The Efficient Implementation of S_8 AES Algorithm

W. Ahmed, H. Mahmood, and U. Siddique

Abstract—**Information security using minimal hardware and software resources is very indispensable in mission and safety critical applications. Currently, various methodologies have been proposed in which hardware exhibits parallelism either implicitly or explicitly. In this paper, we report an enhancement in DLX processor and PicoJavaII processor instruction set for efficient implementation of modified AES algorithm. We create a custom permutation instruction, WUHPERM, in CPUSIM simulator on RISC based architecture. In addition, we implement the same instruction on Mic-1 simulator which is based on IJVM micro architecture. The results show substantial improvements in the execution time of approximately six times when the new instruction is implemented in RISC architecture and eight times for stack architecture.**

***Index Terms*— Micro architecture, Cryptography, RISC architecture, Stack architecture, Permutations algorithms.**

I. Introduction

Cryptography plays a vital role in establishing secure links in modern telecommunication networks. Information is transformed and transmitted in such a way that a third party can not extract valuable and pertinent data from a secure communication link. Many cryptographic algorithms have been proposed such as AES [5], DES [15], Twofish [16], and Serpent [17], etc.

These cryptographic algorithms use permutation operations to make the information more secure. For example, there are six different permutation operations used in DES, two permutation operations in Twofish, and two permutations in Serpent. The efficient computer implementation of permutation algorithms has always been a challenging, interesting, and attractive problem for researchers. During the past twenty years, more than twenty permutations of N elements [9]. The practical importance of permutation generation and its use in solving problems was described by Tompkins [10].

Manuscript received February 05, 2011. This work was supported in part by the Higher Education Commission (HEC) Grant No. 1-308/ILPUFU/HEC/2009-609.

W. Ahmed is a graduate student at Department of Electronics, Quaid-i-Azam University, Pakistan (e-mail: waqasat@ele.qau.edu.pk).

H. Mahmood is with the Department of Electronics, Quaid-i-Azam University, Islamabad, Pakistan (e-mail: hasan@qau.edu.pk).

U. Siddique is a graduate student at Research Center for Modeling and Simulation, National University of Sciences and Technology (NUST), Islamabad, Pakistan (e-mail: umair.siddique@rcms.nust.edu.pk).

.

TABLE I
APPROXIMATE TIME NEEDED TO COMPUTE PERMUTATION OF N
(1µ SECOND PER PERMUTATION) [9]

N	N!	Time
1	1	
2	2	
3	6	
4	24	
5	120	
6	720	
7	5040	
8	40320	
9	362880	
10	3628800	3 seconds
11	39916800	40 seconds
12	479001600	8 minutes
13	6227020800	2 hours
14	87178291200	1 day
15	1307674368000	2 weeks
16	20922789888000	8 months
17	355689428096000	10 years

If we assume that the time taken for one permutation is 1µ sec, then Table I shows the time required to complete the permutation from N=1 to N=17. For N>25; the required time is far greater than the age of the earth. Therefore, it is very important to implement the permutation operation in the most efficient manner.

We modify the DLX [1] and PicoJavaII [3] by adding a new custom permutation instruction WUHPERM in their instruction set. The performance of the new instruction is analyzed for execution time. We create and implement the new permutation instruction in CPUSIM 3.6.8 [11] and MIC-1 simulator [3] respectively.

The paper is organized as follows: Section II presents the modified AES algorithm, which is an enhanced version of the original AES algorithm and utilizes the permutation operation more intensively as compared to other algorithms. Section III presents the details of the architecture of DLX processor, the simulators used in this paper, and the new permutation instruction. Section IV presents the comparison for different implementations of the permutation instruction. We discuss the related work in Section V, and finally the conclusions are presented in Section VI.

II. Modified AES Algorithm

The modified AES algorithm is an improvement in the original AES cryptographic method presented in [4]. AES

is the first algorithm proposed by National Institute of Standards and Technology (NIST) in October 2000 and published it as FIPS 197[4]. Currently, it is known as one of the most secure and popular symmetric key algorithm [5].

S-box plays a vital role in the AES algorithm, as it is widely used in the process of encryption and provides the confusion ability. Many cryptanalysts have studied the structural properties of AES. A simple algebraic structure within AES and its S-box was presented by Gerguson et al. [6]. The most important and essential algebraic structure within AES was further analyzed in [7] and a polynomial description of AES was introduced in [8].

A new S_8 S-box is obtained by using the action of symmetric group S_8 on AES S-box [4], and these new S-boxes are used to construct 40320^{40320} secret keys [2]. The creation of the encryption keys with the permutations of the existing S-boxes results in 40320 new S-boxes, which in turn, enhances the security and makes the system more safe and reliable. As a result, the information can be transmitted more securely over an unsecure and open access channel.

The introduction of additional complexity to the existing AES algorithm increases the computation time in implementing the encryption algorithm, therefore, it is desirable to execute this algorithm in an efficient manner. We present a new instruction which facilitates the efficient execution of the modified and more complex AES method.

III. Architecture of DLX

The DLX architecture provides 32 general-purpose registers of 32 bits each which are named R_0-R_{31}. These registers have special roles. The value of register R_0 is always zero. Branch instructions to subroutines implicitly use register R_{31} to store the return address. Memory is divided into words of 32 bits and is byte addressable. The detailed data path can be seen in [1].

A. DLX microarchitecture

In this paper, we use the micro programming technique to create new instructions. The detailed examples of some important DLX instructions, used in the WUHPERM, with their corresponding micro instructions are presented in Table II. The DLX micro architecture is shown in Fig. 1.

B. Simulators

We use CPUSIM 3.6.8 [11] and MIC-1 [3] simulators to create and test new instructions. These simulators have the ability to create custom instructions. Instructions are created by implementing the microprogramming code for each individual instruction. Some important features of these simulators are described in the subsequent subsections.

C. CPUSIM 3.6.8 Simulator

The CPUSIM 3.6.8 simulator is created by Dale Skrien and is presented in [11]. This simulator has the ability to test and simulate RISC based custom designed instructions, therefore in this work, we use the CPUSIM 3.6.8 simulator to create the proposed permutation instructions for DLX microprocessor. In TABLE II, the microcode for some basic instruction of DLX processor is shown. Using these basic micro instructions, we can create new custom instructions.

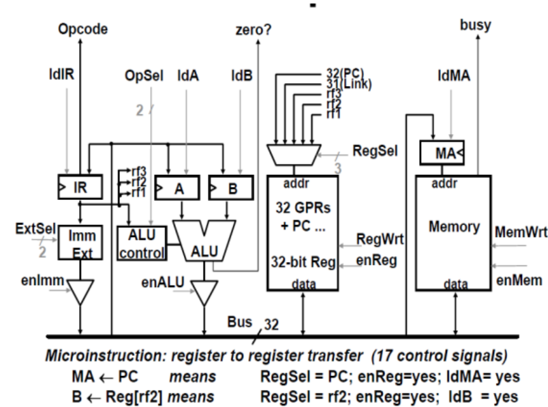

Fig 1. DLX Microarchitecture

D. MIC-1 Simulator

The MIC-1 simulator is proposed by Andrew S. Tanenbaum in his book "Structured Computer Organization" [3]. MIC-1 simulator is a JAVA based simulator which implements stack architecture and simulates the PicoJavaII custom instructions. The basic MIC-1 micro architecture is presented in [3]. It contains 32 registers, named PC, SP and MDR, etc. With the use of micro programming, we can access these dedicated internal registers. A micro program memory known as control store, which contains 512 words, is used to keep the micro program and is relatively faster than main memory.

Control store is similar to ROM and has dedicated memory address register and memory data register. The memory instruction register is called MIR, (Microinstruction Register). Its function is to hold the current micro instruction, whose bits drive the control signals that operate the data path. MIC-1 instructions and their micro instructions are presented in [3]. Using the available micro instructions we can also create custom instructions in MIC-1 simulator.

E. Permutation Instruction

Many permutation algorithms have been proposed such as Heap Method, Johnson-Trotter Method, Loopless Johnson Trotter Method, Ives Method, Alternate Ives Method, Langdon Method, and Fischer-Krause Method, etc. The heap method runs faster and is simpler than other methods as presented in [9]. A ladder diagram for heap algorithm is depicted in Fig. 2.

In this paper, we create custom permutation instructions based on this heap algorithm. The efficient implementation of the permutation operation for S_8 S-box to construct secure keys can be achieved by using these instructions as presented in [2].

The permutation operation in [2] is performed on 32-bit data. We divide these 32-bits into 8 groups of 4-bit nibbles. In this paper, we demonstrate the method used to create the permutation instructions for 4-bit nibbles and this technique can be further enhanced to 32-bit data. The heap algorithm for 4-bit data is shown in Fig. 2. It is apparent that we need four instructions to implement this algorithm. These

instructions swap these 4-bits in the order as this ladder descends. At the end, we have 24 unique set of different permutations.

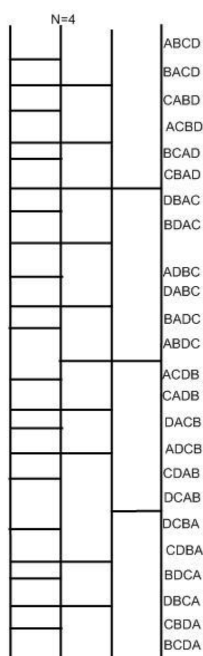

Fig 2. Ladder for Heap Algorithm

Therefore, we create four WUHPERM1, WUHPERM2, WUHPERM3 and WUHPERM4 instructions which initially swap the first and second location bit, second and third location bit, third and fourth location bit, and fourth and first location bit, respectively.

The number of instructions used to calculate the permutation of a given data is given as

$$N(I) = Log_2(N)$$

Where, $N(I)$ is the number of instruction required and N is the number of bits in data.

IV. Comparison

The assembly language program is written for CPUSIM 3.6.8 and MIC-1 simulators in order to compare the performance. It is seen that the number of microinstructions for assembly code is greater than the microinstructions required to create the custom instruction.

TABLE III and TABLE IV show the comparison for an assembly language program which performs the permutation function, and the entire program when replaced by a custom permutation instruction, which swaps any two bits. In Table III, statistics are shown for the CPUSIM simulator program. It is seen that the time consumed by assembly language program is greater when compared to the system that implements the custom instruction in its algorithm. In Table IV, analysis for MIC-1 simulator is presented. Here we can see that the performance of programs, which use custom instruction, has less execution time as compared to simple assembly language program. The performance is further enhanced if we compare these results with CPUSIM simulator output.

This comparison can also be extended to all N! unique permutations. In the case when N is increased, it is seen that performance substantially improves by using the custom instructions. It can be observed from Fig. 3 and Fig. 4, that performance improvement rate on stack based architecture (MIC-1 simulator) is greater as we increase the number of data bits. This increase is due to the redundancy in the assembly language program for this architecture, i.e., we must transfer the data into the stack in order to execute arithmetic operations.

TABLE II
DLX INSTRUCTION AND THEIR MICROCODE

DLX Instruction	Microinstruction
LD R4,100(R1)	Ir(8-15)->mar
	Main[mar]->mdr
	Mdr->Ir(5-7)
	End
SW R4,100(R1)	Ir(8-15)->mar
	Ir(5-7)->mdr
	Mdr->Main[mar]
	End
AND R1,R2,R3	Ir(8-10) ->B
	Ir(11-13) ->A
	Acc<- A & B
	Acc ->Ir(5-7)
	End
SRL R1,R2,R3	Ir(8-10) ->B
	Ir(11-13) ->A
	Acc <- A << B
	Acc->Ir(5-7)
	End

The number of microinstruction used in MIC-1 simulator is greater than the instructions required for CPUSIM 3.6.8 simulator. This is because of the inherent property of RISC architecture, which takes less execution time in performing the same amount of work than stack implementation for any processor.

V. Related Work

Various methodologies have been proposed to implement permutation operation using software and hardware. In software implementations, the permutation operation is achieved by EXTRACT and DEPOSIT instructions [12][13]. These instructions extract the bits individually by using AND mask and place the bits in a different order to produce permutation operation. In this paper, we use only one instruction to perform the tasks of EXTRACT and DEPOSIT operations. A new instruction

ISBN: 978-988-19251-4-5
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

(WUHPERM) is added to achieve the permutation operation. This reduces the complexity in the code and allows easy implementation. Also, the number of additional fetch instructions is reduced by replacing multiple instructions with a single instruction.

Hardware designs are also proposed to implement the permutation operation in an efficient way. A popular approach to achieve permutation is presented by Zhijie Shi [14]. In the hardware approach, the cost of hardware increases and the data path becomes more complex. In a software approach, the hardware changes are minimally reduced and there is no substantial increase in the cost of the microprocessor hardware.

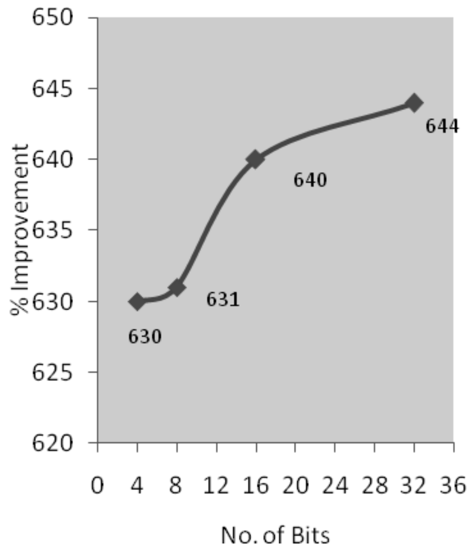

Fig 3. Percentage Performance for CPUSIM

VI. Conclusion

In this paper, a new instruction to efficiently perform permutation operation required in cryptography is presented. The new instruction implements the mathematically intensive operation used by AES algorithms and achieves enhancement in speed and performance. This enhancement to the instruction set is implemented on DLX process and PicoJavaII processor instruction set.

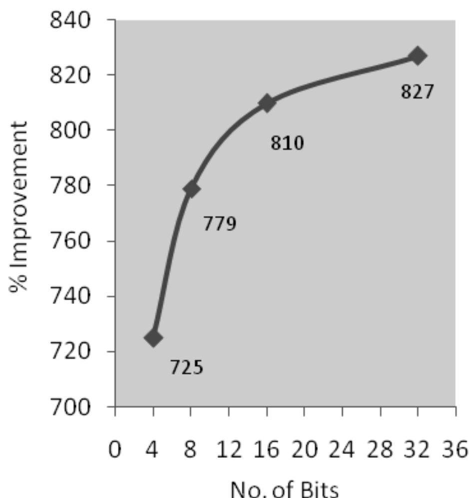

Fig 4. Percentage Performance for MIC-1

The custom permutation instruction, WUHPERM, is designed for CPUSIM simulator for RISC based architecture. In addition, we implement the same instruction on Mic-1 simulator, which is based on IJVM micro architecture. The results show a substantial improvement in the execution time of approximately six times when the new instruction is implemented in RISC architecture and eight times for stack architecture. The proposed technique is suitable for applications which require intensive permutation operations. For future studies, we propose to apply the presented techniques to information theoretic frameworks.

References

[1] J. Hennessy, D. Patterson, Computer Architecture, A Quantitative Approach. San Francisco, USA: Morgan Kaufmann publisher Inc., 1996.

[2] I. Hussain, T. Shah, and H. Mahmood, “A New Algorithm to Construct Secure Keys for AES,” *Int. J. Contemp. Math. Sciences,* vol. 5, no. 26, pp.1263-1270, 2010.

[3] A. S. Tanenubaum, *Structured Computer Organization,* 5th ed., Prentice Hall, 2005.

[4] J. Daemen and V. Rijmen, “AES proposal: Rijindael AES algorithm submission,” 1999.

[5] M. T. Tran, D. K. Bui, and A. D. Duong, “Gray S-Box for Advanced Encryption Standard,” in *Int. Conf. Computational Intelligence and Security,* CIS’08, vol. 1, pp. 253-258, Dec 13-17, 2008

[6] N. Ferguson, R. Schroeppel, and D. Whiting, “A simple algebraic representation of Rijndael,” *In Selected Areas in Cryptography,* SAC01, LNCS2259, pp. 103-111, 2001.

[7] K. Nyberg, “Differentially uniform mapping for cryptography,” *In EUROCRYPT93,* LNCS 765, pp. 386-397, 1994.

[8] J. Rosenthal, “A polynomial description of the Rijndael Advanced Encryption Standard,” *Journal of Algebra and its Applications,* vol. 2, no. 2, pp. 223-236, 2003.

[9] R. Sedwick, “Permutation generation methods,” Computing Surveys, vol. 9, no. 2, Jun. 1977.

[10] C. Tompkin, “Machine attack on problems whose variable are permutations,” in *proc. symposium in Appl. Math. Numerical analysis,* McGraw Hill, Inc., N. Y., vol. 6, pp. 195-211, 1956.

[11] D. Skrien, “CPU Sim 3.1: A Tool for Simulating Computer Architectures for CS3 classes,” *ACM Journal of Educational Resources in Computing,* vol. 1, no. 4, pp. 46-59, Dec. 2001.

[12] R. Lee, “Precision Architecture,” *IEEE Computer,* vol. 22, no. 1, pp. 78-91, Jan. 1989.

[13] R. Lee, M. Mahon, and D. Morris, “Path Length Reduction Features in the PA-RISC Architecture,” in *Proc. of IEEE Compcon,* San Francisco, California, pp. 129-135, Feb. 24-28, 1992.

[14] Z. Shi and R. B. Lee, “Bit Permutation Instructions for Accelerating Software Cryptography,” in *Proc. of the IEEE Int. Conf. on Application-specific Systems, Architectures and Processors,* Boston, Massachusetts, USA, pp. 138-148, Jul.10-12, 2000.

[15] B. Schneier, *Applied Cryptography*, 2nd ed., John Wiley & Sons, Inc., 1996.

[16] B. Schneier and J. Kelsey. Twofish: A 128-bit block cipher. ch. 4. Available: http://www.schneier.com/twofish.html.

[17] B. Smith, R. Anderson, E. Biham, and L. Knudsen. Serpent: A Proposal for the Advanced Encryption Standard. Available: http://www.cl.cam.ac.uk/ftp/users/rja14/serpent.pdf.

[18] K. Asanovic. DLX Microprogramming Slides. *MIT Laboratory of computer science,* Available:http://dspace.mit.edu/bitstream/handle/1721.1/35849/6-823Spring-2002/NR/rdonlyres/Electrical-Engineering-and-Computer-Science/6-823Computer-System-ArchitectureSpring2002/B1A470D6-9272-44BE-8E8D-B77FA84A7745/0/lecture04.pdf.

ISBN: 978-988-19251-4-5
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

TABLE III
MICROINSTRUCTION ANALYSIS OF CPUSIM (A.I= ASSAMBLY INSTRUCTION, M.I = MICROINSTRUCTION, F.I FETCHED INSTRUCTION)

Assembly Language Program						**Custom Instruction**				
Bits	**A.I**	**M.I**	**F.I** (I x 4)	**Total** M.I	**Time (T)** Total M.I x 4	**M.I**	F.I (I x4)	**Total** M.I	**Time (T)** (Total M.I x 4)	**%Performance**
4	33	158	132	290	1160	42	4	46	184	630%
8	58	298	232	530	2120	80	4	84	336	631%
16	112	575	448	1023	4096	156	4	160	640	640%
32	220	1131	880	2011	8044	308	4	312	1248	644%

TABLE IV
MICROINSTRUCTION ANALYSIS OF MIC-1 (A.I= ASSAMBLY INSTRUCTION, M.I = MICROINSTRUCTION, F.I FETCHED INSTRUCTION)

Assembly Language Program						**Custom Instruction**				
Bits	**A.I**	**M.I**	**F.I** (I x 4)	**Total** M.I	**Time (T)** Total M.I x 4	**M.I**	**F.I** (Ix4)	**Total** M.I	**Time (T)** (Total M.I x 4)	**% Performance**
4	90	367	90	457	1828	62	1	63	252	752%
8	174	706	174	880	3520	112	1	113	452	779%
16	342	1384	342	1726	6904	212	1	213	852	810%
32	678	2740	678	3418	1372	412	1	413	1652	827%

ISBN: 978-988-19251-4-5
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)