CO227::Computer Engineering Project – Report

Hardware Based Virus Scanning Acceleration

Dassanayake D.K.L.D.C (E/13/052)
Keerthirathna D.G.K.P (E/13/186)
Gunasena W.S (E/13/120)

http://www.ce.pdn.ac.lk
Department of Computer Engineering
Faculty of Engineering
University of Peradeniya
Peradeniya 20200
Sri Lanka

Abstract

Anti-malware security is an essential feature in modern day because even security of a whole nation can be exploited by them. Although, almost all the time scanning for these malicious software cost a great deal in power and time when either running an Anti-Malware program or scanning for threats. In this report, acceleration of the threat detection process by implementation of a dedicated processor component will be described. The product can be embedded to the processor and expense the dedication in running real time Anti-Malware program or threat scanning process. The fourth chapter describes about the implementation of the product in detailed and achievements made in progress. Fifth chapter gives a brief description about the overall concept and future work.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

In present, we are living in a world of information that constantly demand for the security of data and information. Searching and identifying is has become a common but crucial process in order to prevent the harmful threats from executing. Earlier, methods like signature based virus detection process which generated a unique “hash number” for each and every executable file used for the virus detection process. Previously defined viruses and other malicious software’s unique hash numbers are stored in the database of the Anti-Malware software virus database. If the scanned executable file’s hash number is matched to any of these numbers, then it was considered as a threat and prevented further harm from taking relevant steps of protection. But advanced viruses avoid this detection and hence more advanced virus detection methods invented such as Heuristic and Sandboxing methods.

The signature based virus detection method has different hash number generation algorithm such as CRC and MD5. These algorithms execute series of calculations in order to make every hash number is unique. The number of instructions executed, power and time consumption and processor, RAM and other resource usage is high when running an Anti-Malware program. Speeding up the process, reducing the complexity of the process and reducing the number of instructions executed are some key improvements that can be made for the process.

This Hardware Bases acceleration method basically focuses on the reduction of the number of instructions executed when generating a Hash number for an executable function processing from a “Single Cycle MIPS” processor and analyse the improvements which can be carried forward into the application of the modern day advanced economic level processors.

This report served as a starting point and a prototype for the application of the method and concept in larger scale. It describes the implementation on basic processor to reduce the executed instruction count and develop a dedicated hardware component on processor. Not only that but the usage of this implemented product shows statistics of considerable improvement.

CHAPTER 2

RELATED WORK

"Virus Scan Based on Hardware-Acceleration" by Xuezeng Pan, Bajoun Zhang, Jiebing Wang; published on Second International Multi-Symposiums on Computer and Computational Sciences (IMSCCS 2007, 13 – 15 August, 2007). [Online] (Available; http://ieeexplore.ieee.org/document/4392624/)

Abstract: A hardware-accelerated based computer virus scan system is proposed in this study. To satisfy the high-speed network, three strategies are selected. First, a special accelerator card is used to accelerate the speed of virus scan. Second, a more effective protocol is designed for the communication between the client and the virus scan server. Third one is to enhance the capability of concurrent processing of the system. Experimental results demonstrate that the virus scan system is of large throughput and high accuracy.

CHAPTER 3

OBJECTIVES AND SCOPE

The objective of this product was to implement and upgrade a processor so the final product will accelerate the virus scanning process. The ultimate goal will be a dedicated processor component for economic computers which will significantly accelerate the processes of Anti-Malware programs.

The scope of this product was limited to a prototype where an already developed Single Cycle, MIPS 1 processor with clock rate of 50 Hz. Furthermore, the implementation was done only using Signature based virus detection method and for the Hash number generation, a MD5 hashing algorithm was chosen. In this report the implementation of the above processor to calculate the hash number faster, the modification of MIPS instructions is described.

The goal of this product was to reduce the frequently and iteratively executed MIPS instructions in the Hashing algorithm and hence reduce the executed number of instructions in the processor. Therefore, due to the single cycle architecture of the processor, the overall time and power consumption has reduced.

For the implementation, the scope was R type instructions and other instructions that were not implemented, but possible with the usage of 32bit Instruction Set Architecture. For this prototype implementation, four distinct modified instructions were included and they are described below.

- add3 - executes the addition of three register values and store in destination register
- andor - executes and operation on two register values and executes or operation with the summery and third register value, then stores in the destination register.
- modulus - executes modulus operation using two given register values and stores in destination register.
- rol - executes rotate left on a given register value by another register value and stores the result in destination register.

CHAPTER 4

METHODOLOGY

4.1.1 OVERVIEW OF METHODOLOGY

A general description of the implementation is shown below.

The product has separate function codes for the implemented instructions. The implementation strategy is briefly described below.

- Identification of the implementable MIPS instructions.
- Conceptual design of the implementation of instructions.
- Estimation of optimization, pros and cons.
- Implementation of modified instructions.
- Testing and verification of implemented instructions.
- Testing, finalization of the product and conclustions.

4.1.2 DETAILED DESCRIPTION OF METHODOLOGY

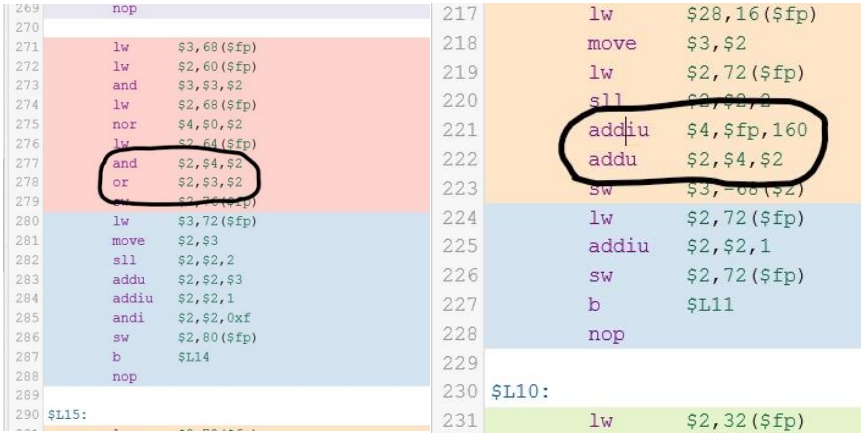
Initially a Single Cycle MIPS 1 32bit architecture processor was selected for the implementation and downloaded from the GITHub, a free online and offline code base. Then MD5 hashing algorithm with 128bit encryption key was selected for the implementation as the algorithm for Signature Based virus detection method.

Identified blocks of MD5 hashing algorithm are shown below. For this process, free online C to MIPS 5.4 converter server was used known as Compiler Explorer.

```
// break chunk into sixteen 32-bit words w[j], 0 ≤ j ≤ 15
for (i = 0; i < 16; i++)
    w[i] = to_int32(msg + offset + i * 4);

// Initialize hash value for this chunk:
a = h0;
b = h1;
c = h2;
d = h3;

// Main loop:
for (i = 0; i < 64; i++) {
    if (i < 16) {
        f = (b & c) | ((~b) & d);
        g = i;
    } else if (i < 32) {
        f = (d & b) | ((~d) & c);
        g = (5 * i + 1) % 16;
    } else if (i < 48) {
        f = b ^ c ^ d;
        g = (3 * i + 5) % 16;
    } else {
        f = c ^ (b | (~d));
        g = (7 * i) % 16;
    }

    temp = d;
    d = c;
    c = b;
    b = b + LEFTROTATE((a + f + k[i] + w[g]), r[i]);
    a = temp;
```

Figure 1.1 – Identified implementable blocks of code

The implementable instructions of relevant code blocks of MIPS 5.4 instructions are shown below.



```
269         nop
270
271         lw      $3,68($fp)
272         lw      $2,60($fp)
273         and     $3,$3,$2
274         lw      $2,68($fp)
275         nor     $4,$0,$2
276         lw      $2,64($fp)
277         and     $2,$4,$2
278         or      $2,$3,$2
279         sw      $2,76($fp)
280         lw      $3,72($fp)
281         move    $2,$3
282         sll     $2,$2,2
283         addu    $2,$2,$3
284         addiu   $2,$2,1
285         andi    $2,$2,0xf
286         sw      $2,80($fp)
287         b       $L14
288         nop
289
290 $L15:
```

```
217         lw      $28,16($fp)
218         move    $3,$2
219         lw      $2,72($fp)
220         sll     $2,$2,2
221         addiu   $4,$fp,160
222         addu    $2,$4,$2
223         sw      $3,-68($2)
224         lw      $2,72($fp)
225         addiu   $2,$2,1
226         sw      $2,72($fp)
227         b       $L11
228         nop
229
230 $L10:
231         lw      $2,32($fp)
```

Figure 1.2 – Respective MIPS 5.4 instructions

The selected instructions which are used for the implementation are as follows.

Name	**Operation**	**Calculation**	**Function Code**
add3	ADD three register values and store in destination	RD = RS + RT + R4	101111
andor	AND two register values and OR third with the result, store in destination	RD = (RS & RT) | R4	110000
modulus	MODULUS on a register value with another register value, store in destination	RD = RS % RT	110010
rol	ROTATE LEFT a register value from a given register value	RD = [RS << RT] | [RS >> (32 - RT)]	110001

Table 2.1 – Implemented instructions and operations

For these instructions, the architecture was changed in order to insert fourth register value to be used in the calculations. The standard R type instruction architecture is shown below.

31:26	25:21	20:16	15:11	10:06	05:00
000000	**01001**	**01010**	**01011**	**00000**	**100000**
opcode	RS	RT	RD	Shift amount	function

Figure 1.3 – Standard R type MIPS instruction architecture

The above architecture was changed and 5bits used for the shift amount was used to insert fourth register which is used for calculations. This implementation was used for both **add3** and **andor** modified instructions. The modified instruction architecture is shown below.

31:26	25:21	20:16	15:11	10:06	05:00
000000	**01001**	**01010**	**01011**	**00000**	**100000**
opcode	RS	RT	RD	Added Register	function

Figure 1.4 – Modified R type MIPS instruction architecture

For **modulus** and **rol** instructions, same standard R type MIPS instruction architecture was used because they only used two register values for calculations.

In order to use the implemented operations, same **opcode** as the standard R type MIPS instructions was used in modified instructions but the function code was uniquely defined for them. This process required additional register fetching needed to be done. Therefore, the register module of the MIPS processor and datapath for the ALU model was modified accordingly. A block representation of the modified register module is shown below.

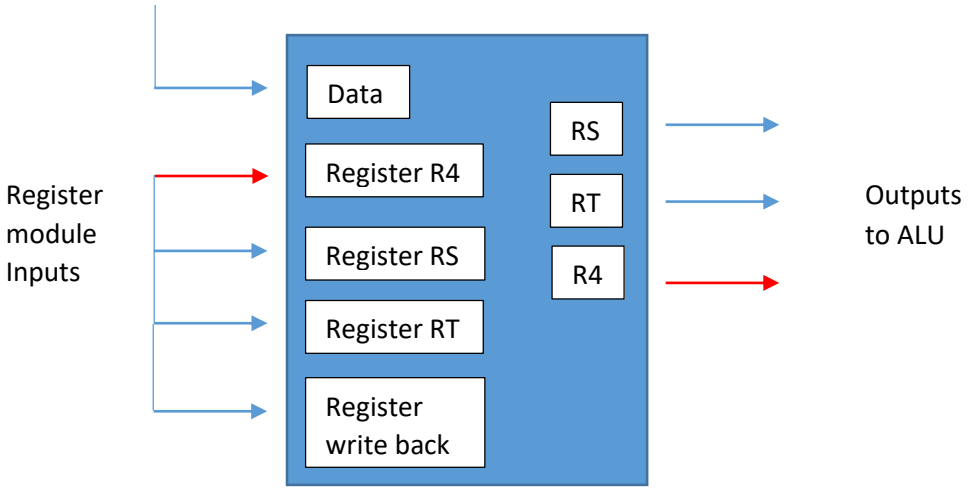


Figure 1.5 – Block diagram of modified register module

To fetch the additionally inserted register only when needed, ALU model controller was modified in addition to the modifications done to the ALU. This modification allowed the processor control and ALU control to decide when to take in the input of additional R4 register value for calculations.

Testing of the project was solely carried on with the use of Altera Cyclone IV E FPGA board, a product of Altera Company, and for compilation and FPGA board programming, Altera Quartus II v10.0 Web Edition, FPGA support tool provided with the board by Altera Company was used.

In order to display the results generated from executed instructions, separate Seven Segment Display decoder module was created. With the decoder module, up to 64bit number was able to display as upper and lower bits in hexadecimal numbers separately. A sample output in SSD shown below.

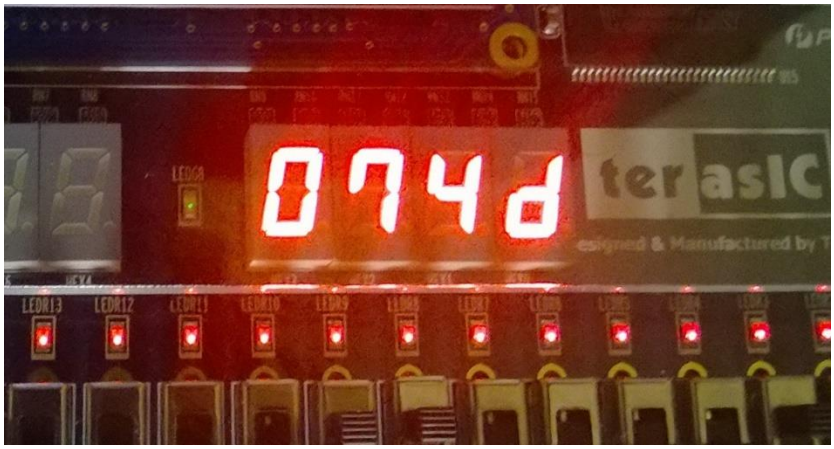


Figure 1.6 – Sample display of output

Finally, the implemented processor was tested and verified with the use of FPGA board and support tools. To fetch instructions to processor, a .mif file was used which contained all the instructions to be executed in the process. To create this file, Mars 4.5 an free Java implemented software which can generate hexadecimal instructions for given MIPS instructions, was used. Testing was done for s small block of C code which contained all the implemented instructions.

The respective MIPS code and converted hexadecimal instruction file (.mif) is displayed below.

Modified MIPS Code	Hexadecimal Instruction Input
li $t1,32 li $t2,3 li $t3,654 add3 $t4,$t1,$t2,$t3 andor $t5,$t1,$t2,$t3 modulus $t6,$t1,$t2 rol $t7,$t1,$t2	WIDTH = 32; DEPTH = 7; CONTENT BEGIN 0000:24090020; 0001:240a0003; 0002:240b028e; 0003:012A62EF; 0004:012A6AF0; 0005:012A7032; 0006:012A7831; END;

Figure 1.7 – Modified MIPS code and Hexadecimal instructions

4.2 ACHIEVEMENTS

As for the main objective of this product, the main achievement was the reduction of executed number of instructions. Processor was single cycle and hence the improvement of the time is directly depending on the number of instructions executed. Given below is a brief description for optimization achieved when executing the MD5 hashing algorithm on a single byte character instead of an executable file. The whole process was not taken into consideration due to the complexity. Important instructions taken from the main loop of the hashing algorithm is described below.

```
// Main loop:
for (i = 0; i < 64; i++) {

    if (i < 16) {
        f = (b & c) | ((~b) & d);
        g = i;
    } else if (i < 32) {
        f = (d & b) | ((~d) & c);
        g = (5 * i + 1) % 16;
    } else if (i < 48) {
        f = b ^ c ^ d;
        g = (3 * i + 5) % 16;
    } else {
        f = c ^ (b | (~d));
        g = (7 * i) % 16;
    }

    temp = d;
    d = c;
    c = b;
    b = b + LEFTROTATE((a + f + k[i] + w[g]), r[i]);
    a = temp;

}
```

Figure 1.8 – Main loop of MD5 hashing algorithm

	Before Implementation		After Implementation	
Instruction	Times Executed * Number of instructions	Total Number of Instructions Executed	Times Executed * Number of instructions	Total Number of Instructions Executed
Consecutive AND & OR	32 * 8	256	32 * 5	160
MODULUS operation	64 * 6	384	64 * 4	256
Rotate Left operation	64 * 10	640	64 * 4	256
Three consecutive ADD operations	64 * 6	384	64 * 5	320
Total number of instructions executed for a single byte character in main loop		**1664**		**992**
Optimization Gained				
Reduction of instructions gained				**672**
Percentage of instruction reduction				**40.385%**
Speed improvement gained (Clock cycle time * Number of instructions reduced)		[(1 / 50) * 672] s		**13.44 s**
Percentage of time saved				**40.385%**

Table 2.2 – Optimization achieved

As shown above achievements, time optimization was obtained by directly reducing the total number of instructions executed. The time and power optimization was **40%** as for the unimplemented product.

CHAPTER 5

CONCLUSION

The product displayed promising results in achieveing the objective as directly reduced the number of instructions executed and thus optimizing and saving time and power of execution. As for a **single cycle** processor, the product's benefits may applicable directly. But the economic level processors in the market are muti cycled, pipelined and high performing (high clock rates as 4.0 GHz) and therefore this methid is not directly applicable. The results of above analyze shows some important facts that this method, if implemented, can be applicable for the modern day economic standard processors.

Above analyze was executed only for a single byte character. But this Signature based virus detection method and MD5 hashing algorithm actually process for executable files with capacity range of 10 – 1000 kilo bytes. Therefore, if processed above calculation on above executable file in prototype processor, hundrands of hours can be saved using the implementation. But modern processors having colck rates of around 4.0 GHz and having multiple cores, the optimization may not be visible enough to make a direct impact. But as for future work, this optimization method can be used on muti cycled, pipelined processors and also on comparing process of Signature based virus detection method.

As for few other future work, developing a separate amd dedicated processor component to accelerate the processes of Anti-Malware programs shows promising resuts. Still there are several other facts that decides the speed of the vius scanning.
This implementation method could be used for other virus detection methods and alforithms, in order to achieve the ultimate goal of developing a separate hardware component to accelerate virus scanning processes.

REFERENCES

[1] E. Al Doud, Iqbal H. Jebril, Belal Zaqaibeh. (2008, September). “Computer Virus Strategies and Detection Methods” (Int. J. Open Problems Compt. Math., Vol 1, No 2) [Online].
Available:http://www.emis.ams.org/journals/IJOPCM/files/IJOPCM(vol.1.2.3.S.08).pdf

[2] Kazunori Sato, “CPU 32” [Online]. Available: https://github.com/kazunori279/CPU32.

[3] Xuezeng Pan, Bajoun Zhang, Jiebing Wang, “Virus Scan System Based on Hardware - Acceleration,” IEEE INSPEC 9879781, [Online] Availabe:
https://doi.org/10.1109/IMSCCS.2007.21.