supercomputers [8, 9]. ZISC has a high performance, capable of operating in real time and providing a very cost-effective way to solve such problems as pattern recognition and classification.

ZISC is an expert system, which uses accumulated knowledge to recognize and classify objects or situations and take immediate decisions. ZISC does not need to be programmed because it learns by examples from samples of data. During training session it is required to enter pairs of examples and solutions and the built-in learning mechanism will accumulate the knowledge. ZISC has also generalization capability, which makes it possible to react to objects, or situations, which were not part of the learning examples.

ZISC has ability to learn while performing classification tasks, i.e. ZISC learning capability is not limited in time. It is also not limited in volume because its chips can be cascaded to create a larger system, which is very important as it ensures that the system architecture will not change when technology density increases. ZISC cascadability means several chips can be linked together to build a wider network, without additional logic and without affecting the classification or learning performances. Those features make the ZISC very easy to use, capable of solving not precisely defined problems. ZISC has the ability to separate noise from the signal, and that makes it a perfect base for signal processing.

ZISC036 [10] as an example is a fully integrated, digital implementation of the RBF-like (Radial Basis Function) model. One ZISC036 device has 36 neurons, but due to its cascadability, the total number of neurons in a network is not limited. Each neuron has a register file for prototype storage, and a distance evaluation unit to ensure a high level of parallelism. In ZISC036, 14 bits are used for distance calculations. To perform a classification, the vector components are fed in sequence and processed in parallel by every neuron. With a ZISC operating at 20MHz, 64 components can be fed and processed in 3.2 ms and the evaluation achieved within 0.5 ms after the last component has been fed. This level of performance allows more than 250,000 evaluations per second. It would require a 2,000 MIPS machine to achieve the same level of performance on a Von-Neumann processor.

The ZISC family is designed to extend the neural network beyond the boundary of a single chip without impact on performance. Some applications can also benefit from multi-layer configurations where decisions of a layer can feed the input features of a second layer. This can be achieved either by connecting several chips or by using subsets of the network operating in a time-multiplexed mode.

9. VERY LONG INSTRUCTION WORD (VLIW)

Very Large Instruction Word (VLIW) describes an instruction-set philosophy in which the compiler packs a number of simple, non-interdependent operations into the same instruction word. When fetched from cache or memory into the processor, these words are easily broken up and the operations dispatched to independent execution units. VLIW or EPIC can perhaps be best described as software based super scalar technology [11].

VLIW represents the ultimate of internal parallelism in microprocessor designs. There are two things to make a microprocessor run faster: Speed up its clock or make it perform more operations during each clock cycle. Speeding up the clock requires inventing smaller fabrication processes and adopting architectural features such as deep pipelines to keep the silicon busy. More operations per cycle imply building multiple function units on the same chip as well as executing enough instructions concurrently to keep those units busy.

The scheduling problem is the core problem of modern processor design [12]. Super scalar processors employ special hardware to uncover instruction dependencies. However, this approach goes only so far, since the scheduling hardware increases geometrically with the number of function units and eats more chip real estate. The alternative is to let the software do all the scheduling, and that is precisely what a VLIW design does. A smart compiler can examine a program, find all instructions with no dependencies, string them together in a very long batch, and execute them concurrently on an equally big array of function units. Very long instructions are typically between 256 and 1024 bits wide. Such long instructions contain many smaller fields, each of which directly encodes an operation for a particular function unit.

In hardware terms, a VLIW processor is very simple, consisting of little more than a collection of function units (adders, multipliers, branch units, etc.) connected by a bus, plus some registers and caches. This benefits semiconductor manufacturers since more silicon goes to the actual processing (rather than being spent on branch prediction, for example) and VLIW processor should run fast, as the only limit is the latency of the function units themselves. CISC implements such instructions as microprograms in a microcode ROM on the chip. Microcoding is the ultimate low-level language: synchronizing gates and buses and passing data between function units.

RISC eliminated microcoding in favor of hard-wired instructions. VLIW, on the other hand, is like taking that microcode off the chip and putting it into the compiler. The trouble is that writing microcodes is unbelievably hard. VLIW becomes viable only if a smart compiler can write it for you. This difficulty has thus far confined VLIW machines to niches such as scientific array processing and signal processing.

The compiler packs groups of independent operations into very long instruction words in a way that uses all the function units efficiently during each cycle. The compiler discovers all the data dependencies, and then determines how to resolve these dependencies, reordering the whole program by moving blocks of code around. This process differs from a superscalar CPU, which uses special hardware to determine dependencies dynamically at run time (Optimizing compilers can certainly improve the performance of a superscalar CPU, but the CPU does not depend on them). Most superscalar processors will detect dependencies and schedule parallel execution, only within basic blocks (a group of consecutive statements with no halting or branching except at the end). To find more parallelism, a VLIW machine must look for operations from different basic blocks to pack into the same instruction. Trace scheduling is a common technique to do this [12, 13, 14].

Among the issues of concern is the static nature of VLIW compiler optimizations [15]. How well will such programs perform when faced with dynamic run-time events (such as waiting for I/O) unforeseen at compile time? VLIW arose to meet the needs of scientific number crunching, but it might prove less capable on the sorts of object-oriented and event-driven programs that are more common in the PC community.

10. NEW CLASSIFICATION METHODS

Beside the classification of computers by their instruction sets, other criteria have been used such as cost, capacity, performance, component density, and many others. This section looks at some of these classification criteria that can be used to classify computer systems.

10.1. Cost, Capacity and Performance

Computers may be compared on the basis of cost, capacity (memory size, data word length, size of secondary storage) and performance (speed, throughput and number of users). They may be classed broadly as: Supercomputers, Mainframes and Minicomputers supporting variable number of users and offering several hundreds of GIPS, down to Workstations offering with several tens of GIPS. Personal desktop computers, Notebooks, Palmtops and Personal Data

Assistants offer portable performance at reasonable price.

10.2. Components Density

For many years, computers used integrated circuits to build the major internal computer components [16]. Based on the density of the components, computers can be classified as Small, Medium, Large and Very Large Scale Integration with hundreds of millions of transistors.

10.3. Word-ordering Schemes

There are two possible conventions of storing bytes of a word in the memory and computers can be classified as Big Endian or Little Endian.

10.4. Communication

Data transfer among different points in a computer system is achieved through sets of lines carrying signals called buses. Typically, we need data, address and control information to carry out such operations [11]. The way signals are multiplexed resulted in double, triple and quadruple bus styles of architecture. In multiprocessor systems, multiple bus or crossbar switched is used.

10.5. Input/Output

Data exchange with the outside is carried out through input/output ports, which are polling or interrupt driven styles.

10.6. Number of processors

.

Higher performance is achieved through the use of more than one processor. The number of processors used places a computer into one of the following categories: Single processor, modestly parallel multiprocessor, parallel multiprocessor, and massively parallel computer.

10.7. Processor Level Parallelism

Processor level parallelization is concerned with offering multiple streams of data, instructions or both. The result is Single Instruction stream Single Data stream (SISD), which are as sequential machines, Single Instruction stream Multiple Data stream (SIMD), Multiple Instruction stream Single Data stream (MISD) for fault tolerance and Multiple Instruction stream Multiple Data stream (MIMD) that are capable of executing several programs simultaneously, each of which has its own data set.

10.8. Processor-Memory Dependency

In complex computing machines, there might be a large number of processors. Those processors are typically programmed to work on one problem by partitioning tasks and data objects. Synchronization and intercommunication is a major factor that contributes to the performance. Memory partitions in such systems are either tightly coupled or lightly coupled. Coupling results in tightly coupled multiprocessor systems (Shared Memory) and lightly coupled multiprocessor systems (Distributed Memory)

10.9. Program Control Mechanism

Conventional Von Neumann computers use a program counter to sequence the execution of instructions in a program. This sequential execution style has been called control driven. Control-flow computers use shared memory to hold program instructions and data objects. Data Flow Computers perform a computation when all the operands are available, instead of being guided by a program counter. Data flow is one kind of data driven architecture, the other is demand driven. It is a technique for specifying parallel computation at a fine-grain level, usually in the form of two-dimensional graphs [17, 18, 19].

10.10. Intelligence

This class of computers has CPU and memory where information is represented in terms of structures of symbols. These computers do not have assets peculiar to human reasoning i.e. they don't have generalization capability or not able to accumulate knowledge. They operate in terms of true or false logic. The CPU fetches the instruction from the memory and performs tasks according to an algorithm (written as a program). Intelligent computers are a class of computer hardware or software, which is able to implement reasoning skills based on accumulating knowledge. Intelligent systems approaches are Expert Systems, Neural networks and Fuzzy Logic.

Expert Systems are computer program that can advise, analyze, categorize, communicate, consult, design, diagnose, explain, explore, forecast, form concepts, identify, interpret, justify, learn, manage, monitor, plan, present, retrieve, schedule, test or tutor. Such systems are good only in a particular domain; and require large amount of memory to hold all the information and a powerful computer [20, 21].

Neural networks are based upon the structure of the brain, which consists of billions of cells called neurons. They need to be trained using of one of the various training methods and learning rules [22, 23, 24, 25] .

Fuzzy logic is a superset of conventional (Boolean) logic that has been extended to handle the concept of partial truth-values between "completely true" and "completely false", i.e. thinking of membership to a set as a degree rather than a yes or no situation. Fuzzy logic is used for solving problems in real-time systems that must react to an imperfect environment of highly variable, volatile or unpredictable conditions, in particular where it is hard to formalize a solution to the problem in hand [26, 27, 28].

11. NEW CLASSIFICATION METHODS

In this section we will present several new classification methods based on different criteria.

11.1. Number of storage hierarchy levels

Computers can be classified according to the number of hierarchy levels of data storage, starting from the internal registers, cache memory, main memory, magnetic or optical disks. Cache memory may consist of one to three levels.

11.2. Number of Addressable Fields

For a two-operand arithmetic instruction, five items are needed to be specified; Operation to be performed, location of the first operand, location of the second operand, place to store the result and location of next instruction to be executed. The 4-address machine specifies all the items mentioned in addition to the operation to be performed. However, the inclusion of a program counter in the processor eliminates the need to specify the location of the next instruction, and we can assume it is always the next in line except for branch instructions. Real machines are usually classified as being in the load/store, register-memory or memory-memory classes. Modern RISC processors are of the load/store, sometimes called register-register variety. These are 1.5-address machines in which memory access instructions are limited to two instructions; load and store. The range of choices of processor-state structure and instruction type trades off flexibility in the placement of operands and results against the amount of information that must be specified by an instruction

11.3. Fault Tolerance

According to the capability of fault detection, computers can be classified into: Non-fault tolerant computers, partially fault tolerant computers with degradable performance and completely fault tolerant computers, where more than one processor performs the same task.

11.4. Processor identity

SIMD computers can be subdivided into two categories: Coprocessor architectures have two or more processors analyze the instruction stream concurrently, like an integer processor and a floating point processor. Multiple unit architectures offer a central decoding unit, which starts executing units according to the instruction that has been decoded. The decoding unit, for example, can start an integer addition in the integer unit - one cycle later; it can start the floating-point multiplication unit.

11.5. Code Morphing

A revolutionary new approach to microprocessor design uses a compact hardware engine surrounded by a software layer. The hardware component is generally a simple high-performance engine like the VLIW, with a software layer, called the Code Morphing software, surrounding this engine. Programs designed for a certain processor are dynamically translated (morphed) into the hardware engine's native instruction set [29].

An example of such technologies is the Transmeta's Crusoe. Blocks of x86 instructions are translated once, saving the resulting translation in a translation cache. The next time the (now translated) code is executed; the system skips the translation step and directly executes the existing optimized translation at full speed. This approach eliminates millions of transistors, replacing them with software. The current implementation of the Crusoe processor uses roughly one-quarter of the logic transistors required for an all-hardware design of similar performance. This results in small and more efficient hardware that is decoupled from the target instruction set architecture making future upgrades easier

11.6. Reconfigurability

Reconfigurable computing is intended to fill the gap between hardware and software, achieving potentially much higher performance than software, while maintaining a higher level of flexibility than hardware. This type of computing is based upon Field Programmable Gate Arrays (FPGAs), and is being termed Dynamic Instruction Set Computers (DISC) and Flexible Instruction Set Computers (FISC). These devices contain an array of computational elements whose functionality is determined through multiple SRAM configuration bits. These elements, known as logic blocks, are connected using a set of programmable routing resources. In this way, custom circuits can be mapped to the FPGA by computing the logic functions of the circuit within the logic blocks and using the configurable routing to connect the blocks together to form the necessary circuit [30, 31, 32, 33].

Reconfigurable systems are usually formed with a combination of reconfigurable logic and a general-purpose microprocessor. The processor performs the operations that cannot be done efficiently in the reconfigurable logic, such as loops, branches, and possibly memory accesses, while the computational cores are mapped to the reconfigurable hardware. Run-time reconfiguration allows for the acceleration of a greater portion of an application, but may introduce some overhead, which limits the amount of acceleration possible. Because configuration can take milliseconds or longer, rapid and efficient configuration is a critical issue.

12. CONCLUSION

There are numerous methods of classifying computers. Computers can be classified according to variety of criteria such as: cost, performance, memory capacity, number of users, processor level parallelism, targeted

applications, component base, intercommunication features, input/output architecture, processor-memory dependency, etc. Classes of computers are not absolute. Classification according to certain criteria may not be an easy task since the differences between different computers classes may be ambiguous. For example, designers of CISC processors implement more and more RISC features.

Classification on the instruction set basis is one of the most common classification methods. Instruction set and addressing modes has a big influence on the processor architecture. The more developed instruction set, the more complex addressing modes available, the more complex instruction decoding logic is required. Variable instruction length also contributes to the complexity of the decoding logic. Table 1 summarizes the existing computer architectures with respect to various properties.

Post-RISC features are results of the transistor counts [1, 34]. Today, transistor counts are extremely high, and they are getting even higher. The problem now is not how do to fit needed functionality on one piece of silicon, but what to do with all these transistors. In fact, designers are actively looking for things to integrate onto the die to make use of the wealth of transistor resources, and asking what to include rather than what to throw out. In conclusion, there is no advantageous architecture. There is always a trade off; simplifying hardware can put more burdens on software and vice versa. Design of computer systems or choice of commodity computer products for particular application requires consideration of wide range of aspects relating to hardware, software and their interfacing.

In this paper new classification methods were introduced based on classification criteria such as number of storage hierarchy levels, number of addressable fields, fault tolerance, processor identity, code morphing, and reconfigurability.

13. REFERENCES

[1] Bhandarkar, "RISC versus CISC: A Tale of Two Chips", Computer Architecture News, vol. 25, no. 1, March 1997

[2] R. Russell and R. Grewell. “Software Aids Pull for Real-time RISC: RISC/CISC Tradeoffs.” Electronic Engineering Times, vol. 51, September 1994.

[3] L. Gwennap, “Processor Performance Climbs Steadily.” Microprocessor Report, vol. 9, no. 1, Jan 23, 95, pp. 17-20.

[4] Charles Moore and C.H. Ting, "Minimal Instruction Set Computer", Forth Dimensions, January 1995

[5] C. Ting and C. Moore. “Mup21: a high performance MISC processor.” Forth Dimensions, January 1995.

[6] Anthony Fong. "HISC: A High-level Instruction Set Computer". 7th European Simulation Symposium, 406-410. Society for Computer Simulation, Oct 95.

[7] Phil Koopman "Stack Computers & Forth" and “MISC M17," http://www.cs.cmu.edu/~koopman

[8] Robert David, Erin Williams, Ghislain de Trémiolles, Pascal Tannhof," Description and Practical Uses of IBM ZISC036" Virtual Intelligence - Dynamic Neural Networks, June 22-26, 1998

[9] J-P LeBouquin, IBM Microelectronics ZISC, “Zero Instruction Set Computer, Preliminary Information,” Poster Show WCNN, San Diego, CA, 1994 and addendum to conference proceedings

[10] C. S. Lindsey et al, "Experience with the IBM ZISC036 Neural Network Chip", International Journal of Modern Physics, page 579.

[11] Kai Hwang, Zhiwei Xu, Scalable Parallel Computing, the McGraw-Hill, 1998

[12] J. Sanchez, A. Gonzales, “Instruction Scheduling for Clustered VLIW Architectures,” International Symposium on System Synthesis (ISSS), 2000

[13] K. Ebcioglu, E. Altman, S. Sathaye, and M. Gschwind. "Execution Based Scheduling for VLIW Architectures," Proceedings Europar '99, ILP and Uniprocessor Architecture, Sep 99.

[14] Sanjeev Banerjia. “Instruction Scheduling and Fetch Mechanisms for Clustered VLIW Processors.” PhD thesis, Dept. of Electrical and Computer Engineering, North Carolina State University, 1998.

[15] P. Faraboschi, G. Desoli, J.A. Fisher, “VLIW Architectures for DSP and Multimedia Applications - The Latest Word in Digital and Media Processing,” IEEE Signal Processing Magazine, March 1998.

[16] Vincent Heuring and Harry Jordan, Computer Systems Design & Architecture, Addison Wesley 97

[17] Arvind, L. Bic, and T. Ungerer, Evolution of Data-Flow Computers, Advanced Topics in Dataflow Computing, Prentice Hall, 1991.

[18] G. R. Gao, "A Flexible Architecture Model for Hybrid Data-Flow and Control-Flow Evaluation," Advanced Topics in Data--Flow Computing, Prentice Hall, Englewood Cliffs, NJ, 1991, pp. 327-346.

[19] M. Takesue, “A unified resource management and execution control mechanism for data flow machine.” International Annual Symposium on Computer Architecture, ACM, 1987, pp. 90-97.

[20] Barry G. Silverman, “Survey of expert critiquing systems: Practical and theoretical frontiers.” Communications of the ACM, vol. 35, no. 4, 1992, pp. 106-127.

[21] Fink, P.K., Lusth, J.C. and Duran, “A General Expert System Design for Diagnostic Problem Solving.” IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 7, no. 5, Sep. 1985.

[22] A. Konig, “Survey and Current Status of Neural Network Hardware.” Proceedings of the Int’l

Conference on Artificial Neural Networks 95, pp. 391-410.
[23] M.W. Roth, “Survey of Neural Network Technology for Automatic Target Recognition.” IEEE Transactions on Networks 1, 1990, pp. 28-43.
[24] Y. Le Cun, L.D. Jackel, B. Boser, J.S. Denker, H.P. Graf, I. Guyon, Henderson, R.E. Howard and W. Hubbard. “Handwritten digit recognition: Application of neural network chips and automatic learning.” IEEE Communications Magazine, Nov. 1989, pp. 41-46.
[25] Opitz, R. Das Lernfahrzeug, “Neural network application for autonomous mobile robots.” Advanced neural Computers, Elseviers, 1990, pp. 373-379.
[26] M.S. Yang, "A survey of fuzzy logic", Math. Computing. Modelling (18, 11), 1993. pp. 1-16.
[27] Cordon, O., Herrera, F. & Lozano, M. “On the combination of fuzzy logic and evolutionary computation: a short review and bibliography.” In W. Pedrycz (Ed.), Fuzzy Evolutionary Computation, Kluwer Academic 1997, pp. 33-56.
[28] M. Patyra, J. Grantner, K. Kirby. "Digital Fuzzy Logic Controllers: Design and Implementation." IEEE Transactions on Fuzzy Systems, vol. 4, no. 4, Nov 96, pp. 439-459.
[29] http://www.transmeta.com
[30] M. Wirthlin and B. Hutchings, “A Dynamic Instruction Set Computer,” Proceedings of IEEE Workshop on FPGAs for Custom Computing Machines, Napa, CA, April 1995, pp. 99-107.
[31] G. Goossens et al, “Integration of Medium-Throughput Signal Processing Algorithms on Flexible Instruction-Set Architectures, ”Journal of VLSI Signal Processing, vol. 9 no. 1, 95, pp. 49-65.
[32] Katherine Compton and Scott Hauck, “Configurable Computing: A Survey of Systems and Software”, Northwestern University, Technical Report, 99.
[33] J. Turley, "Soft computing reconfigures designer options" http://www.computer-design.com/Editorial/19
[34] Bhandarkar and Clark. “Performance from architecture: Comparing a RISC and a CISC with similar hardware organization.” International Conference on Architectural Support for Programming Languages and Operating Systems, CA, April 1991, pp. 310-319.

	CISC	RISC	MISC	WISC
Instruction Complexity				
Very Complex				
Complex	✓			
Moderate		✓		
Simple			✓	✓
Instruction Type				
Advanced	✓			✓
Moderate		✓		
Basic			✓	
Instruction Set Size				
Very Large (>256)	✓			
Large (64 -256)		✓		✓
Medium (32 – 64)				
Small (<32)			✓	
Memory Bandwidth				
Large	✓			
Medium		✓	✓	✓
Small				
Control Flow				
Asynchronous				✓
Synchronous	✓	✓	✓	
Data Flow				
Serial	✓		✓	
Parallel				✓
Serial/Parallel		✓		
Performance Domination				
Hardware	✓			✓
Software		✓	✓	
Hardware/Software				
Applications				
General Purpose	✓	✓	✓	
Special Purpose				✓
Programming Style				
Procedural	✓	✓	✓	✓
Non-procedural				
Execution Order				
Deterministic	✓	✓	✓	✓
Speculative				
Instruction Length				
Uniform		✓	✓	
Non-uniform	✓			✓
Clocks Per Instruction				
Large (>1)	✓			
Medium (=1)				✓
Small (<1)		✓	✓	
OS Support Needed				
Good		✓		
Fair	✓		✓	
Poor				✓

Table 1a: Summery of Existing Architectures

	HISC	ZISC	EPIC	FISC
Instruction Complexity				
Very Complex	✓	✓		
Complex			✓	
Moderate				
Simple			✓	✓
Instruction Type				
Advanced	✓	✓		
Moderate			✓	✓
Basic				
Instruction Set Size				
Very Large (>256)				
Large (64 -256)	✓		✓	✓
Medium (32 – 64)				
Small (<32)		✓		
Memory Bandwidth				
Large	✓	✓	✓	✓
Medium				
Small				
Control Flow				
Asynchronous	✓	✓		✓
Synchronous			✓	✓
Data Flow				
Serial				
Parallel		✓	✓	
Serial/Parallel	✓			✓
Performance Domination				
Hardware	✓	✓		
Software			✓	
Hardware/Software				✓
Applications				
General Purpose	✓		✓	✓
Special Purpose		✓		✓
Programming Style				
Procedural			✓	✓
Non-procedural	✓	✓		
Execution Order				
Deterministic	✓		✓	✓
Speculative		✓		
Instruction Length				
Uniform		✓	✓	✓
Non-uniform	✓			
Clocks Per Instruction				
Large (>1)		✓		✓
Medium (=1)	✓			
Small (<1)				
OS Support Needed				
Good			✓	
Fair		✓		✓
Poor	✓			

Table 1b: Summery of Existing Architectures