Analysis of Various Computer Architectures

Kulvinder Singh, Pushpendra Kumar Pateriya, Devinder Kaur, Mritunjay Rai

***Abstract:** With the tremendous acceptance of computers globally in the different fields, there comes the challenge of managing instructions, processing, and improving the efficiency of computer systems. Ever since the inception of computers from earliest ENIAC (developed at the University of Pennsylvania) to modern supercomputer Sunway TaihuLight (developed at National Supercomputing Center, China) a broad research along with great ideas in the field of computer architecture has laid its way. It has lead to the advances which has made the computers capable to perform computations at lightning speeds. For an example, the computing speed of TaihuLight exceeds 93 petaflops. It uses 40,960 individual RISC processors to reach such high speeds. In this, the architecture plays an indispensable role right from accessing memory, allocating instruction to processors for execution, and exploiting the architecture to increase the efficiency [12]. Moreover, reducing energy consumption and cost computer architecture needs to be highly specific and fast in making decisions. With the passage of time, various ideas have come up to enhance this performance index. In this paper, we describe most frequently used computer architectures with a general review and working process which can help the readers to understand the underlying principles of various computer architectures.*

***Index Terms:** Instruction Set Architecture, Load/store Architecture, Stack Architecture, Pipelining, NOP*

I. INTRODUCTION

Before the criticism of David Patterson and John L. Hennessy, computer architecture was entirely referred to the instruction set architecture. While, other features of the architecture were called as the implementation. But, the criticism shifted the view of the community to accept implementation aspect also in the formal definition of computer architecture. Hence, most general approach defines computer architecture as the architecture which comprises of the organization of different components, physical design of individual hardware elements, and instruction set architecture [1]. Although, hardware and organization paradigms have got advanced, but still general principles have not changed much in their case. On the other hand, ISA (Instruction Set Architecture) has introduced new ways of implementing ideas related to the interface between the hardware and software. Through past years various attempts have been made in this direction. In this paper, we have discussed various standard architectures, ideas, their working, historical perspective, challenges, and their significance in present scenario as well as the future scope.

Revised Manuscript Received on May 22, 2019.

Kulvinder Singh, Lovely Professional University, Punjab, India**.** Email: kulvindersingh97@gmail.com

Pushpendra Kumar Pateriya, Lovely Professional University, Punjab, India. E-mail: pushpendra.mnnit@gmail.com

Devinder Kaur, Lovely Professional University, Punjab, India. E-mail: kaurdevinder07@gmail.com

Mritunjay Rai, Lovely Professional University, Punjab, India. E-mail: raimritunjay@gmail.com

II. ISA

Instruction Set Architecture (ISA) is a mechanism which governs the interaction of hardware and software. ISA comprises of hardware implemented instructions, defined addressing modes, types, and sizes of different operands used as well as registers and input-output control [10]. Broadly speaking, an ISA is a set of operations which defines everything a machine language programmer must be familiar with to be able to program a computer [2]. Every ISA has its own set of supported operations and valid instructions format which are decoded by the control unit of the processor while execution is performed by ALU. The first ISA appears on the IBM System 360 in 1964.

III. CLASSIFICATION OF ISA

Amongst the different possible characteristics, the type of instruction set architecture supported can be the most significant feature. It fundamentally defines how the computer will be executing the instructions. Although, there exist different approaches for comparing instruction set architectures, a popular approach is based on the architectural complexity. The restriction of fixed instruction length adds simplicity, also the number of operations supported by that particular instruction set get reduced. The architecture designs which follow similar approach are called Reduced Instruction Set Computers (RISC)[4].

While, an opposite approach also exists alternative to the RISC which makes use of complex instructions. In this, size of instructions may vary adding capability of specifying sophisticated and multi step operations. These architecture designs are called Complex Instruction Set Computers (CISC) and were widely used before advent of the RISC in the 1970s. On the other hand, Very Long Instruction Word (VLIW) architecture relies on the compiler for rearranging the program in advance for exploiting instruction-level parallelism. It is contrary to other types discussed above which uses additional hardware to maximize parallel execution of instructions [5]. Based on complexity, instruction sets like Minimal Instruction Set Computer (MISC) have also been studied which have lesser complexity than RISC.

Some others are One instruction Set Computer (OISC) and Zero Instruction Set Computer (ZISC) which along with many others are theoretically important for research perspective. But, they have insignificant performance and hence not been commercialized [10]. Several other architectures also exist like Transport Triggered Architecture (TTA) which deals with data transfer buses to enhance the performance with the help of direct buses.

A. RISC

A primary set of operations, a compatible, and scalable design can be used to build a computer which can give effectively faster results with lower latency. The decoding of the instruction is faster due to the fixed set of addressing modes, while the simplicity of instruction set enhances the execution of most instructions in a single machine cycle. In addition, such a machine should have a much shorter design time [6]. As the name suggests it's architecture enables usage of lesser instruction as compared to CISC, the instructions are precise, simple, and general. Table no. I [7] can be referred to compare these two. RISC strictly follows Load/store architecture which means that only load and store instructions have the privilege to access memory. No other instruction can individually access memory by itself [6]. Since RISC design is based on load/store design, it is important to provide a compatible and fast memory methodology to work in sync with processor [5]. In a RISC system, the memory methodology comprises of effective use management of registers and caches to increase performance, and write buffers. Usually, there is also an on-chip memory management unit [8].

In a RISC design, the capability to predict the approximate time for processing of the instructions enable pipelines to work efficiently and faster [9]. The term RISC first appeared in a paper by David Patterson of the University of California, Berkeley, though similar concepts had appeared before [1]. A Similar paradigm of designing microprocessor architecture was discussed by John L. Hennessy of Stanford University in 1981, with the name of MIPS project where the emphasis was on a simplified instruction set to get high performance, which became functioning in 1983, and was able to run simple programs by 1984 [10]. Henessey explained that, The RISC architecture is simple both in the instruction set and the hardware needed to implement that instruction set. Although the MIPS instruction set has a simple hardware implementation (i.e. it requires a minimal amount of hardware control), the user level instruction set is not as straightforward, and the simplicity of the user level instruction set is secondary to the performance goals [6].

The MIPS approach was compiler-driven and emphasized on minimizing the complexity of individual instructions to execute high number of instructions at per cpu cycle , it also used pipelining with effective distribution of machine's resources between specific stages to ensure that they are 100 % utilized [10]. Later trends in architecture design showed that MIPS was the most refined among the other effective RISC architectures due to its influence on successor architectures.

B. CISC

CISC architecture were used prior to introduction of RISC, since after the advent of VLSI technology, making single processors capable of performing multiple operations became a trend, which started with namely the Motorolas 68040 and Intels 80486 processors [8]. A complex instruction set computer(CISC) design is much focused on number of operations a processor can perform, where individual instructions itself can execute several low-level operations, from common arithmetic operations to fetching and storing instructions in memory [6]. Design of CISC is based on minimum program length- maximum work on instructions methodology, unlike RISCs increasing program length to minimize work on instruction methodology, where CISC enabled processors rely on increasing more load on processor for decoding and execution of instruction whereas reducing the efforts in programming the problem [11].

The CISC architecture consists of a large number of bits in instruction formats, which specifies the type of operation, the number, type and position of the operands etc. Unlike RISC, it doesn't follow load/store architecture and fetches instruction from memory every time it needs to perform some operation with it, refer to Table 1 [6]. Code density,
i.e. the memory program takes in memory, is more than in CISC than in RISC [12].

TABLE I: RISC VS. CISC [13]

Sr no.	CISC	RISC
1.	Complex instructions with variable length	Simple instructions with fixed length
2.	Dont follow Load/Store design	Strongly follow Load/Store design
3.	Pipelining not efficient	Highly efficient with pipelining
4.	Complex addressing modes	Simple and limited ad-dressing modes
5.	Focuses on embedding large number of operation on a single processor	Focuses on execution of one instruction per cycle
6.	high number of opcode with variable size	fixed number of opcodes with fixed sizes
7.	fetches data or every time an instruction needs it	makes use of principle of locality for lesser interaction with memory
8.	limited number of registers	large number of registers
9.	Minimum program length and maximum work on instruction	Increased program length and minimum work on instruction

Since CISC consists of instructions of variable length due to use of complex addressing modes, hence. unlike RISC instructions, cannot be fetched in a single operation and the time of execution for every stage of pipeline may not be same, hence pipelining is not possible [8].

C. VLIW

The design of VLIW architecture first appeared in a paper by Josh Fisher, where he discussed aspects of ILP in RISC-like architecture, as he explained, 'To me, the part of VLIW that mattered then and matters now is the philosophy that one should get a lot of ILP in a processor without asking the hardware to do much to locate and schedule it. As with anything of this sort, theres a spectrum. I think of an implementation that expects operations to have been arranged so it can trivially put them in parallel at run-time as a good embodiment of this philosophy' [14].

VLIW architecture provides better performance which include techniques of pipelining, independent individual instruction execution and out-of-order execution.

VLIW architecture provided software designers the abstraction over hardware to extract more parallelism through optimized compilers with capability of seeking possible parallelism aspects in the software code, something which has been lacking in RISC design. It depended on hardware implementations for parallelism. VLIW brought an alternative aspect of architecture design to traditional superscalar architectures [15]. VLIW provides parallel execution of instructions on separate individual RISC- like execution units. The instruction has all the normal ISA opcodes, so each of VLIW instruction can do what normal instructions would have performed individually i.e. it provides multiple instructions in one bundle, can be executed in a parallel manner. There is no dependency checking among individual instructions inside a VLIW instruction set which helps to reduce operation overhead and designing. It can be assumed that traditional model do not consider dependency among instructions to leverage execution efficiency. All dependence checking between instructions is done during compile time by compiler only [16]. There are no data interlocks in the traditional VLIW architecture, although some capabilities were added later on.

VLIW design is solely dependent on compiler to provide the insights regarding the arrangement of instructions execution and to expose opportunities of parallelism, for instruction scheduling VLIW needs additional registers [14]. Often due to dependencies among instruction modules, the pipelines which have completed the execution have to wait for results from some other pipeline which would be processing the required instructions meanwhile in some other pipeline. Hence, the idle execution pipelines must execute No Operation (NOP) instructions until the required results from other module finishes processing.

VLIW design architecture greatly relies on an intelligent compiler which means complex and increased size of code for building the compiler. It could also be able to handle unpredicted memory conflicts that may occur on runtime and also be able to control and manage the scheduling of instructions which is really a hard job. The reason being, maintaining policies for all kind of instruction can be really complicated [17]. Few of most widely recognized CPUs with VLIW design include Elbrus 2000 and Intel's Itanium IA-64 EPIC [14].

D. TTA

Transport Triggered Architecture (TTA) design emerged during research at the Delft University of Technology [5]. TTA is similar by VLIW architecture since both exploit the ILP during compile time [5]. As discussed by Johan Janssen in his book, below example shows how TTAs can be beneficial [19].

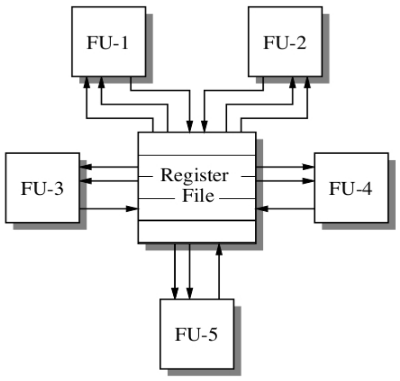

Fig. 1: Register file connectivity within a VLIW [18]

According to the example, for the VLIWs it is not even possible to keep all functional units (FUs)(where operations are processed) busy, assuming single cycle pipelined FUs. Even when all FUs are busy, the data path is not likely to be fully used due to operations that require only one source operand or do not produce a result. [18] Unlike VLIWs, TTAs do not require that each FU has its own private connection to the RF (Register File). An example of the computing core of a TTA is given in Figure 2.

An FU is connected to the RF by means of an interconnection network. It contains buses and sockets. A socket can be viewed as a gateway, which is able to pass one data item per cycle. The inputs and outputs of the FUs and RFs are connected to respectively input and outputs sockets. It is not necessary that all buses are connected to a socket. In the Fig 2, the dots indicate to which bus a socket is connected. Operations are the primary attributes which define traditional architecture, where we only need to specify the instructions and have no control over hardware implementation, hence they are often called Operation Triggered Architecture or OTAs .

The data-flow path between their processing units and the registers do not need to be programmed explicitly, they are taken care by hardware which depends on execution of operation in individual processing which the operations can be performed [20]. Although, VLIW and TTA both consists of aggregated set of instructions, but in case of VLIW, these instructions define the operations to be performed by individual processing unit. while, in TTAs the aggregation of multiple data paths or transports in a single instruction is done. Individual slot of a TTA instruction set has a control over individual bus. Data transport can be defined as the paths through which the data flows between the sender and receiver occurs, as specified by the sender-id and receiver-id given in a slot, where sender and receiver-ids defines register address [8].

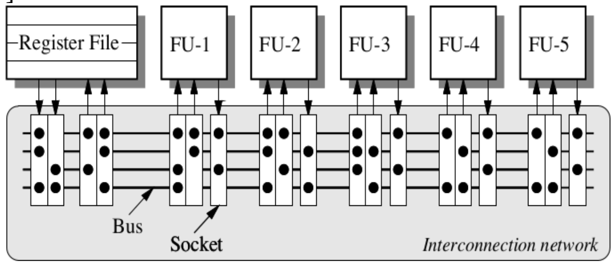

Fig. 2: Block diagram of a TTA [18]

E. MISC

MISC (minimal instruction set computer architecture) also referred as hyperscalar computer architectures, are based on reducing the number of supported instructions to a point where only essential instructions which are necessary for functioning of microprocessor are left, resulting in simple and reduced opcodes [21]. The number of instructions in MISC are much less than that of RISC, but similar to RISC, MISC tends to synthesize necessary instructions from simpler instruction whenever possible [7]. As the inventor of MISC, Michael A. Baxter highlighted necessary components of MISC architecture in his paper where architecture comprised of a central memory, an instruction buffer, a control unit, an I/O control unit, a collection of functional units, a set of register files, and a data routing circuit [21].

In MISC architecture, a collection of instructions are transferred from the central memory to the instruction buffer, and the control unit, which received these collection of instructions from the instruction buffer, which find out the set of instructions which are common for different processing units. The control unit then sends the set of common instructions one by one parallelly to relevant processing units. After every processing unit received the data from control unit , it execute the instructions and sends the results to associated register file. The transfer of data between various components is carried through the data routing circuit [21].

The only addressing mode compatible with MISC is load and store. MISC also lacks sophisticated hardware to ensure ILP and other optimizing methods for faster execution. MISC resembles RISC in other implementations [22].

IV. ORGANIZATION OF COMPUTER ARCHITECTURE

Organization refers to basic arrangement of storage, processing units and various pathways for instructions and data. The basic entities in a model of computation are Control unit, ALU, Instruction memory, data memory, and I/O devices. There are 2 computer architectures which are different in the way of accessing memories and organization of these basic modules of essential in computing : Von Neumann Architecture (also names Princeton Architecture) and Harvard Architecture.

A. Von Neumann Architecture

Von Neumann architecture is the most widely used and well accepted model which describes an organized system that allows effective execution of instructions on computer's hardware by programming the machine as per need. An important aspect behind the Von Neumann architecture is the idea of reducing load operations from memory by storing in-structions in memory with the data on which those instructions operate once in the memory and use them a many times. This was based on notion of locality of data as shown in Fig 3. Since the transfer bus is shared between data memory and program memory, any simultaneous data operation and load/store operation is not possible. This limitation is referred to as von Neumann Bottleneck which is an important problem that often restricts the performance.

Before Von Neumann proposed his model no general computing machine existed and a new computing machine was needed to be built for new problems or purposes. Earlier, the programming involved the manual rewiring of circuits of machine which was prone to errors and was a hectic job. If mistakes were made, they were difficult to detect and hard to correct. The traditional Von Neumann architecture is comprised of three individual components, a central processing unit (for controlling and processing the instructions), a memory module (for storing/ fetching instructions) , and input/output interfaces (for interacting with input/output devices) [23].
Given figure represents one of several possible ways

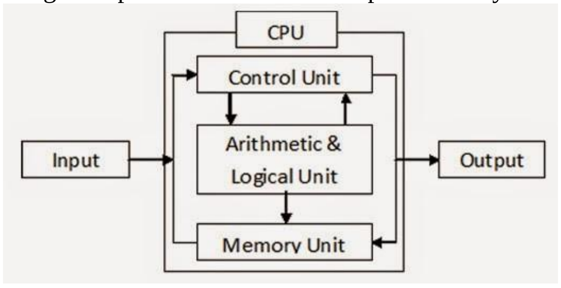

Fig. 3: Basic Operational von Neumann Architecture

of interconnecting these components. processing the instructions), a memory module (for storing/ fetching instructions) , and input/output interfaces (for interacting with input/output devices) [23].

B. Harvard Architecture

The fundamental idea which differs the Harvard architecture is that it uses two separate memory spaces for program instructions and data which results in higher bandwidth for instruction transfer, refer to Fig. 4. An important advantage of the pure Harvard architecture is that it provides simultaneous access to both program and data memory. Since fetching instructions from memory takes time, Harvard architecture faces the problem of slow processing. Modern Harvard architecture involves using caches to improve data locality while operating on data, till the data needed for processing is available in the cache, the performance is better [9].

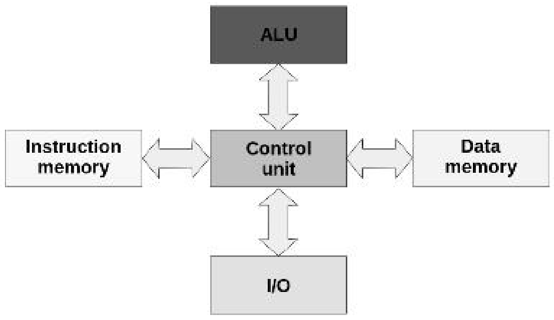

Fig. 4: Design of a Harvard Architecture

V. RESULTS AND DISCUSSIONS

In this paper, we discussed several attempts of computer architects to make execution of systems faster by setting trade-offs and making effective use of hardware by improving resource usage through supporting ideas like parallel execution and pipelining etc. with their designs. While some ideas proved to be highly significant like RISC and CISC, others faced challenges and had their own limitations like VLIWs.

*Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication*

Although, VLIW design claimed superior performance over until it turned out that the wished for compilers were basically impossible to write. The complexity of writing code for rearranging the program for exploiting parallelism turned out to be challenging and the results were still unreliable [19]. In 2017, Intel finally pulled hands out of VLIW design architecture with Itanium 9700 series to be last Intel Itanium processor commercially manufactured.
traditional architectures, but it is not proved to be commercially successful as in case of Intels Itanium. As remarked by Donald Knuth, The Itanium approach was supposed to be so terrific TTAs are also based on VLIW design so they face similar challenges, though TTAs reduced the data paths for register and functional units inter- action, hence they provide parallelism aspects with reduced power consumption, which may be one of its feature and useful in battery enabled systems. CISC however remained to be most significant commercially, as they are ubiquitous and extensively used in personal computers, laptops, even in cloud computing and workstations till date, and the credit goes to Intels efforts behind x86 family of processors, which are improved version of the earlier 8080 processors with added capabilities like backward compatibility and virtualization.

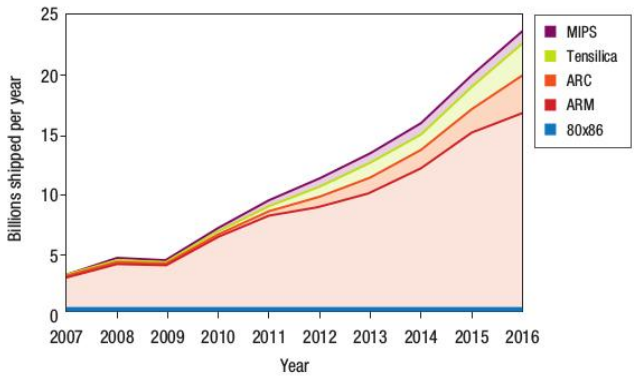

Fig. 5: Billions of chips shipped from 2007-2016 [24]

On the other hand, RISC stands out as an another commercially successful design. RISC enabled processors like ARM, MIPS, ARC, and many others are massively used in embedded systems and mobile computing devices like smartphones and tablets. Moreover, Intel's CISC enabled x86 processors are greatly influenced by RISC design and uses hardware to translate the instructions into RISC-like instructions. It internally makes it capable of augmenting any improvement made for RISC processors which is one of the main reason for its success [18].
As explained by figure 5, the sales of RISC enabled processors has increased gradually aftermath the advent of smartphones namely Apple's Iphone before which CISC based x86 processors dominated the market. RISCs makes up to 99 percent of microprocessor volume today [24]. No new CISC architecture have been given since 1985 and nor VLIW designs have been proposed after 2000.

But, RISC V, a new open source ISA is becoming popular after collaboration of major giants in the field like Google, HP, Nvidia, Oracle, Qualcomm, IBM, and many more which aims to use RISC approach to build more reliable and improved system. Therefore, RISC may bring paradigm shift to current CISC dominant PC market soon [24]-[25]. The Domain-specific architectures (DSAs), which are tailored for focusing on limited operations relevant to domain rather than capturing the whole spectrum of operations like traditional architectures do, are attaining outstanding results in performance like Googles TPU [26]. These are still being studied and improved, as they work incredibly well on domain specific tasks. A possibility of next reform can be expected from these architectures.

TABLE II: Comparison of Different Architectures based on Certain Parameters

Architecture	Instruction length	Code density	Reliance on optimizing compilers	Addressing modes	Pipelining capability	Register design	Challenges
RISC	Fixed and smal	Low	Mostly	Few	Efficient	Yes	Due to simpler instructions, size of program increases
CISC	Variable	High	Less	Large	Inefficient	No	Fails to achieve faster execu- tion due to complex instructions
VLIW	Large and Fixed	High	Always	Few	Highly Efficient (Theoretically)	Yes	Requires a highly capable compiler which is practically hard to program
TTA	Large and Fixed	High	Always	Few	Highly Efficient (Theoretically)	Yes	Faces similar challenges as VLIW due to similar design

MISC	Fixed and small	Low	Mostly	Very Few	Efficient	Yes	MISC lacks sophisticated hardware to ensure ILP and other optimizing methods for faster execution

VI. CONCLUSION

In this paper, we discussed several technological advances in computer architecture which tried to improve the performance and challenged then existing paradigms to bring out more efficiency and reliability in computing systems. We talked about a few enlightening ideas in architecture design which enabled a wide community of computer architects to think about how to get more out of their machines, making computers able to perform execution with less latency even with massive computational input. These ideas made computing systems stable even with exponentially growing challenges. Hence, making them useful everywhere from playing games to weather prediction, astronomical researches, and what not. The models which we have studied here worked well in some cases and faced challenges in some of the other. Although, these challenges may be tackled in near future for which we will have to wait, and till then new paradigms like Domain- specific architectures (DSAs) will surely find significant place in the field. Moreover, RISC still have a lot of possibilities and there is always a hope for improvement with projects like RISC V. Currently, community is collaboratively working to extract those possibilities out.

REFERENCES

1. J. L. Hennessy, “David a. patterson,” Computer Architecture: A Quantitative Approach, 1996.
2. R. P. Colwell, C. Hitchcock III, D. Jensen, H. B. Sprunt, and C. Kollar, “Instruction sets and beyond: Computers,” Complexity and Controversy. IEEE Computer, 1985.
3. G. M. Amdahl, G. A. Blaauw, and F. Brooks, “Architecture of the ibm system/360,” IBM Journal of Research and Development, vol. 8, no. 2, pp. 87–101, 1964.
4. D. A. Patterson and C. H. Sequin, “Risc i: a reduced instruction set vlsi computer,” in 25 years of the international symposia on Computer architecture (selected papers). ACM, 1998, pp. 216–230.
5. H. Corporaal, “Microprocessor architectures from vliw to tta proceedings,” 1998.
6. A. D. George, “An overview of risc vs. cisc,” in System Theory, 1990., Twenty-Second Southeastern Symposium on. IEEE, 1990, pp. 436–438.
7. Risc - https://en.wikipedia.org/wiki/risc.
8. J. Silc, B. Robic, and T. Ungerer, Processor architecture: from dataflow to superscalar and beyond. Springer Science & Business Media, 2012.
9. S. Weiss and J. E. Smith, Instruction issue logic for pipelined supercomputers. ACM, 1984, vol. 12, no. 3.
10. J. Hennessy, N. Jouppi, S. Przybylski, C. Rowen, T. Gross, F. Baskett, and J. Gill, “Mips: A microprocessor architecture,” in ACM SIGMICRO Newsletter, vol. 13, no. 4. IEEE Press, 1982, pp. 17–22.
11.] S. Simha, “The design of a custom 32-bit simd enhanced digital signal processor,” 2017.
12. E. Blem, J. Menon, T. Vijayaraghavan, and K. Sankaralingam, “Isa wars: Understanding the relevance of isa being risc or cisc to performance, power, and energy on modern architectures,” ACM Transactions on Computer Systems (TOCS), vol. 33, no. 1, p. 3, 2015.
13. V. M. Weaver and S. A. McKee, “Code density concerns for new architectures,” in Computer Design, 2009. ICCD 2009. IEEE International Conference on. IEEE, 2009, pp. 459–464.
14. J. A. Fisher, Very long instruction word architectures and the ELI-512. ACM, 1983, vol. 11, no. 3.
15. J. A. Fisher, P. Faraboschi, and C. Young, “Vliw processors: Once blue sky, now commonplace,” IEEE Solid-State Circuits Magazine, vol. 1, no. 2, 2009.
16.
17. M. H. Weiss and G. P. Fettweis, “-dynamic code width reduction for vliw instruction set architectures in digital signal processors,” in Proceedings IWISP’96. Elsevier, 1996, pp. 517–520.
18. R. Cohn, T. Gross, and M. Lam, “Architecture and compiler tradeoffs for a long instruction word processor,” ACM SIGARCH Computer Architecture News, vol. 17, no. 2, pp. 2–14, 1989.
19. D. Patterson, “50 years of computer architecture: From the mainframe cpu to the domain-specific tpu and the open risc-v instruction set,” in Solid-State Circuits Conference-(ISSCC), 2018 IEEE International. IEEE, 2018, pp. 27–31.
20. J. Janssen, “Compiler strategies for transport triggered architectures,” 2001.
21. J. Hoogerbrugge and H. Corporaal, “Transport-triggering vs. operation- triggering,” in International Conference on Compiler Construction. Springer, 1994, pp. 435–449.
22. M. A. Baxter, “”minimal instruction set computer architecture and multiple instruction issue method”,” Patent, 1993.
23. K. Scott, “Misc: the minimal instruction set computer,” in ACM SIGCSE Bulletin, vol. 34, no. 3. ACM, 2002, pp. 223–223.
24. J. von Neumann, First Draft of a Report on the EDVAC. Berlin, Heidelberg: Springer Berlin Heidelberg, 1982, pp. 383–392. [Online].
25. Available: https://doi.org/10.1007/978-3-642-61812-3 30
26. D. Patterson, “Reduced instruction set computers then and now,” Computer, vol. 50, no. 12, pp. 10–12, 2017.
27. D. Patterson and A. Waterman, “The risc-v reader: An open architecture atlas,” 2017.
28. N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa,
29. S. Bates, S. Bhatia, N. Boden, A. Borchers et al., “In-datacenter performance analysis of a tensor processing unit,” in Proceedings of the 44th Annual International Symposium on Computer Architecture. ACM, 2017, pp. 1–12.