International Journal of Advanced Computer Science, Vol. 1, No. 4, Pp. 152-162, Oct. 2011.

An Analysis of x86-64 Instruction Set for Optimization of System Softwares

Amr Hussam Ibrahim, Mohamed Bakr Abdelhalim, Hanadi Hussein, & Ahmed Fahmy

Manuscript
Received:
28, Aug., 2011
Revised:
13, Sep., 2011
Accepted:
25, Sep., 2011
Published:
30, Oct., 2011

Keywords
Intel x86-64 instruction set, instruction set analysis, 32-bit applications analysis, 64-bit applications analysis, Windows7

***Abstract*— A thorough analysis of an instruction set provides a fundamental step for development of highly optimized system software, applications and intermediate language. As one of the most abundantly used processor families, x86-64 instruction set is utilized in the development of a wide variety of applications. This paper provides an up-to-date analysis of the x86-64 instruction set on Windows 7 operating system for both 32-bit and 64-bit applications. To be able to study different types of applications, a set of executables and libraries from different software categories were chosen for the analysis. The results include the frequencies of the executed instructions, the average instruction lengths, the number of instructions for each number of operands and registers usage. The results show that the average instruction length is about 2 to 3 bytes. The mov, push and add instructions are the most abundant.**

1. Introduction

Developing an intermediate language such as Java Bytecode and Microsoft's CIL for a cross platform language requires an analysis of the instruction frequencies of the different platforms. By examining the instruction frequencies, small sized opcodes can be assigned to the most frequently used instructions. This enables the design of an intermediate language that produces small sized bytecode files and has minimum memory consumption. This paper analyzes the x86-64 instruction set not only because of its large market volume but also because it is one of the most complex architectures available.

To the best of our knowledge, very few papers have applied analysis on the x86 instruction set. Although Huang and Peng [1] have done an instruction frequency analysis, it was intended for superscalar implementation and it was implemented on windows 95/DOS operating systems. The other paper by Peng [2] was aimed to examine the implications of the instruction frequencies on the microarchitecture design and the study was also made on windows 95/DOS operating systems. An instruction frequency analysis has also been done by Rico [3] but only on MS-DOS with focus on integer number processing applications using 16-bit system.

This paper provides an up-to-date analysis of the x86-64 instruction set on Windows 7 operating system. The analysis includes both 32-bit and 64-bit binary files from different software categories. In addition to producing results of the instruction frequencies, we also study the addressing modes registers' utilization, instruction lengths and the number of instructions vs the number of operands of an instruction are studied for each category. We also divided the binary files according to their bittedness (32-bit and 64-bit). The registers' utilization is also analyzed for both the 32-bit and 64-bit division in addition to the other criteria. Finally, the summarized results are then presented to show the overall statistical information of all the categories combined.

This paper is organized as follows: in Section 2, the setup of the analysis is presented. Section 3 provides a brief overview of the x86-64 architecture, followed by the analysis metrics used in Section 4. Then, the results of each software category are presented along with the overall summary of all categories' results are shown in Section 5. Finally, Section 6 draws some important conclusions and guidelines for future work.

2. Analysis Setup

The analysis was performed on a 64-bit PC running Windows 7 (x64), which allowed for studying both 32-bit and 64-bit applications. A Portable Executable (PE) reader has been developed [4] to extract the binary code from windows executables and dynamic library files and determine the bittedness of the application. The extracted code was then disassembled using a free utility library called udis86 [5]. A statistics generator has been developed that consumed the disassembled instructions generated by udis86 to produce the results (see Fig. 1).

The analyzed applications were picked to cover the broad spectrum of application categories. The applications include both executable and library files. Table 1 lists the applications used in the analysis with a description of each.

Each instruction in the disassembled code is checked for the type of instruction, operands type, instruction length, addressing mode used and number of operands. The statistics are then updated with the results of that instruction.

Amr Hussam Ibrahim., Mohamed Bakr Abdelhalim, Hanadi Hussein, & Ahmed Fahmy are with the department of Computer Engineering in the Arab Academy of Science And Technology.

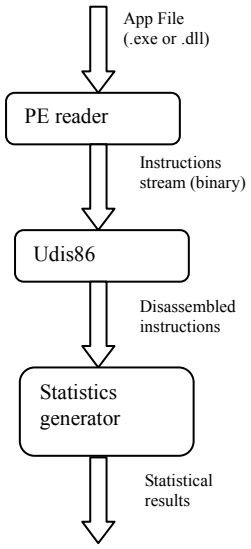


Fig. 1 Process used for statistics generation

TABLE 1

APPLICATIONS USED IN THE ANALYSIS

Application	Description	Type
Acrobat.exe	The application used to read the popular PDF format.	32-bit executable
ChkDsk.exe	Windows' disk checking utility.	32-bit executable
Explorer.exe	Windows' file system browsing utility.	64-bit executable
IExplorer.exe	The 9th version of Microsoft's web browser.	64-bit executable
Hearts.exe	A small game that is preinstalled with windows.	64-bit executable
Link.exe	A command-line linker application.	32-bit executable
Maya.exe	3D animation software used to produce realistic computer generated scenes.	32-bit executable
Mdbg.exe	The 64-bit version of .NET framework's command-line debugger.	64-bit executable
Mpc-hc.exe	Classic Media Player	32-bit executable
NTDLL.dll	Exports the Windows Native API. Part of all Windows' native applications	32-bit dynamic library
Opera.exe	Freeware web browser.	32-bit executable
POAsm.exe	An assembler used by MASM.	32-bit executable
PES2010.exe	A game published by Konami that makes intensive use of 3D graphics, audio and AI.	32-bit executable
Shift.exe	A game published by Electronic Arts.	32-bit executable
WinRAR.exe	A popular application for compressing and decompressing RAR files among other compressed extensions	64-bit executable
Wordpad.exe	A simple editor for rich text files (rtf).	64-bit executable

3. Overview of the x86-64 Architecture

The x86-64 is an extension of the x86 architecture that includes 64-bit support. The extensions include 64-bit registers, larger address and virtual spaces.

A. Registers:

In addition to the x86 registers, the x86-64 architecture introduces new registers (RAX, RBX, RCX, RDX, RBP, RSP, RSI, RDI, R8, R9, R10, R11, R12, R13, R14, R15). The R8-R15 are newly introduced general purpose registers. RAX, RBX, RCX, RDX, RBP, RSP, RSI, RDI, RIP are 64-bit extensions of the 32-bit registers, EAX, EBX, ECX, EDX, EBP, ESP, ESI, EDI, EIP [9].

B. Addressing Modes:

The architecture has several addressing modes that have been used in this document [9].

1) *Register Addressing Mode:* when the data resides in a register.

mov RAX, RCX

2) *Based Addressing Mode:* when the contents of a register (with or without a displacement value) is used as the effective address to retrieve the data from memory.

mov RAX, [RBX]
mov RAX ,[RBX+3]

3) *Indexed Addressing Mode:* when the contents of a register is multiplied by a scalar (with or without a displacement value) to calculate the effective address of the data in memory.

*mov RAX, [RSP*4]*
*mov RAX,[RAX*4+3]*

4) *Based Index Addressing Mode:* when the contents of two registers are used in the calculation of the effective address. One of the registers can be multiplied by a scalar and a displacement value may or may not be used.

mov RAX, [RBX+RSP]
*mov RAX, [RBX+RSP*4]*
*mov RAX, [RBX+RSP*4+3]*

5) *Direct Addressing Mode:* when the effective address is hardcoded in the instruction's code and therefore doesn't require any calculation.

mov RAX, [00FF455400000008]

6) *Immediate Addressing Mode*: when the data exists in the instructions' code itself

mov RAX, 10h

4. Results Analysis

The results of each category are divided into five parts:

1) *Instruction frequencies:* The number of times an instruction has been found in the dissassembled code. This provides information on the most commonly used instructions for each category.

2) *Register's utilization:* The utilization of each register. This provides an indication of memory usage (indicated by index and data registers' usage), stack usage (indicated by stack registers' usage).

3) *Addressing Modes:* The number of uses of each addressing mode. This provides an insight of the optimization level of the application. An excessive use of costly addressing modes such as "based-indexed" modes indicates a significant overhead that can be avoided if the application is further optimized.

4) *Average instruction length:* The instruction lengths represent the application's memory consumption. Advanced compilers are capable of replacing instructions with shorter instructions to save memory and CPU fetch cycles.

5) *Number of instructions vs number of operands of an instruction:* The number of zero-operand, one-operand, two-operand and three-operand instructions for each category. This helps the intermediate language designers to assign operation codes (Opcodes).

5. Results

To present the results in a descriptive and simple fashion, applications have been divided up into five categories. The results of each category are demonstrated each with the divisions described in Section III except for registers' utilization. The applications are then categorized according to their bittedness and the statistics are generated for all five divisions including the registers' utilization. The summary of these categories' results is then presented at the end.

A. Web Browsers:

This category is made up of *opera.exe* and *iexplorer.exe*. It represents internet applications that make use of text parsing, datastreams, threads, and the consumer-producer model. The results of this category are shown in *Fig. 2* through *Fig. 5*.

The *push* instruction is used twice more than the *pop* instruction. This is because the *push* instruction is used to pass arguments to functions. A number of *nop* instructions are used to empty the processor's pipeline while a *jmp* instruction is executed. This caused 6% of the instructions to be *nop*. 23% of the code was dedicated to moving data among registers and memory. The conditional jump instructions, *jnz* and *jz*, are also used for looping over variables such as semaphore counters.

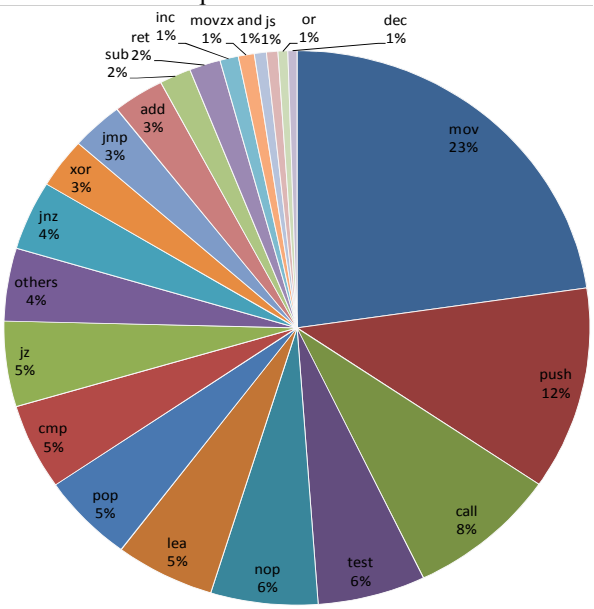


Fig. 2 Web browsers' instruction frequencies

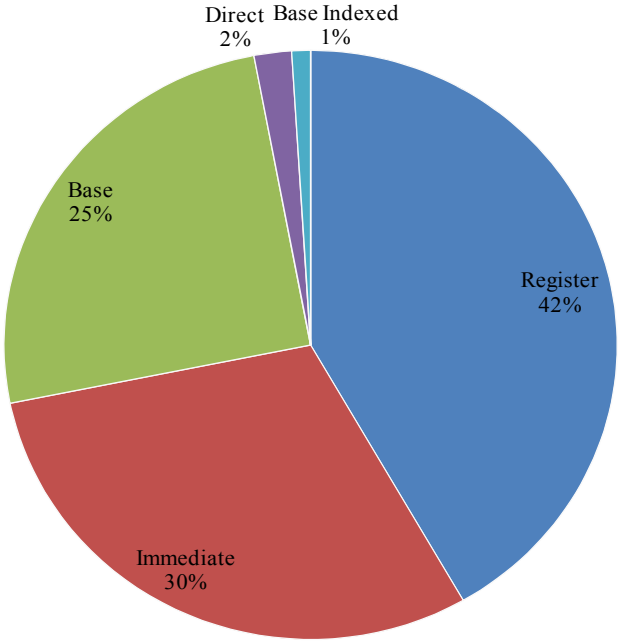


Fig. 3 Web browsers' addressing modes

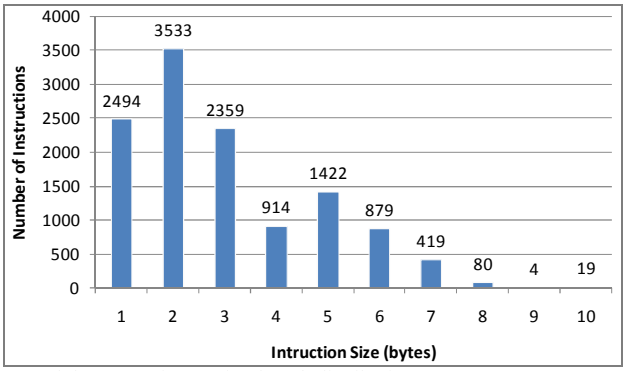


Fig. 4 Web browsers' instruction length distribution

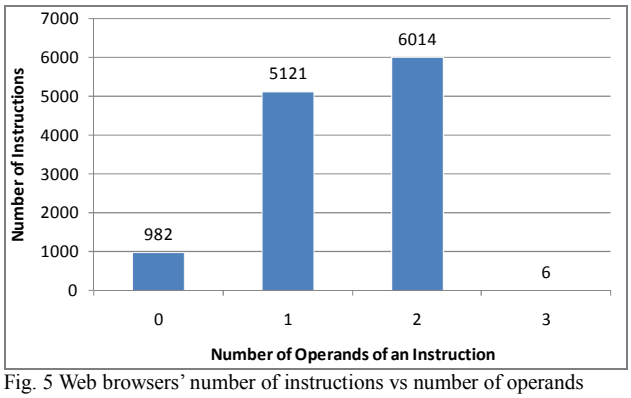


Fig. 5 Web browsers' number of instructions vs number of operands

B. *Graphics Applications:*

Graphics Applications represent executables that utilize the graphics capabilities of a system. 3D animation packages such as *maya.exe* and 2D/3D Games such as *hearts.exe*, *pes2010.exe*, and *shift.exe* make intensive use of graphics resources. Besides the use of graphics, games employ audio, network and algorithm programming.

The results of this category are shown in *Fig. 6* through *Fig. 9*.

The debug exception instruction (*int3*) [6] makes up 41% of the graphics applications. Function calls and passing parameters to graphics APIs, graphics engines and other APIs (represented by *call*, *push* and *ret* instructions) form 18% of the number of instructions used. The *add* operation is 3% of the instructions necessary for geometrical and algorithmic calculations. The *mov* instruction forms a significant percentage of the instructions (16%) used to move data to and from the graphics-processing unit and memory. Due to the copiousness of the *int3* instruction (one byte), the average instruction size is very close to one byte. This also affects the number of zero-operand instructions.

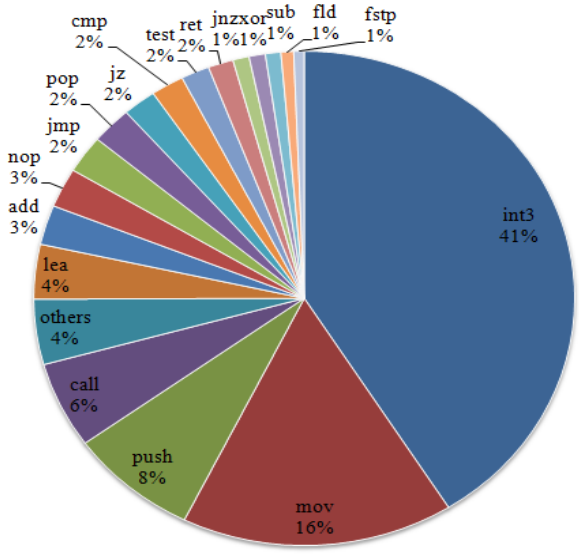


Fig. 6 Graphics Applications' instruction frequencies

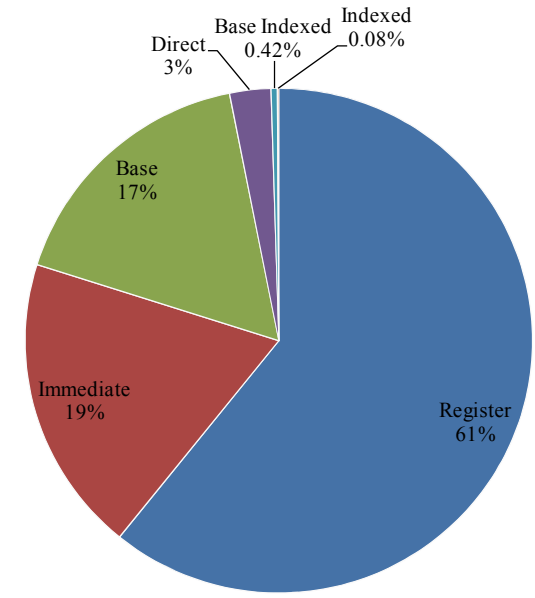


Fig. 7 Graphics Applications' addressing modes

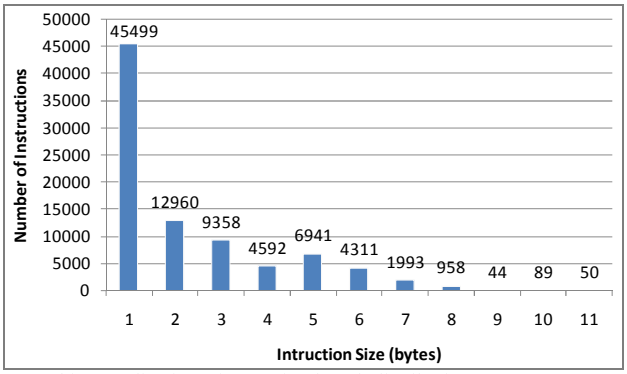


Fig. 8 Graphics Applications' instruction length distribution

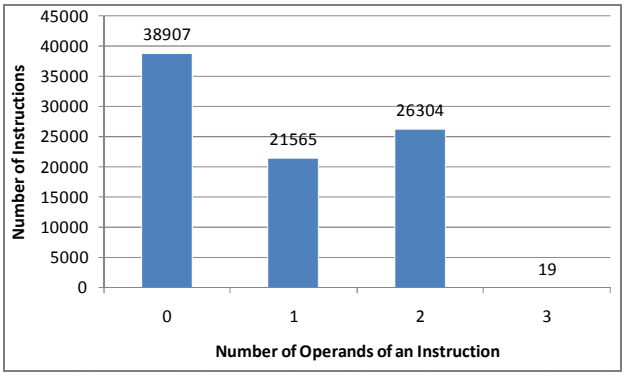


Fig. 9 Graphics Applications' number of instructions vs number of operands

C. *OS related components:*

This category contains all the analyzed operating system related components, namely *chkdsk.exe*, *ntdll.dll*, and

explorer.exe. Analysis of this category demonstrates the difference between operating system's code and that of others.

The results of this category are shown in *Fig. 10* through *Fig. 13*.

Just as for the other categories, the *mov*, *add* and *push* instructions are among the most used. The *nop* instruction contributes to 7% of the components' assembly code. Although some of the *nop* instruction occurrences were used for flushing the pipeline while a jump or *ret* instruction, other occurrences were only present for delay purpose. The memory reference mode comprises 33% of the addressing modes.

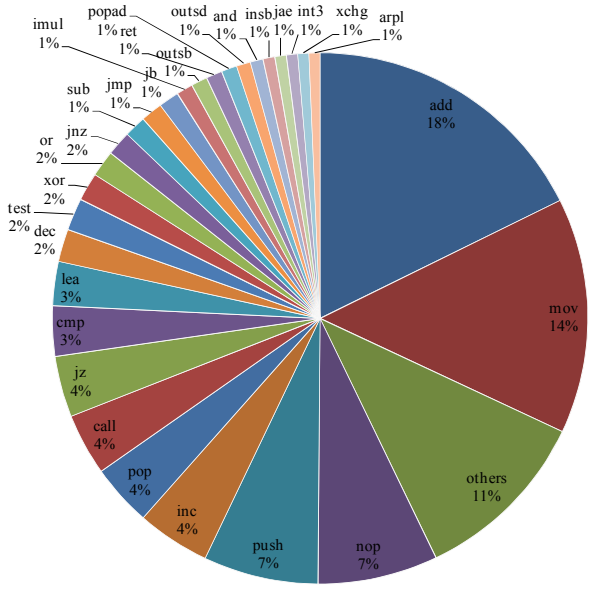


Fig. 10 OS Components' instruction frequencies.

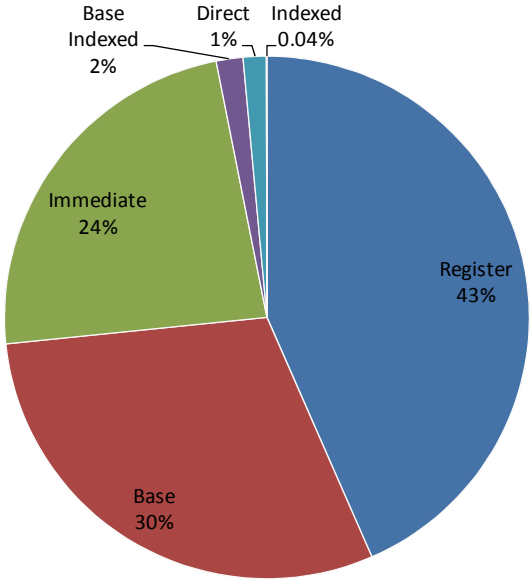


Fig. 11 OS Components' addressing modes

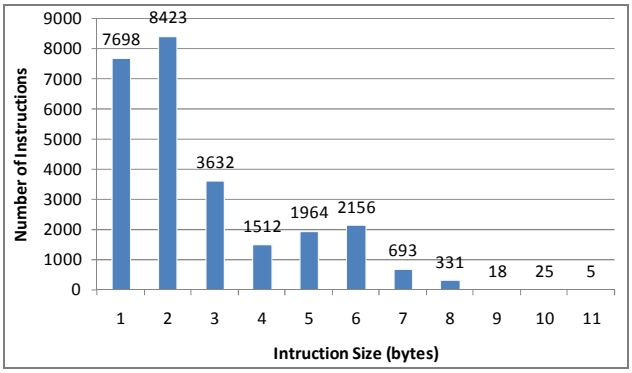


Fig. 12 OS Components' instruction length distribution

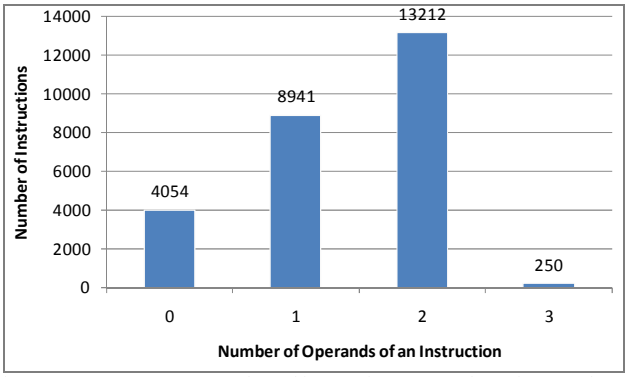


Fig. 13 OS Components' number of instructions vs number of operands

D. General Purpose Applications:

Acrobat.exe, *mpc-hc.exe*, *winrar.exe* and *wordpad.exe* comprise this category. It represents commonly used applications that are used more often than others. Compared to the other categories, this category represents a close approximation to the average results of all the other applications.

The results of this category are shown in *Fig. 14* through *Fig. 17*.

Although the instruction frequencies do not have a significant change compared to other categories, it closely resembles the results of the Web Browser's category.

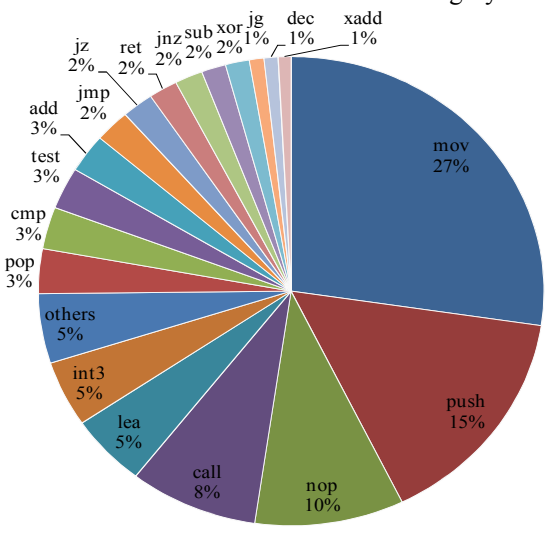


Fig. 14 General Purpose Applications' Instruction Frequencies

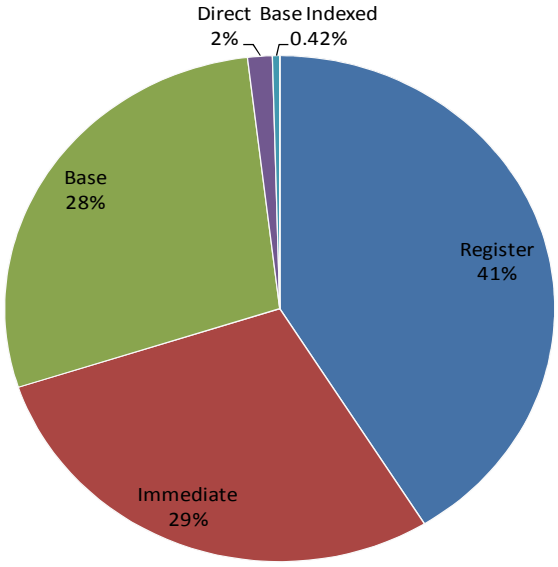


Fig. 15 General Purpose Applications' addressing modes

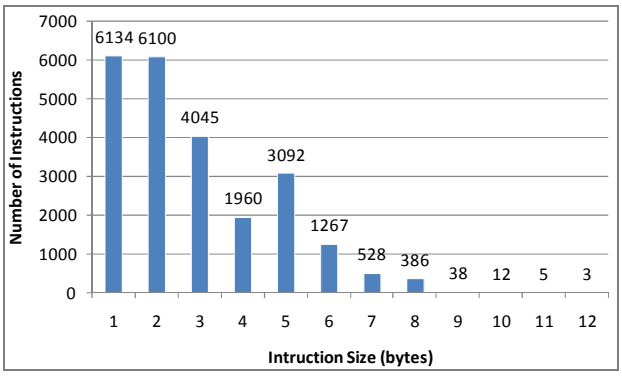


Fig. 16 General Purpose Applications' instruction lengths distribution

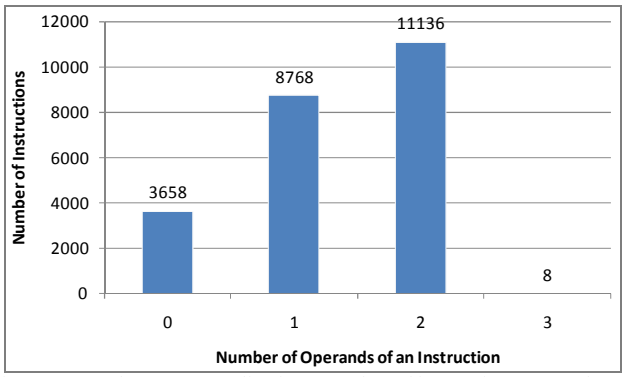


Fig. 17 General Purpose Applications' number of instructions vs number of operands

E. Software Development Tools:

This category is for tools such as compilers, debuggers, linkers, and disassemblers. *Link.exe*, *Mdbg.exe* and *POAsm.exe* are analyzed to represent this category. The results of this category are shown in *Fig 18* through *Fig 21*.

Logic instructions such as *or* (7%) and *and* (7%) are used intensively in this type of applications. It is notable that the register addressing mode is used less than any of the other categories. Memory reference instructions make up 54%, which is the highest percentage among all categories.

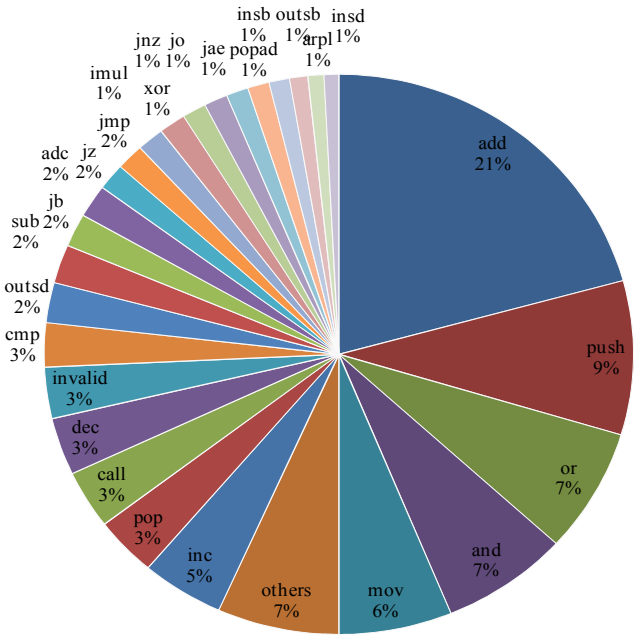


Fig. 18 Software Development Tools' Instruction Frequencies

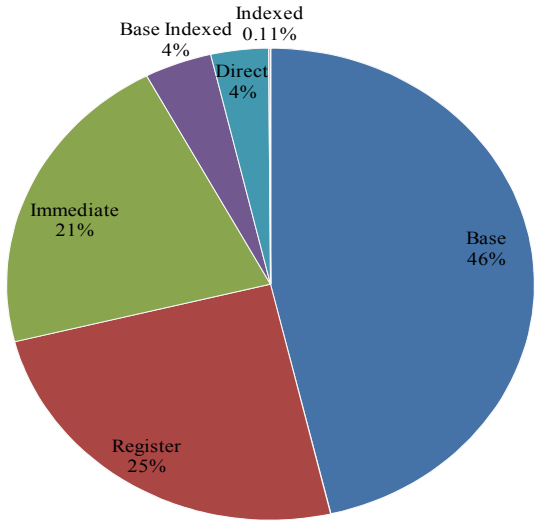


Fig. 19 Software Development Tools' addressing modes

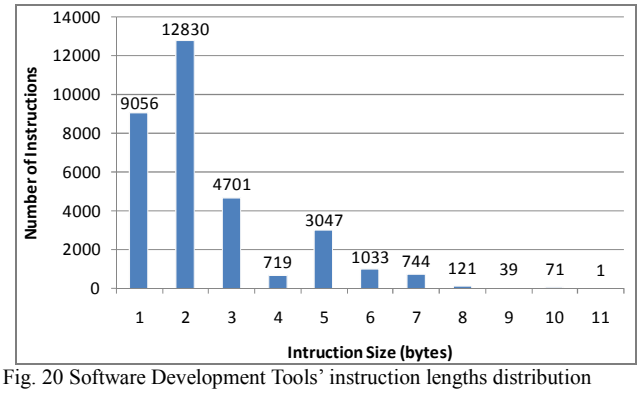


Fig. 20 Software Development Tools' instruction lengths distribution

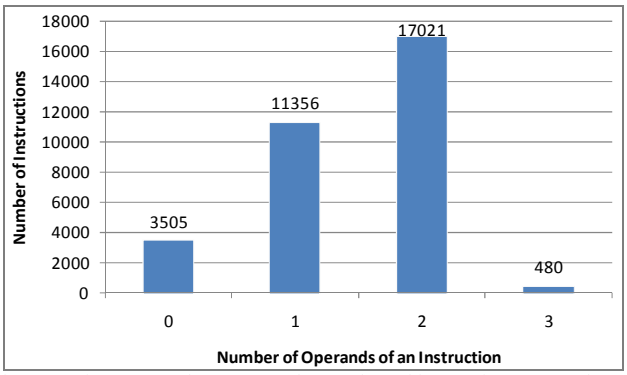


Fig. 21 Software Development Tools' number of instructions vs number of operands

F. 32-bit Executables and DLLs:

This is an analysis for 32-bit binary files. It examines the registers' utilization as well as all the other result divisions. This category is composed of acrobat.exe, chkdsk.exe, link.exe, maya.exe, mpc-hc.exe, ntdll.dll, opera.exe, poasm.exe, pes2010.exe and shift.exe. Figure 22 to Figure 26 illustrate the results of all this category.

The instruction frequencies have 33% of the int3 instruction, 14% for the mov instruction and the push instruction composes 11%. The 32-bit Accumulator register (EAX) is the most used register with a 27% usage, followed by the 32-bit counter register (ECX). Register reference addressing modes comprised more than 50% of the addressing modes with the rest divided between memory reference and immediate addressing modes. The average instruction length is 2.35 bytes. The instructions are almost evenly distributed among zero-operand, one-operand, and two-operand instructions.

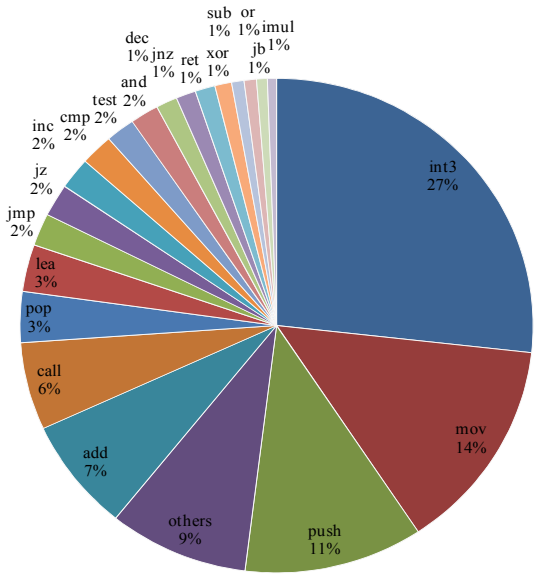


Fig. 22 32-bit Executables and DLLs' Instruction Frequencies

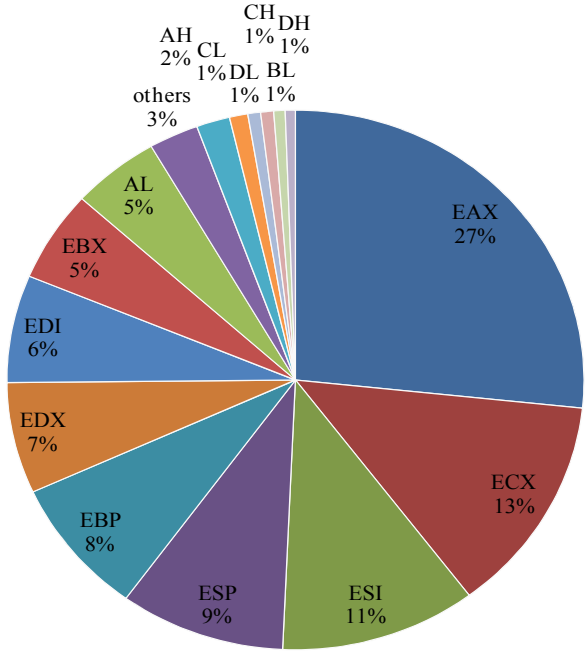


Fig. 23 32-bit Executables and DLLs' registers' utilization

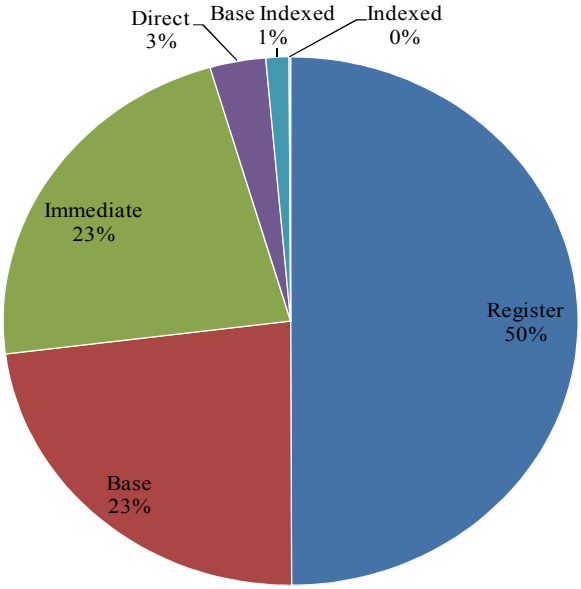


Fig. 24 32-bit Executables and DLLs' addressing modes

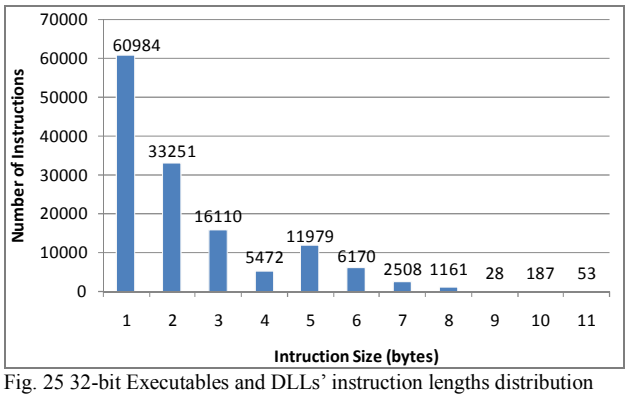


Fig. 25 32-bit Executables and DLLs' instruction lengths distribution

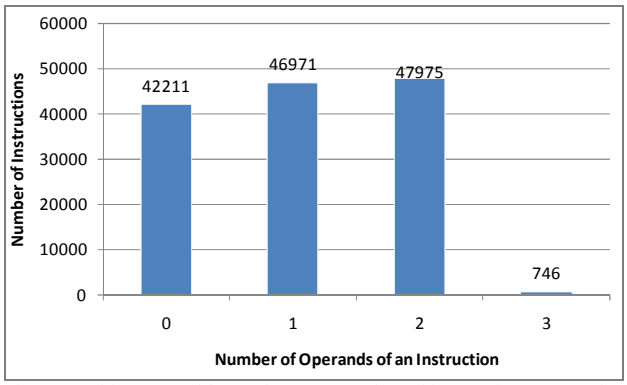


Fig. 26 32-bit Executables and DLLs' number of instructions vs number of operands

G. 64-bit Executables and DLLs:

Similar to the previous category, this subsection contains the results of 64-bit applications. *Explorer.exe*, *iexplorer.exe*, *hearts.exe*, *mdbg.exe*, *winrar.exe* and *wordpad.exe* are the applications classified in this category. The registers' utilization is also included in this subsection for 64-bit applications. The results are in *fig. 27* to *fig. 31*.

Similar to 32-bit applications, the accumulator register (RAX) is the most used register (13%). Both the stack pointer (RSP) and counter register (RCX) have the second most frequent utilization with 9% and 8% respectively. The *mov* instruction is the most frequent instruction (23%), followed by the *nop* instruction, which has a 15% frequency. Otherwise, the instruction frequencies are similar to that of 32-bit instructions. The average instruction size is 3.12 bytes, which is less than twice the average of the 32-bit application (considering that addresses and immediate values are twice the size). Two-operand instructions are more than zero-operand and one-operand instructions.

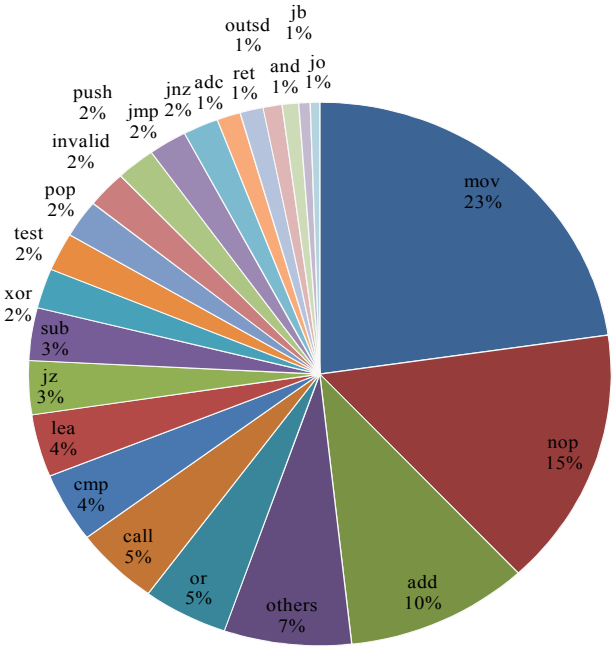


Fig. 27 64-bit Executables and DLLs' Instruction Frequencies

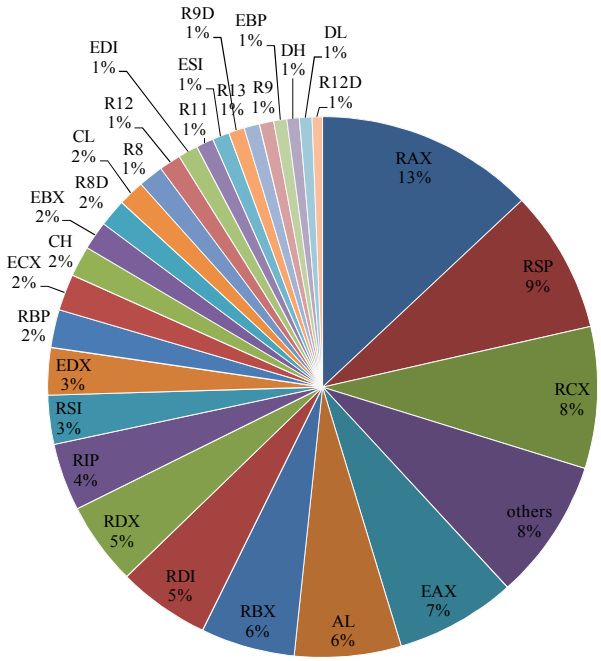


Fig. 28 64-bit Executables and DLLs' registers' utilization

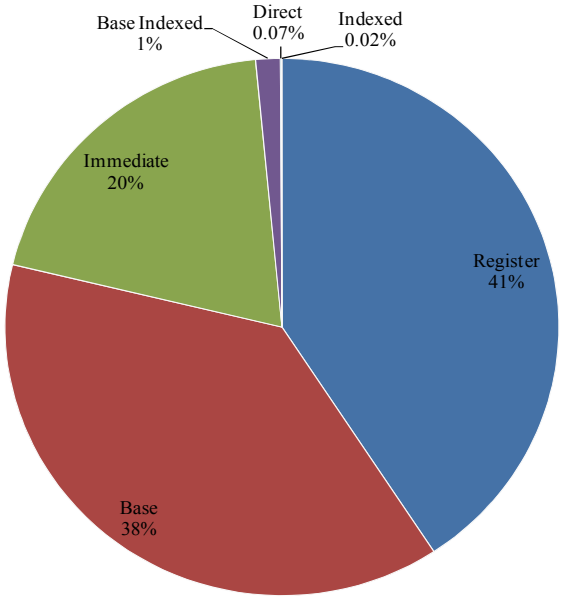


Fig. 29 64-bit Executables and DLLs' addressing modes

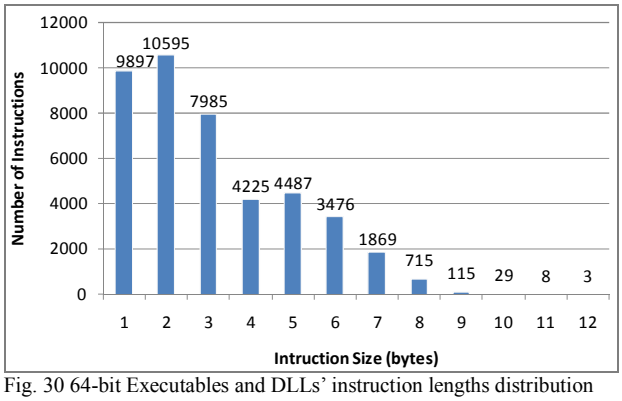


Fig. 30 64-bit Executables and DLLs' instruction lengths distribution

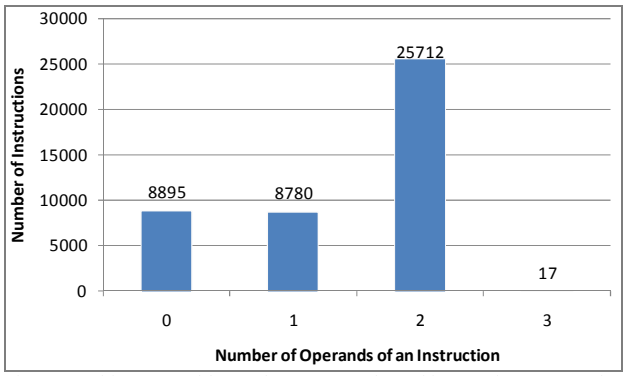


Fig. 31 64-bit Executables and DLLs' number of instructions vs number of operands

H. Summary:

This section summarizes all the outcomes of all the analyzed applications. It is intended to bring forth the overall statistics that have been generated to facilitate drawing a conclusion. Instead of averaging the results from the application categories, the summarized statistics have been generated by summing up all the results of individual applications together to eliminate rounding errors that might have occurred during the results' generation of each category.

The overall results are shown in Fig. 18 through Fig. 21 followed by Table 2. Register utilization is not included in the summary, since combining the register utilization of both 32-bit and 64-bit applications has no significance.

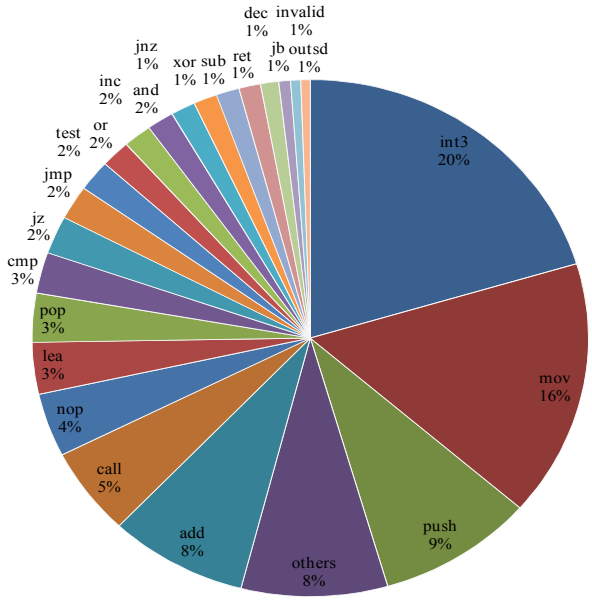


Fig. 32 Summary of the instruction frequencies

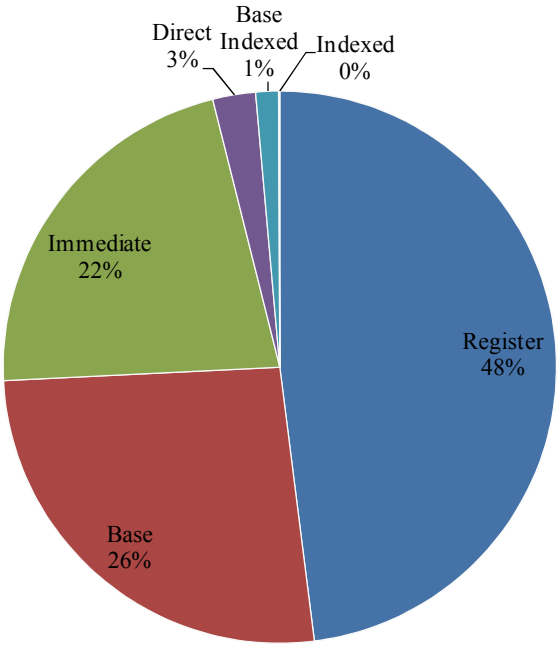


Fig. 33 Summary of the addressing modes

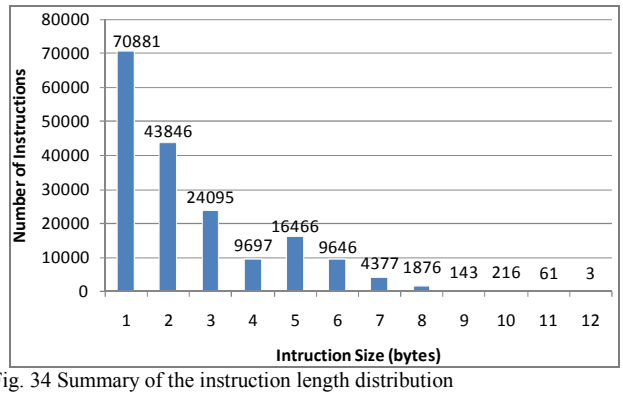


Fig. 34 Summary of the instruction length distribution

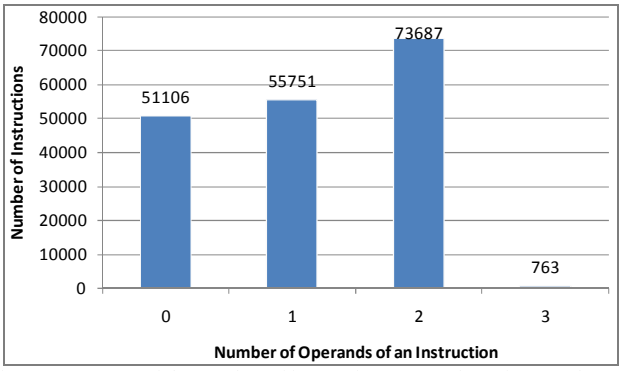


Fig. 35 Summary of the number of instructions vs number of operands

TABLE 2

SUMMARY OF THE ANALYSIS

Total number of instructions:	181307
Number of instruction mnemonics used:	244
Average instruction size (in Bytes):	2.54
Top 20 most used instruction mnemonics (in descending order): int3 –mov – push – add – call – nop – lea – pop –cmp – jz – jmp –test – or – inc – and – jnz – xor – sub – ret	

6. Conclusion

An analysis has been conducted as a tool for system software development, application programming and intermediate language design. A group of different executable and library files have been selected from various software categories. The chosen application binary files were grouped into categories. Each category has been analyzed for instruction frequencies, addressing modes, instruction lengths and the number of instructions for each number of operands (zero-operand, one-operand, two-operand and three-operand). The applications have been also classified according to their bittedness. Each of the two classifications (32-bit and 64-bit) has been analyzed for all the result divisions including registers' utilization. The instruction frequencies show that 37 instructions out of 391 instructions [7-8] cover over 95% of the instructions used in applications. The *mov* and *push* instructions comprise 15% and 9% respectively of the instruction frequencies. The results further show that 20% of the applications' code consist of debug exception instructions while another 4% are made of no operation instructions (*nop*).

Three-operand instructions have the smallest number of instructions for all categories and classifications, while two-operand instructions are the most used instructions in all categories and classifications. The accumulator register (RAX or EAX) are the most utilized registers.

The memory reference addressing modes comprise 28% of the instructions' addressing modes. As the optimum addressing mode for performance, register addressing mode make up 49% of the used addressing modes.

The instructions' lengths distribution shows that the average instruction length is around 2~3 bytes.

In the future, we would like to extend our analysis to include the x86-64 processors in real mode since the protected mode is the only accessible mode in Windows 7 operating system. Moreover, we would like to perform the same analysis for the other architectures, such as IBM PowerPC.

References

[1] I. J. Huang & T. C. Peng, "Analysis of x86 Instruction Set Usage for DOS/Windows Applications and Its Implication on Superscalar Design," (2002) *IEICE Transactions on Information and Systems*, vol. 85, no. 6, pp. 929-939.

[2] T. C. Peng, "Analysis of x86 Instruction Set Usage in DOS/Win95 Applications and It's Implication on Microarchitecture Design", (1997) *Master Thesis, Institute of Computer and Information Engineering, National Sun Yat-sen University, Taiwan, R. O. C.*

[3] R. Rico, "Proposal of test-bench for the x86 instruction set (16 bits subset)," (2005) *Technical Report TR-UAH-AUT-GAP-2005-21-en*.

[4] Microsoft Corporation, "Microsoft portable executable and common object file format specification", version 8.1, available online @ http://www.microsoft.com/whdc/system/platform/firmware/PECOFF.mspx, 2008

[5] V. Thampi, "udis86 Disassembler Library for x86 and x86-64", http://udis86.sourceforge.net

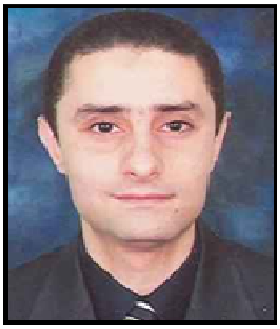
[6] Intel Corporation, "Intel 64 and IA-32 Architectures Software Developer's Manual Volume 3A: System Programming Guide", Part 1, 2009.

[7] Intel Corporation, "Intel 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference A-M", 2009.

[8] Intel Corporation, "Intel 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference N-Z", 2009.

[9] Intel Corporation, "Intel 64 and IA-32 Architectures Software Developer's Manual Volume 1: Basic Architecture", 2009.

Amr Hussam Ibrahim has gotten his BSc from the department of Computer Engineering in the Arab Academy of Science And Technology. He is currently pursuing his MSc and is a Teaching Assistance in Computer Engineering at the Arab Academy of Science And Technology. His research interests include Computer Algorithms, Software Optimization, Embedded Systems and FPGA Design.



Mohamed Bakr Abdelhalim received his BSc, MSc and PhD degrees in Electronics Engineering from Cairo University in 1999, 2003 and 2008 respectively. He was a researcher at the Department of Electrical and Communications Engineering, Cairo University. He was also a teaching and lab assistant at the Electronics Engineering Department, American University in Cairo. Currently he is an assistant professor at the College of Computing and Information Technology - Arab Academy for Science, Technology, and Maritime Transportation. His research interests are FPGA-based design, System-level design and description languages, CAD tools, and Soft computing techniques.

Hanady Hussein is currently an assistant professor at Arab Academy for Science and Technology, electronics and Communication department, Egypt. She received her Ph.D. degree in Electronics and Communication Engineering from Ain Shams University in 2009. Her research interests include Digital/Analog design, VHDL based FPGA design, simulation and synthesis.

Ahmed Fahmy Amin is a professor in Computer Engineering at AAST. He has a Ph.D. in Electronics and Communications from Cairo University, Egypt and a Ph.D. in Computer Engineering from NPGS, USA.