iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 6 **Issue:** III **Month of publication:** March 2018

DOI: http://doi.org/10.22214/ijraset.2018.3337

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 6.887
Volume 6 Issue III, March 2018- Available at www.ijraset.com

A Review on Superscalar Technology with instruction level parallelism (ILP) for Faster Microprocessor

Mr. Chirag R.Patel[1] , Mr. Jignesh N. Solanki[2] , Mr. Vijaysinh K. Jadeja[3], Mr. Sohil A. Gadhiya[4] , Mr. Shaktisinh S. Parmar[5]
[1]*Computer Engineering, C.U. Shah University,*
[2, 3, 4] *Information Technology, C.U. Shah University,*
[5]*Computer Engineering, C.U. Shah University,*

Abstract: ***Over the last two decades, computer and communication technologies have literally transformed the world we live in. Superscalar technology and instruction level parallelism has emerged as the key enabling technology in modern computers for higher performance, lower costs, and sustained productivity in real-life application. A CISC or a RISC scalar processor can be improved with a superscalar or vector architecture. Scalar processor is those executing one instruction per cycle. Only one instruction is issued per cycle, and only one completion of instruction is expected from the pipeline per cycle per .In superscalar processor, multiple instructions are issued per cycle and multiple results are generated per cycle. Instruction level parallelism show maximum no of instruction executed simultaneously. Using ILP, we can achieve parallelism based on instruction. There are mainly two ways for exploiting ILP, dynamically and statically. This paper also contain information regarding future Microprocessor architecture.***

Keywords: ***Scalar; ILP; Superscalar; exploit; dynamically; statically; simultaneously.***

I. INTRODUCTION

Today's microprocessors are the powerful descendants of the von Neumann computer dating back to a memo of Burks, Goldstine, and von Neumann in 1946 [3].Superscalar processing is the latest in a long series of innovations aimed at producing ever-faster microprocessors. By exploiting instruction-level parallelism, superscalar processors are capable of executing more than one instruction in a clock cycle[1]. Because processing speeds are measured in clock cycles per second (megahertz), a superscalar processor will be faster than a scalar processor rated at the same megahertz. Superscalar processors are designed to exploit more instruction-level parallelism in user programs. Only independent instructions can be executed in parallel without causing a wait state. The amount of instruction-level parallelism varies widely depending on the type of code being executed. A superscalar architecture includes parallel execution units, which can execute instructions simultaneously. This parallel architecture was first implemented in RISC processors, which use short and simple instructions to perform calculations. Because of their superscalar capabilities, RISC processors have typically performed better than CISC processors running at the same megahertz. However, most CISC-based processors (such as the Intel Pentium) now include some RISC architecture as well, which enables them to execute instructions in parallel. Nearly all processors developed after 1998 are superscalar.

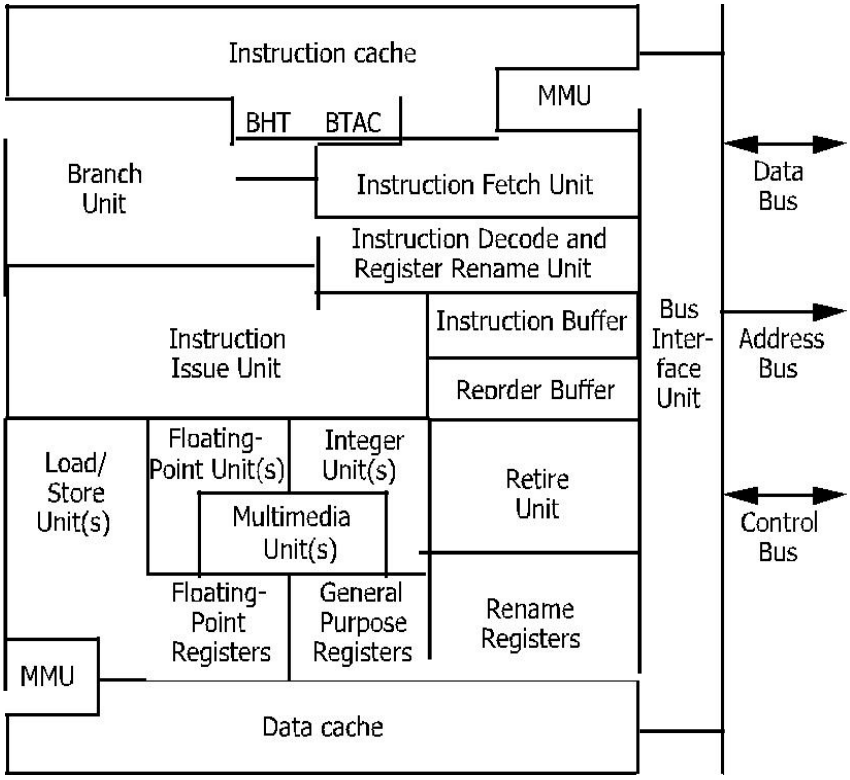


Fig. 1. Component of superscalar processor.

At least four classes of future possible developments can be distinguished; all of which continue the ongoing evolution of the von Neumann computer: Micro architectures that retain the von Neumann architecture principle (the result serialization), although instruction execution is internally performed in a highly parallel fashion. However, only instruction-level parallelism can be exploited by the contemporary microprocessors. Because instruction-level parallelism is limited for sequential threads, the exploited parallelism is enhanced by speculative parallelism. Besides the superscalar principle applied in commodity microprocessors, the super speculative, multi scalar, trace, and data scalar processor principles are all hot research topics. All these approaches belong to the same class of implementation techniques because result serialization must be preserved. A reordering of results is performed in a retirement or commitment phase in order to fulfill this requirement Processors that modestly deviate from the von Neumann architecture but allows the use of sequential von Neumann languages. Programs are compiled to the new instruction set principles. Such architectural deviations include very long instruction word (VLIW), SIMD in the case of multimedia instructions, and vector operations. Processors that optimize the throughput of a multiprogramming workload by executing multiple threads of control simultaneously. Each thread of control is a sequential thread executable on a von Neumann computer. The new processor principles are the single-chip multiprocessor and the simultaneous multithreaded processor.

Architectures that break totally with the von Neumann principle and that need to use new languages, such as dataflow with dataflow single-assignment languages, or hardware–software codesign with hardware description languages. The processor-in-memory, reconfigurable computing, and the asynchronous processor approaches also point in that direction.

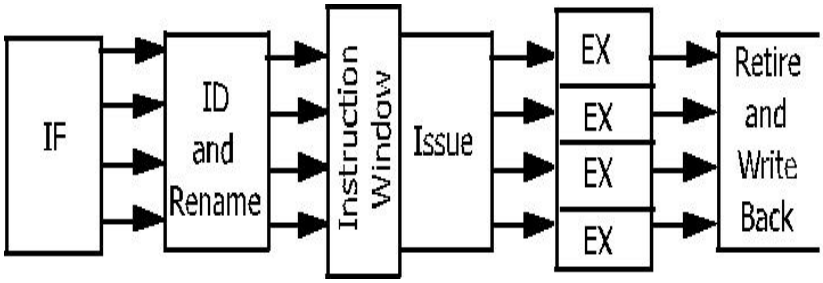


Fig. 2. Superscalar pipeline.

II. STATE-OF-THE-ART MICROPROCESSORS

Processor architecture covers the following two aspects of microprocessor design: the instruction set architecture which defines the boundary between hardware and software (often also referred to as the "architecture" of a processor), and the "microarchitecture" or internal organisation of a processor which concerns features like pipelining, superscalar techniques, primary cache organization, etc.
First we briefly discuss a common framework of the state of-the-art and future uniprocessor architectures (for more details see Ref. [29]). A state-of-the-art uniprocessor is usually a deep pipelined, superscalar RISC processor whose components include (see Fig. 1):

A. (primary) instruction cache, which holds the instructions to be fetched and executed;
B. instruction fetch unit, which fetches in each clock cycle several instructions from the instruction cache into a fetch buffer; the fetch unit is supported by a branch target address cache (BTAC) and a memory management unit (MMU);
C. instruction decode and register rename unit, where the fetched instructions are decoded, placed in the instruction buffer; rename registers (specific physical registers) are associated with the architectural registers referred to in the decoded instructions;
D. issue unit examines the waiting instructions in the instruction buffer and simultaneously assigns a number of them to the functional units;
E. reorder buffer, which stores the program order of the issued instructions;
F. functional units, which usually comprise one or two load/ store units for transferring data between the data cache and integer or floating-point registers, one or more floating-point units, one or more integer units, a multimedia unit for performing specific multimedia operations with a single instruction; reservation stations which hold instructions that wait for unavailable operands may be arranged in front of the individual functional units or in front of groups of functional units;
G. branch unit, which monitors branch history, updates the branch history table (BHT) and unrolls speculatively executed instructions in the case of mispredicted branches;
H. retire unit, which is used to assure that instructions retire (or commit)in program order even though functional units may have completed them out of program order;
I. physical registers, which can be architectural (integer, floating-point) and rename (integer, floating-point);
J. internal buffers (instruction buffer, reorder buffer, etc.);
K. data cache that only holds the data of a program;
L. bus interface unit for connecting to the environment (e.g. secondary cache when this is not already on the chip, external memory, I/O devices, etc.).

The pipelining (see Fig. 2) starts with the instruction fetch stage that fetches several instructions from the instruction cache into a fetch buffer. Typically, at least as many instructions as the maximum issue rate are fetched at once. To avoid pipeline interlocking due to jump or branch instructions, the BTAC contains the jump and branch target addresses that are used to fetch instructions from the target address. The fetch buffer decouples the fetch stage from the decode stage.
In the instruction decode stage, a number of instructions are decoded. The operand and result registers are renamed, i.e. available physical registers are assigned to the architectural registers specified in the instructions. Then the instructions are placed in an instruction buffer, often called the instruction window. Instructions in the instruction window are free from control dependences due to branch prediction, and free from name dependences due to register renaming. So, only data dependences and structural conflicts remain to be solved. The instruction window decouples the fetch and decode part, which is operated strictly in the program order, from the execution core, which executes instructions out of the program order.
The issue logic examines the waiting instructions in the instruction window and simultaneously assigns ("issues") a number of instructions to the functional units. The program order of the issued instructions is stored in the reorder buffer. Instruction issue from the instruction window can be in order (only in program order) or it can be out of order. It can be subject to either simultaneous data dependences or resource constraints, or divided into two (or more) stages, checking structural conflict in the first and data dependences in the next stage. In the case of structural conflicts first, the instructions are issued to reservation stations (buffers) in front of the functional units where the issued instructions await unavailable operands. Depending on the specific processor, reservation stations can be central to a number of functional units (e.g. Intel Pentium III), or each functional unit can have one or more dedicated reservation stations.

©IJRASET (UGC Approved Journal): All Rights are Reserved

The instructions await their operands in the reservation stations. An instruction is then *dispatched* from a reservation station to the functional unit when all operands are available, and execution starts. The dispatch sends operands to the functional unit.

When the functional unit finishes the execution of an instruction and the result is ready for forwarding and buffering, the instruction is said to complete. Instruction completion is out of program order. During completion the reservation station is freed and the state of the execution is noted in the reorder buffer. The state of the reorder buffer entry can denote an interrupt occurrence. The instruction can be completed and still be speculatively assigned, which is also monitored in the reorder buffer.

After completion, operations are committed in order. By or after commitment, the result of an instruction is made permanent in the architectural register set, usually by writing the result back from the rename register to the architectural register. Branch prediction is already a well-developed part of the state-of-the-art micro architecture design. State-of-the-art microprocessors use either a two-bit prediction scheme, a correlation-based prediction or a combination of both. In a two-bit prediction scheme two bits are assigned to each entry in the branch history table. The two bits stand for the prediction states “predict strongly taken”, “predict weakly taken”, “predict strongly not taken”, and “predict weakly not taken”. In the case of a misprediction out of the strongly state cases, the prediction direction is not changed yet. The two-bit predictor scheme uses only the recent behavior of a single branch to predict the future of that branch. Correlation between different branch instructions is not taken into account. Many integer workloads feature complex control-flows whereby the outcome of a branch is affected by the outcomes of recently executed branches. The correlation-based predictors [9] or two-level adaptive predictors [11,12] are branch predictors that additionally use the behavior of other branches to make a prediction. Some two-level adaptive branch predictors depend on the history of neighboring branches.

III. INSTRUCTION LEVEL PARALLISAM

Instruction-level parallelism (ILP) is a measure of how many of the instructions in a computer program can be executed simultaneously.

A. There are two approaches to instruction level parallelism:

1) Hardware

2) Software

Hardware level works upon dynamic parallelism where, the software level works on static parallelism. Dynamic parallelism means the processor decides at run time which instructions to execute in parallel, whereas static parallelism means the compiler decides which instructions to execute in parallel. The Pentium processor works on the dynamic sequence of parallel execution but the Itanium processor works on the static level parallelism. Consider the following program:

1. e = a + b
2. f = c + d
3. m = e * f

Operation 3 depends on the results of operations 1 and 2, so it cannot be calculated until both of them are completed. However, operations 1 and 2 do not depend on any other operation, so they can be calculated simultaneously. If we assume that each operation can be completed in one unit of time then these three instructions can be completed in two units of time, giving an ILP of 3/2. A goal of compiler and processor designers is to identify and take advantage of as much ILP as possible. Ordinary programs are typically written under a sequential execution model where instructions execute one after the other and in the order specified by the programmer. ILP allows the compiler and the processor to overlap the execution of multiple instructions or even to change the order in which instructions are executed. How much ILP exists in programs is very application specific.

B. Micro-Architectural Techniques that are Used to Exploit Ilp Include

1) Instruction pipelining where the execution of multiple instructions can be partially overlapped.

2) Superscalar execution, VLIW, and the closely related explicitly parallel instruction computing concepts, in which multiple execution units are used to execute multiple instructions in parallel.

3) Out-of-order execution where instructions execute in any order that does not violate data dependencies. Note that this technique is independent of both pipelining and superscalar. Current implementations of out-of-order execution dynamically (i.e., while the program is executing and without any help from the compiler) extract ILP from ordinary programs. An alternative is to extract this parallelism at compile time and somehow convey this information to the hardware. Due to the complexity of scaling

the out-of-order execution technique, the industry has re-examined instruction sets which explicitly encode multiple independent operations per instruction.

4) Register renaming which refers to a technique used to avoid unnecessary serialization of program operations imposed by the reuse of registers by those operations, used to enable out-of-order execution.

5) Speculative execution which allow the execution of complete instructions or parts of instructions before being certain whether this execution should take place. Branch prediction which is used to avoid stalling for control dependencies to be resolved. Branch prediction is used with speculative execution.

III. CHALLENGES AND REQUIREMENTS FOR FUTURE MICROPROCESSORS

Today's general trend in microprocessor design is driven by several architectural challenges. For example, scalable and faster busses are needed to support fast chip-to-chip communication [13, 9]. Memory latency (bottleneck) is a challenge which may be solved by a combination of technological improvements [6,11,14] and advanced memory hierarchy techniques (e.g. larger caches, more elaborate cache hierarchies, prefetching, compiler optimisations, streaming buffers, intelligent memory controllers, and bank interleaving) [4,10]. Fault-tolerant chips are needed to cope with soft errors caused by cosmic rays of gamma radiation [10]. To deal with legacy binaries, object code compatibility should be preserved [4,8], and low power consumption is of specific importance for the expanding market for mobile computers and appliances. Moreover, processor architecture must take into account the technological aspects of the hardware, such as logic design and packaging technology. The long interconnect wire delay problem requires a strict functional partitioning within the microarchitecture and a floor planning that avoids long interconnects. Designers can probably best accomplish this by dividing the microarchitecture into multiple processing elements, each no larger than today's superscalar processors. As noted in, co-ordinating these processing elements to act as a single unified processor will require an additional level of microarchitecture hierarchy for both control (distribution of instructions) and data (for communicating values among the processing elements).

IV. FUTURE MICROPROCESSOR ARCHITECTURES

In this section we survey five research directions in complex uniprocessor architectures:

A. Advanced superscalar processors, which are wide-issue superscalars that can issue up to 32 instructions per cycle (IPC).

B. Superspeculative processors, which are wide-issue superscalars that use aggressive speculation techniques.

C. Multiscalar processors, which divide a program into a collection of tasks that are distributed to a number of parallel processing elements (PEs) under the control of a single hardware sequencer.

D. Trace processors, which break up the processor into several PEs (similar to multiscalar) and the program into several traces so that the current trace is executed on one PE while the future traces are speculatively executed on other PEs.

E. Datascalar processors, which run the same sequential program redundantly across multiple processors using distributed data sets. Loads and stores are only performed locally by the processor that owns the data, but a local load broadcasts the loaded value to all other processors.

V. CONCLUSION

In this paper we have surveyed the uniprocessor alternatives such as advanced superscalar, superspeculative, multiscalar, trace, and datascalar processor, which are all examples of new micro architectures with instruction level parallelism that suitable for the next generation.

Superscalar technology and instruction level parallelism are one of the ways to get high performance processor. Developing super speculative techniques prepares the way for exceeding the classical dataflow limit imposed by data dependences. The multiscalar processor proposed the basics for a functional distribution of processing elements working simultaneously on tasks generated from sequential machine programs. The trace processor is similar to the multiscalar processor except for its use of hardware-generated dynamic traces rather than compiler-generated static tasks. The datascalar approach reduces interprocessor data traffic in favor of broadcasting.

REFERENCES

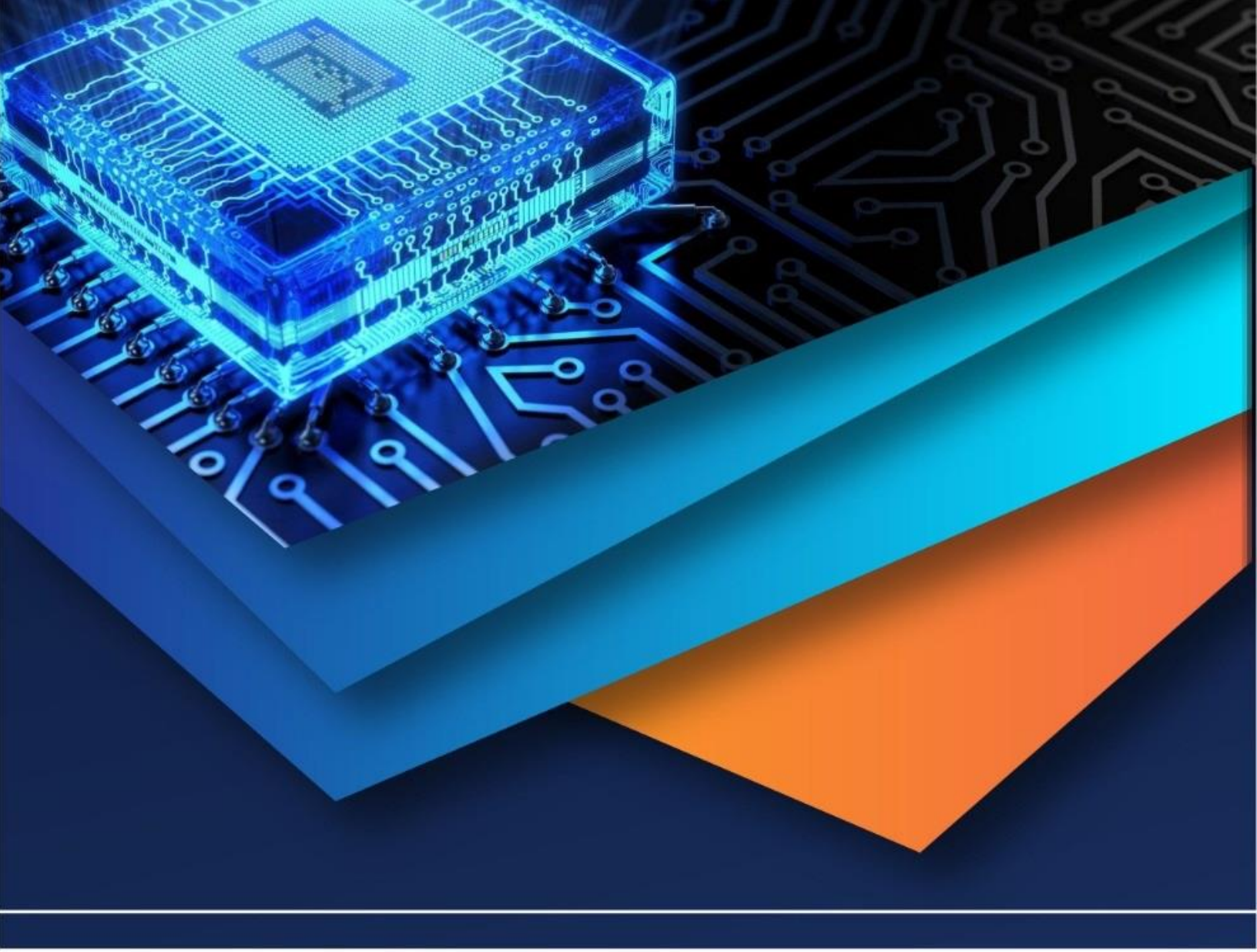
[1]. Aragon-Zavala, J. L. Cuevas-Ruiz and J. A. A. Delgado-Penin, High Altitude Platforms for Wireless Communications, West Sussex, UK: Wiley, 2008, pp 156.

[2]. J. Slic, B. Robic, T. Ungerer, Processor Architecture, Springer, Berlin,1999.

[3]. A.P. Burks, H.H. Goldstine, J.von Neumann, Preliminary discussion of the logical design of an electronic computing instrument. Report to the US Army Ordnance Department, 1946 Reprint in :W. Aspray,A.P. Burks (Eds.), Papers of John von Neumann, MIT Press, Cambridge, MA, 1987,pp. 97-146.

©IJRASET (UGC Approved Journal): All Rights are Reserved

[4]. R.P. Colwell, Maintaining a leading position, Computer 31 (1998) 45-47.
[5]. Y. Katayama, Tends in Semiconductor memories IEE Micro 17 (1997) 10-17.
[6]. R Crisp, Direct rambus technology: the new main memory standard, IEEE Micro 17 (1997) 18-28.
[7]. P. Gillingham, B. Vogley, High-performance, open-standard memory, IEEE Micro 17 (1997) 29-39.
[8]. M. Tramblay, Increasing the work, publishing the clock, Computer 31(1998) 40-41.
[9]. P.I. Rubinfield, Managing problems at high speed, Computer 31(1998) 47-48.
[10]. S.T. Pan, K. So, J.T. Rahmeh, Improving the accuracy of dynamic branch prediction using branch correlation, in: Proceedings of the ASPLOS V,Boston, MA,1992,pp.76-84.
[11]. T.-Y. Yeh, Y. N. Patt, Alternative implementation of two-level adaptive branch prediction, in: Proceedings of the ISCA 19, Gold Coast,Australia,1992,pp. 124-134.
[12]. T.-Y. Yeh, Y. N. Patt, A comparision of dynamic branch predictors that use two levels of branch history, in: Proceedings of the ISCA 20,San Diego, CA, 1993,pp.257-266.
[13]. E. Killian, Challenges, not roadblocks, Computer 31 (1998) 44-45.



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY