Assembly level analysis of certain cryptography algorithms for Instruction Set Extension

A Project Report

submitted by

APTE PRIYA NARAYANRAO, EE16M044

in partial fulfilment of requirements
for the award of the degree of

MASTER OF TECHNOLOGY

DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY MADRAS

MAY 2018

THESIS CERTIFICATE

This is to certify that the thesis titled **Assembly level analysis of certain Cryptography algorithms for Instruction Set Extension** , submitted by **Apte Priya Narayanrao** bearing Roll number **EE16M044**, to the Indian Institute of Technology, Madras, for the award of the degree of **Master of Technology**, is a bona fide record of the research work done by him under our supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Dr. V. Kamakoti
Project Guide
Professor
Dept. of Computer Science
and Engineering
IIT Madras, 600036

Place: Chennai

Date: 10th May 2018

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude towards several people who enabled me to reach this far with their timely guidance, support and motivation.

Firstly, I would like to thank my guide, Prof. V. Kamakoti who despite his very busy schedule, always gave me a patient hearing and showed me the way through thick and thin on all matters concerned. His knowledge and an extraordinary ability to lighten the students with his positive approach towards things always infused me with great energy and positivity. The invaluable inputs and suggestions from him enabled me to achieve the desired goals during the project work.

I would like to extend a special thanks to Dr.Chester for his invaluable suggestions on Cryptography.I would also like to thank my faculty advisor Dr.Saurabh Saxena who continuously supported me throughout my course with his advise.

My special thanks to Dr.Neel Gala, Arjun and Rahul who have been very supportive during the course of this project.They have enriched the project experience with their knowledge of the subject matter, active participation and invaluable suggestions.

I thank to all my RISE lab-mates whose acquaintance and support helped me in one way or other, throughout this learning process.

I would like to thank my parents and my best friend Anamika Sinha for always standing by me.

ABSTRACT

Data Encryption/Decryption has become an essential part of computing systems. However, executing these cryptographic algorithms often introduces a high overhead. It can bog down a processor and slow down the overall transaction. To speed up the computations on its advanced processors,new instructions can be added to accelerate encryption and decryption.In this project I first analyzed assembly code of various cryptography algorithms on executing them on RISCV cross compiler. With the help of python script profiling is carried out to identify potential set of consecutive instructions for ISE.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Data security is important in pervasive computing systems because the secrecy and integrity of the data should be retained when they are transferred among mobile devices and servers in this system. The cryptography algorithm is an essential part of the computing security mechanism.However, performance is one concern regarding cryptography algorithms: these algorithms are extremely expensive in terms of execution time. To make cracking the code more difficult, many arithmetic and logical operations are bound to be executed during the encryption/decryption process. Furthermore, a huge amount of data needs to be transferred between the CPU and the memory. Using a general-purpose processor for this purpose would not be a cost-effective solution, and the performance is not that satisfying either.

There are roughly three dominant paradigms for hardware crypto. The first is a wholly-separate device connected via a system bus, which implements various functions. One of the key advantages of this is that the sensitive data can remain on the device, never accessible to the rest of the system . However, this can't rightly be considered an ISA extension, as it's wholly-separate.The other end of the spectrum is occupied by cipher-specific instructions such as Intel's AESNI instruction set. These are often very efficient, as many cryptographic ciphers can be implemented very efficiently in hardware. However, they don't do much for protection of sensitive data. Moreover, writing specific ciphers into the ISA is generally a bad idea. Ciphers are sometimes broken, and more often phased out and replaced by newer, better algorithms. Moreover, such a practice can enshrine weak crypto, as is seen in the continuing use of weak and broken crypto.In my project I began by attempting to generalize the instruction-based approach, initially planning for a generalized framework for crypto instructionsGrabher *et al.* (2008).

In this project various cryptography algorithms impplemented in high level language of C and analyzed their assembly code on executing them on RISCV cross compiler. With the help of python script static as well as runtime assembly level profiling is

carried out to identify potential set of consecutive instructions for ISE. Overall top 10 set of consecutive instructions are found with and without operand dependency.

CHAPTER 2

CRYPTOGRAPHY OVERVIEW

Cryptography is the science of secret writing is an ancient art; the first documented use of cryptography in writing dates back to circa 1900 B.C. when an Egyptian scribe used non-standard hieroglyphs in an inscription. Some experts argue that cryptography appeared spontaneously sometime after writing was invented, with applications ranging from diplomatic missives to war-time battle plans. It is no surprise, then, that new forms of cryptography came soon after the widespread development of computer communications. In data and telecommunications, cryptography is necessary when communicating over any untrusted medium, which includes just about any network, particularly the InternetKatz *et al.* (1996).

There are five primary functions of cryptography today:

1. Privacy/confidentiality: Ensuring that no one can read the message except the intended receiver.
2. Authentication: The process of proving one's identity.
3. Integrity: Assuring the receiver that the received message has not been altered in any way from the original.
4. Non-repudiation: A mechanism to prove that the sender really sent this message.
5. Key exchange: The method by which crypto keys are shared between sender and receiver.

In cryptography, we start with the unencrypted data, referred to as plaintext. Plaintext is encrypted into ciphertext, which will in turn (usually) be decrypted back into usable plaintext. The encryption and decryption is based upon the type of cryptography scheme being employed and some form of key. For those who like formulas, this process is sometimes written as:

$$C = E_k(P)$$

$$P = D_k(C)$$

where P = plaintext, C = ciphertext, E = the encryption method, D = the decryption method, and k = the key.

In many of the descriptions below, two communicating parties will be referred to as Alice and Bob; this is the common nomenclature in the crypto field and literature to make it easier to identify the communicating parties. If there is a third and fourth party to the communication, they will be referred to as Carol and Dave, respectively. A malicious party is referred to as Mallory, an eavesdropper as Eve, and a trusted third party as TrentKatz *et al.* (1996).

Finally, cryptography is most closely associated with the development and creation of the mathematical algorithms used to encrypt and decrypt messages, whereas cryptanalysis is the science of analyzing and breaking encryption schemes. Cryptology is the term referring to the broad study of secret writing, and encompasses both cryptography and cryptanalysis.

2.1 TYPES OF CRYPTOGRAPHIC ALGORITHMS

There are several ways of classifying cryptographic algorithms.They will be categorized based on the number of keys that are employed for encryption and decryption, and further defined by their application and use. The three types of algorithms2.1:

- Secret Key Cryptography (SKC): Uses a single key for both encryption and decryption; also called symmetric encryption. Primarily used for privacy and confidentiality.
- Public Key Cryptography (PKC): Uses one key for encryption and another for decryption; also called asymmetric encryption. Primarily used for authentication, non-repudiation, and key exchange.
- Hash Functions: Uses a mathematical transformation to irreversibly "encrypt" information, providing a digital fingerprint. Primarily used for message integrity.

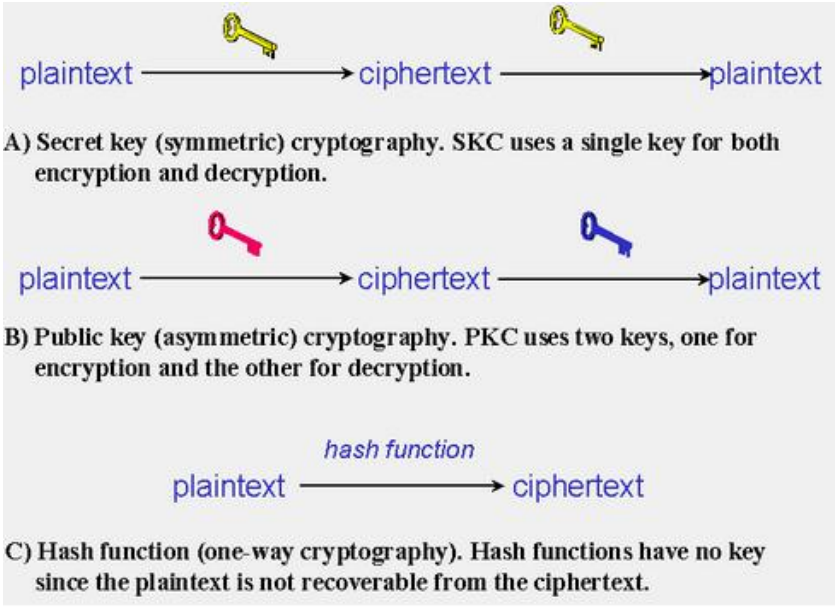


Figure 2.1: Three types of cryptography: secret key, public key, and hash function

2.1.1 Secret Key Cryptography

Secret key cryptography methods employ a single key for both encryption and decryption.The sender uses the key to encrypt the plaintext and sends the ciphertext to the receiver. The receiver applies the same key to decrypt the message and recover the plaintext. Because a single key is used for both functions, secret key cryptography is also called symmetric encryption.

We choose seven popular secret key cryptography algorithms as benchmarks for our study:

- **Advanced Encryption Standard (AES):** In 1997, NIST initiated a very public, 4-1/2 year process to develop a new secure cryptosystem for U.S. government applications (as opposed to the very closed process in the adoption of DES 25 years earlier). The result, the Advanced Encryption Standard, became the official successor to DES in December 2001. AES uses an SKC scheme called Rijndael, a block cipher designed by Belgian cryptographers Joan Daemen and Vincent Rijmen. The algorithm can use a variable block length and key length; the latest specification allowed any combination of keys lengths of 128, 192, or 256 bits and blocks of length 128, 192, or 256 bits. AES is an iterative rather than Feistel cipher. It is based on âĂŸsubstitutionâĂŞpermutation networkâĂŹ. It comprises of a series of linked operations, some of which involve replacing inputs by specific outputs (substitutions) and others involve shuffling bits around (permutations).

- **Data Encryption Standard (DES):** One of the most well-known and well-studied SKC schemes, DES was designed by IBM in the 1970s and adopted by the National Bureau of Standards (NBS) [now the National Institute for Standards

and Technology (NIST)] in 1977 for commercial and unclassified government applications. DES is a Feistel block-cipher employing a 56-bit key that operates on 64-bit blocks. DES has a complex set of rules and transformations that were designed specifically to yield fast hardware implementations and slow software implementations, although this latter point is not significant today since the speed of computer processors is several orders of magnitude faster today than even twenty years ago.

- **Rivest Ciphers (aka Ron's Code):** Named for Ron Rivest, a series of SKC algorithms.
  - RC4: A stream cipher using variable-sized keys; it is widely used in commercial cryptography products.
  - RC2: A 64-bit block cipher using variable-sized keys designed to replace DES.

- **Blowfish:** A symmetric 64-bit block cipher invented by Bruce Schneier; optimized for 32-bit processors with large data caches, it is significantly faster than DES on a Pentium/PowerPC-class machine. Key lengths can vary from 32 to 448 bits in length. Blowfish, available freely and intended as a substitute for DES or IDEA, is in use in a large number of products.

- **Camellia:** A secret-key, block-cipher crypto algorithm developed jointly by Nippon Telegraph and Telephone (NTT) Corp. and Mitsubishi Electric Corporation (MEC) in 2000. Camellia has some characteristics in common with AES: a 128-bit block size, support for 128-, 192-, and 256-bit key lengths, and suitability for both software and hardware implementations on common 32-bit processors as well as 8-bit processors (e.g., smart cards, cryptographic hardware, and embedded systems).

- **GOST(Block Cipher):** GOST is a Feistel network of 32 rounds.Its S-boxes can be secret, and they contain about 354 (log2(16!8)) bits of secret information, so the effective key size can be increased to 610 bits.

2.1.2 Public Key Cryptography

Public key cryptography has been said to be the most significant new development in cryptography in the last 300-400 years. Modern PKC was first described publicly by Stanford University professor Martin Hellman and graduate student Whitfield Diffie in 1976. Their paper described a two-key crypto system in which two parties could engage in a secure communication over a non-secure communications channel without having to share a secret key.

PKC depends upon the existence of so-called one-way functions, or mathematical functions that are easy to compute whereas their inverse function is relatively difficult to compute. Generic PKC employs two keys that are mathematically related although knowledge of one key does not allow someone to easily determine the other key. One key is used to encrypt the plaintext and the other key is used to decrypt the ciphertext. The important point here is that it does not matter which key is applied first, but that both keys are required for the process to work (Figure 1B). Because a pair of keys are required, this approach is also called asymmetric cryptography.

We choose following popular public key cryptography algorithms as benchmarks for our study:

- **RSA:** The first, and still most common, PKC implementation, named for the three MIT mathematicians who developed it âĂŤ Ronald Rivest, Adi Shamir, and Leonard Adleman. RSA today is used in hundreds of software products and can be used for key exchange, digital signatures, or encryption of small blocks of data. RSA uses a variable size encryption block and a variable size key. The key-pair is derived from a very large number, n, that is the product of two prime numbers chosen according to special rules; these primes may be 100 or more digits in length each, yielding an n with roughly twice as many digits as the prime factors. The public key information includes n and a derivative of one of the factors of n; an attacker cannot determine the prime factors of n (and, therefore, the private key) from this information alone and that is what makes the RSA algorithm so secure. (Some descriptions of PKC erroneously state that RSA's safety is due to the difficulty in factoring large prime numbers.
- **Elliptic Curve Cryptography (ECC):** In 1985, Elliptic Curve Cryptography (ECC) was proposed independently by cryptographers Victor Miller (IBM) and Neal Koblitz (University of Washington). ECC is based on the difficulty of solving the Elliptic Curve Discrete Logarithm Problem (ECDLP). Like the prime factorization problem, ECDLP is another "hard" problem that is deceptively simple to state: Given two points, P and Q, on an elliptic curve, find the integer n, if it exists, such that P = nQ.

2.1.3 Hash Functions

Hash functions, also called message digests and one-way encryption, are algorithms that, in essence, use no key. Instead, a fixed-length hash value is computed based upon the plaintext that makes it impossible for either the contents or length of the plaintext to be recovered. Hash algorithms are typically used to provide a digital fingerprint of a file's contents, often used to ensure that the file has not been altered by an intruder or virus. Hash functions are also commonly employed by many operating systems to encrypt passwords. Hash functions, then, provide a mechansim to ensure the integrity of a file.

We choose folowing hash functions as benchmark:

- **MD2:** Designed for systems with limited memory, such as smart cards.MD2 is an earlier, 8-bit version of MD5, an algorithm used to verify data integrity through the creation of a 128-bit message digest from data input (which may be a message of any length) that is claimed to be as unique to that specific data as a fingerprint is to the specific individual.
- **MD5:** Ronald Rivest, founder of RSA Data Security and institute professor at MIT, designed MD5 as an improvement to a prior message digest algorithm, MD4.The MD5 algorithm is intended for digital signature applications, where a large file must be 'compressed' in a secure manner before being encrypted with a private (secret) key under a public-key cryptosystem such as RSA.
- **Blake224:**It is cryptographic hash function based on Dan Bernstein's ChaCha stream cipher, but a permuted copy of the input block, XORed with some round constants, is added before each ChaCha round.
- **Snefru:**Snefru is a cryptographic hash function invented by Ralph Merkle in 1990 while working at Xerox PARC.[1] The function supports 128-bit and 256-bit output.
- **SipHash:** SipHash is an addâĂŞrotateâĂŞxor (ARX) based family of pseudorandom functions created by Jean-Philippe Aumasson and Daniel J. Bernstein in 2012 in response to a spate of "hash flooding" denial-of-service attacks in late 2011.

  Although designed for use as a hash function in the computer science sense, SipHash is fundamentally different from cryptographic hash functions like SHA in that it is only suitable as a message authentication code: a keyed hash function like HMAC.

CHAPTER 3

About RISCV

RISC-V is an open instruction set architecture (ISA) based on established reduced instruction set computing (RISC) principles.

In contrast to most ISAs, the RISC-V ISA can be freely used for any purpose, permitting anyone to design, manufacture and sell RISC-V chips and software. While not the first open ISA, it is significant because it is designed to be useful in modern computerized devices such as warehouse-scale cloud computers, high-end mobile phones and the smallest embedded systems. Such uses demand that the designers consider both performance and power efficiency. The instruction set also has a substantial body of supporting software, which avoids a usual weakness of new instruction setsWaterman *et al.* (2011).

The RISC-V ISA has been designed with small, fast, and low-power real-world implementations in mind,but without over-architecting for a particular microarchitecture style.

3.1 RISC-V software tools

The RISC-V GNU toolchain consists of GCC, Binutils, newlib and glibc ports.Available RISC-V software tools include a GNU Compiler Collection (GCC) toolchain (with GDB, the debugger), an LLVM toolchain, the Renode multi-node simulation framework, the OVPsim simulator (and library of RISC-V Fast Processor Models), the Spike simulator, and a simulator in QEMU. The RISC-V website has a specification for user-mode instructions, and a preliminary specification for a general-purpose privileged instruction set, to support operating systems. For this project we have exploited RISCV cross compiler and spike simulator.Spike is a RISC-V functional ISA simulator. It models a RISC-V core and cache system.

CHAPTER 4

Results

Certain popular cryptography algorithms are selected for this project.Their benchmark c codes are compiled RISC-V ELF binary using RISCV cross compiler.Used objdump to disassemble the .o file and annotate it with the C source file.This gives static assembly of program.4.1

```
Disassembly of section .text:

00000000800000f2 <blowfish_encrypt>:
    800000f2:   7139                    addi    sp,sp,-64
    800000f4:   fc22                    sd      s0,56(sp)
    800000f6:   0080                    addi    s0,sp,64
    800000f8:   fca43c23                sd      a0,-40(s0)
    800000fc:   fcb43823                sd      a1,-48(s0)
    80000100:   fcc43423                sd      a2,-56(s0)
    80000104:   fd843783                ld      a5,-40(s0)
    80000108:   0007c783                lbu     a5,0(a5) # 0 <_start-0x80000000>
    8000010c:   2781                    sext.w  a5,a5
    8000010e:   0187979b                slliw   a5,a5,0x18
    80000112:   0007871b                sext.w  a4,a5
    80000116:   fd843783                ld      a5,-40(s0)
    8000011a:   0785                    addi    a5,a5,1
    8000011c:   0007c783                lbu     a5,0(a5)
    80000120:   2781                    sext.w  a5,a5
    80000122:   0107979b                slliw   a5,a5,0x10
    80000126:   2781                    sext.w  a5,a5
    80000128:   8fd9                    or      a5,a5,a4
    8000012a:   0007871b                sext.w  a4,a5
    8000012e:   fd843783                ld      a5,-40(s0)
    80000132:   0789                    addi    a5,a5,2
    80000134:   0007c783                lbu     a5,0(a5)
    80000138:   2781                    sext.w  a5,a5
    8000013a:   0087979b                slliw   a5,a5,0x8
    8000013e:   2781                    sext.w  a5,a5
```

Figure 4.1: snippet of disassembled file of blowfish algorithm

The command-line arguments to Spike are listed below4.2

```
usage: spike [host options] <target program> [target options]
Host Options:
  -p<n>                 Simulate <n> processors [default 1]
  -m<n>                 Provide <n> MiB of target memory [default 2048]
  -m<a:m,b:n,...>       Provide memory regions of size m and n bytes
                          at base addresses a and b (with 4 KiB alignment)
  -d                    Interactive debug mode
  -g                    Track histogram of PCs
  -l                    Generate a log of execution
  -h                    Print this help message
  -H                    Start halted, allowing a debugger to connect
  --isa=<name>          RISC-V ISA string [default RV64IMAFDC]
  --pc=<address>        Override ELF entry point
  --hartids=<a,b,...>   Explicitly specify hartids, default is 0,1,...
  --ic=<S>:<W>:<B>      Instantiate a cache model with S sets,
  --dc=<S>:<W>:<B>        W ways, and B-byte blocks (with S and
  --l2=<S>:<W>:<B>        B both powers of 2).
  --extension=<name>    Specify RoCC Extension
  --extlib=<name>       Shared library to load
  --rbb-port=<port>     Listen on <port> for remote bitbang connection
  --dump-dts            Print device tree string and exit
  --progsize=<words>    progsize for the debug module [default 2]
```

Figure 4.2: Command line arguments to Spike simulator

Using -l argument of Spike, pc log of execution is obtained 4.3

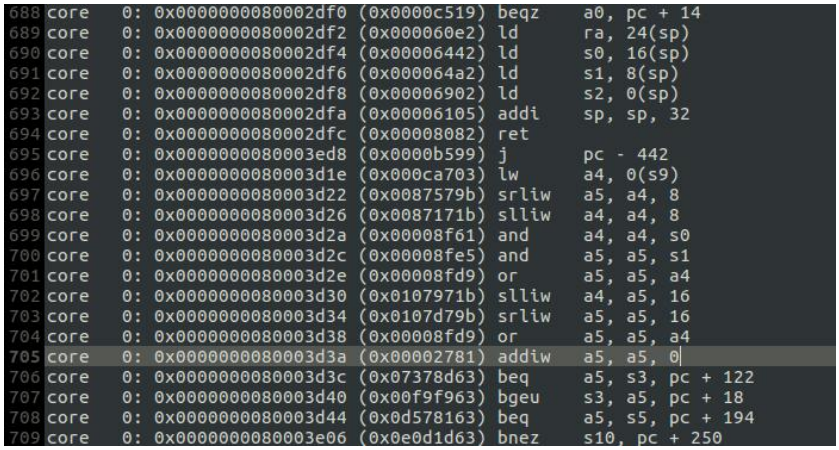


Figure 4.3: snippet of pc log of execution of blowfish algorithm

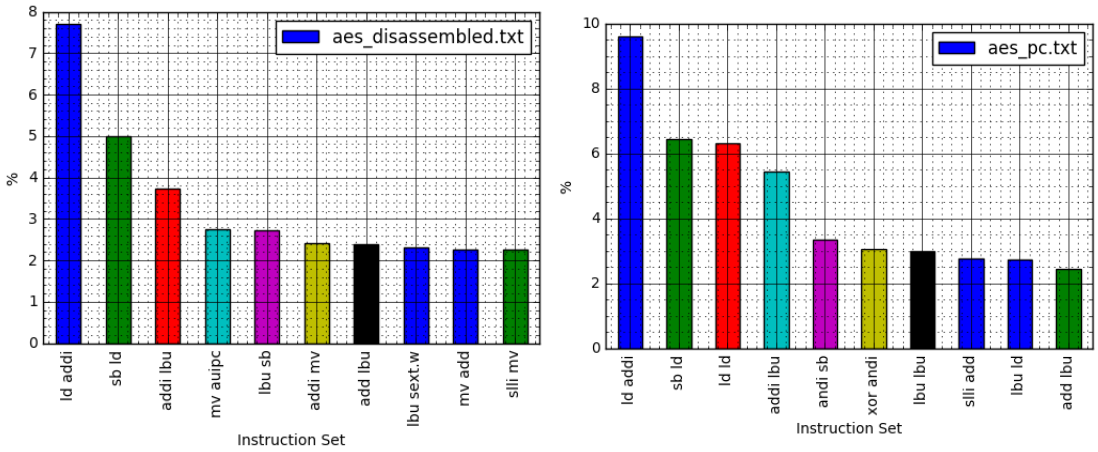
Python script is written to take simulation output and find frequency of set of instructions. Using python script to traverse through log files of cryptography algorithms, common set of consecutive instructions are found.This gives us total percentage of execution time taken by set of instructions. Analysis of disassembled code gives percentage of memory taken by set of instructions and analysis of log of execution files gives percentage of execution time taken by set of instructions.

Two cases are considered, first we have considered only opcodes of instructions.Secondly operands are also taken into account to find dependent consecutive instructions.Focus was on instruction set starting with load operation and ending with store operation.

Histogram of top 10 such cases are categorized accordiing to type of cryptography and also overall are shown below:

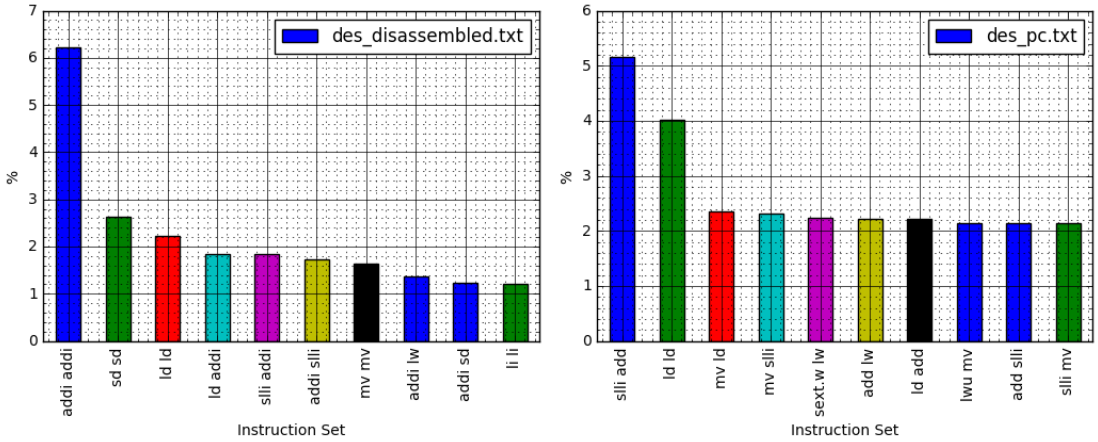
4.1 Analysis of all algorithms

4.1.1 For Set of two insturctions



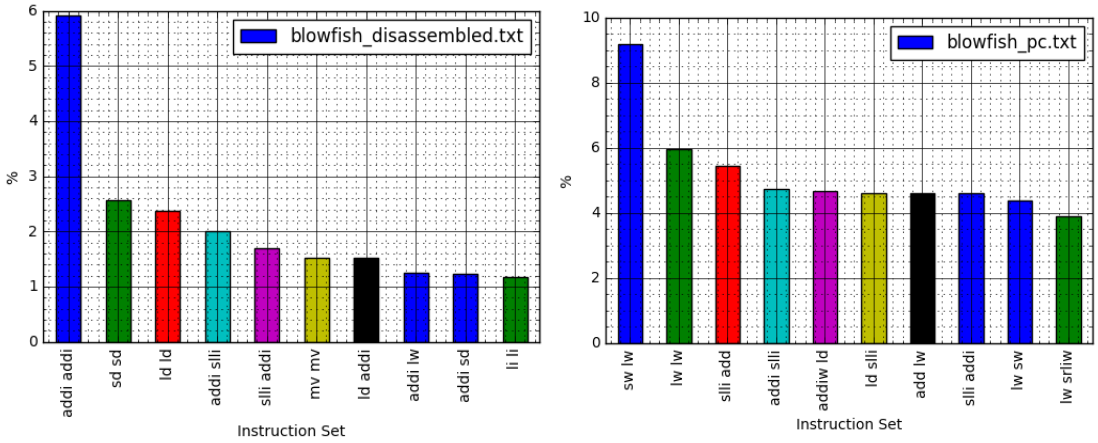
(a) Static (b) Runtime

Figure 4.4: AES:percentage of set of two instructions in terms of memory(static) and execution time(runtime)



(a) Static (b) Runtime

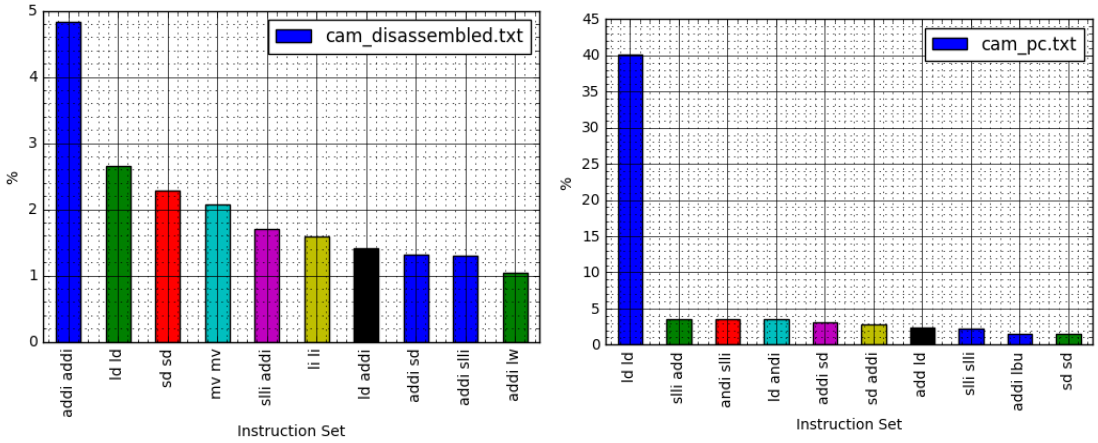
Figure 4.5: DES:percentage of set of two instructions in terms of memory(static) and execution time(runtime)



(a) Static

(b) Runtime

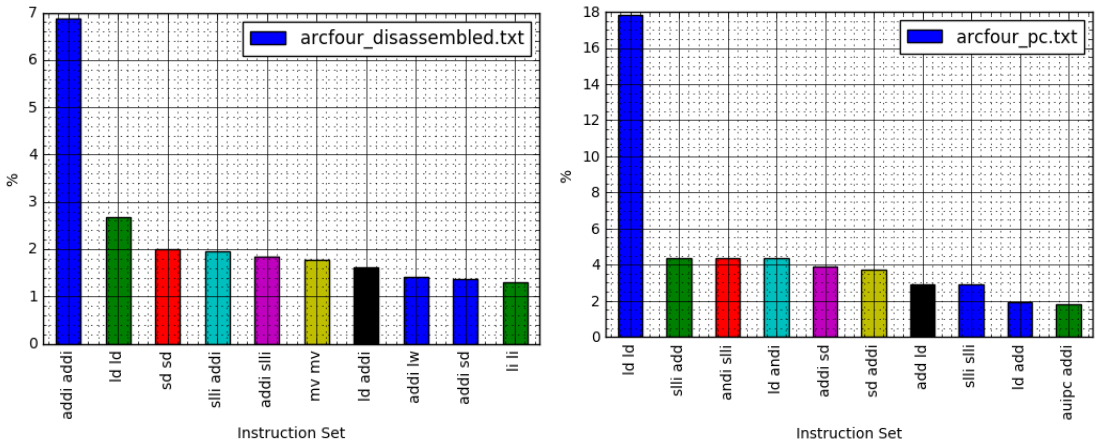
Figure 4.6: BLOWFISH:percentage of set of two instructions in terms of memory(static) and execution time(runtime)



(a) Static

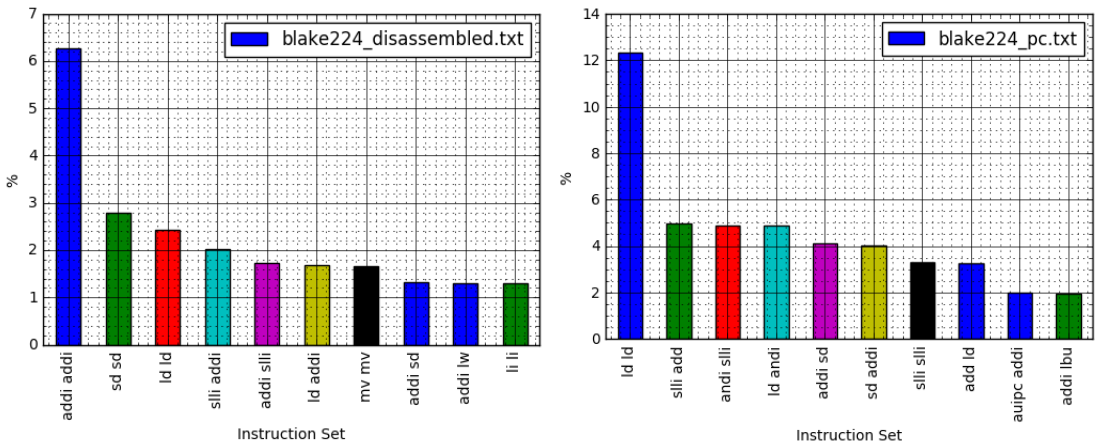
(b) Runtime

Figure 4.7: CAMELLIA:percentage of set of two instructions in terms of memory(static) and execution time(runtime)



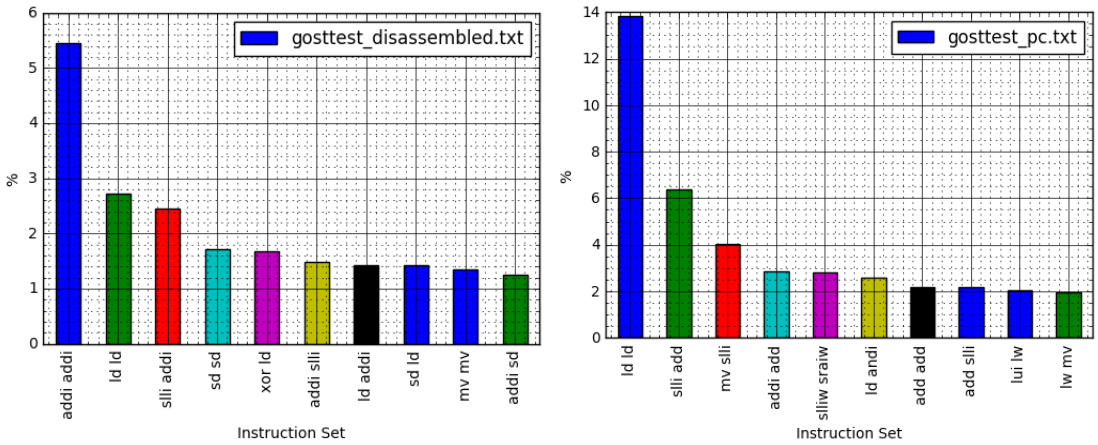
(a) Static (b) Runtime

Figure 4.8: ARCFOUR:percentage of set of two instructions in terms of memory(static) and execution time(runtime)



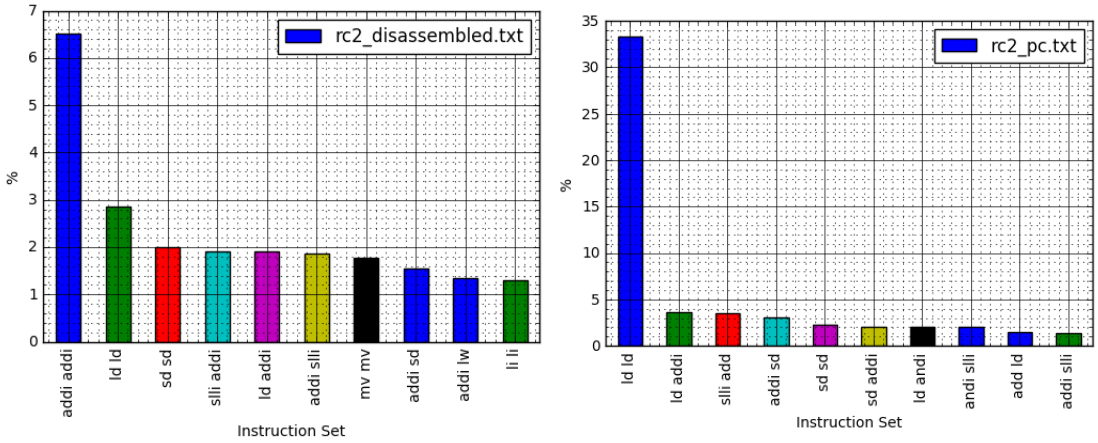
(a) Static (b) Runtime

Figure 4.9: BLAKE224:percentage of set of two instructions in terms of memory(static) and execution time(runtime)



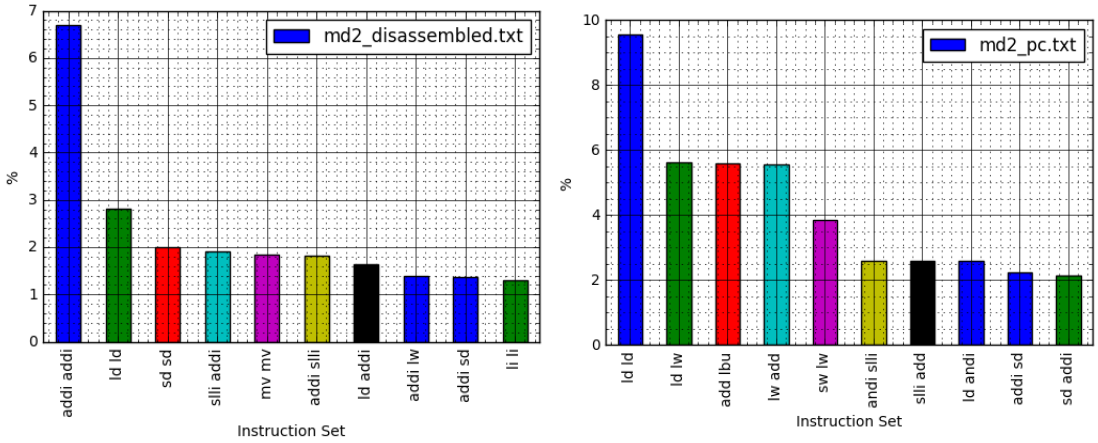
(a) Static (b) Runtime

Figure 4.10: GOST:percentage of set of two instructions in terms of memory(static) and execution time(runtime)



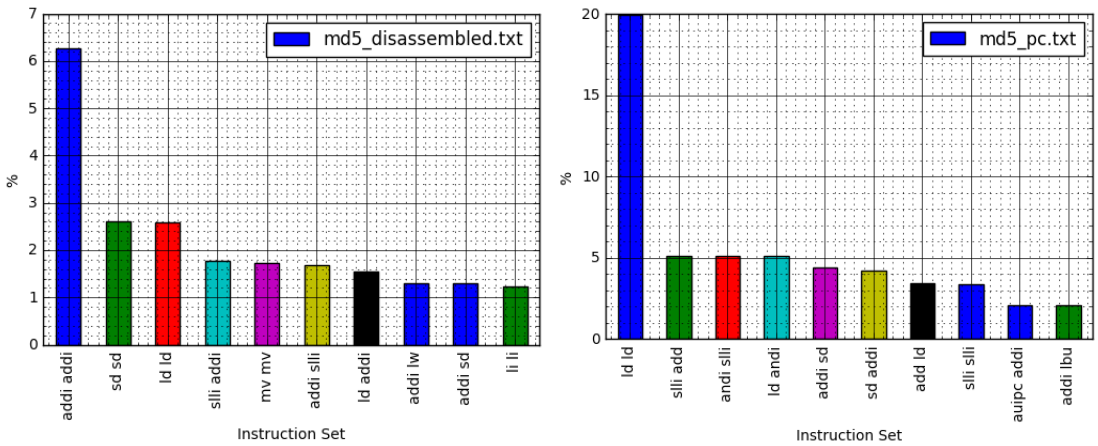
(a) Static (b) Runtime

Figure 4.11: RC2:percentage of set of two instructions in terms of memory(static) and execution time(runtime)



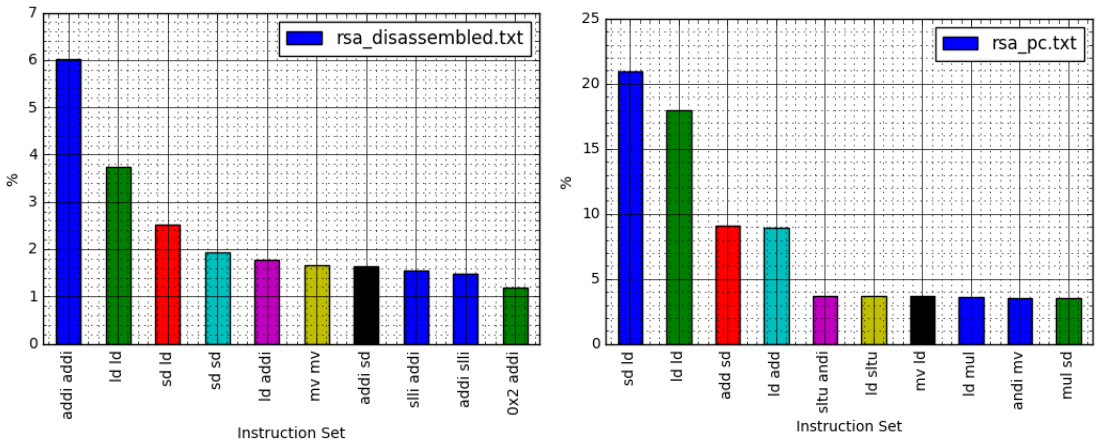
(a) Static (b) Runtime

Figure 4.12: MD2:percentage of set of two instructions in terms of memory(static) and execution time(runtime)



(a) Static (b) Runtime

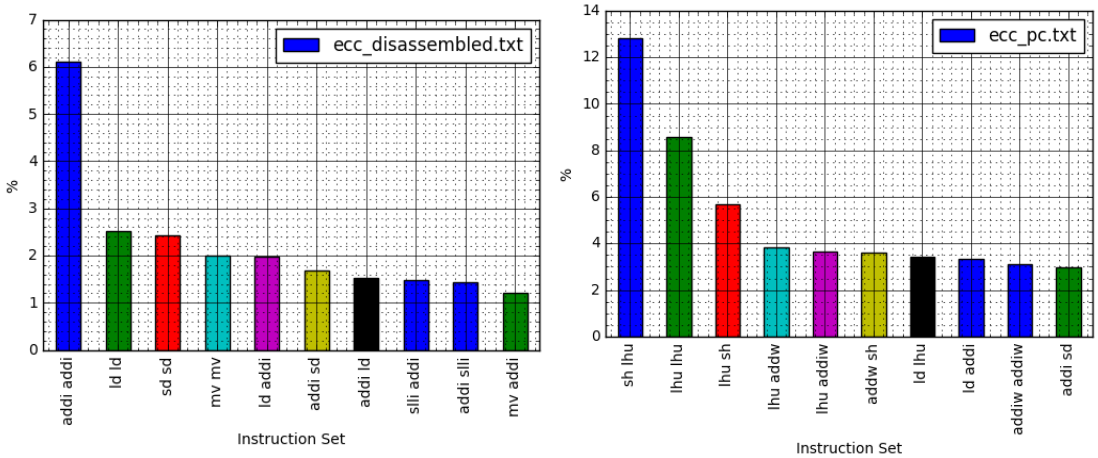
Figure 4.13: MD5:percentage of set of two instructions in terms of memory(static) and execution time(runtime)



(a) Static

(b) Runtime

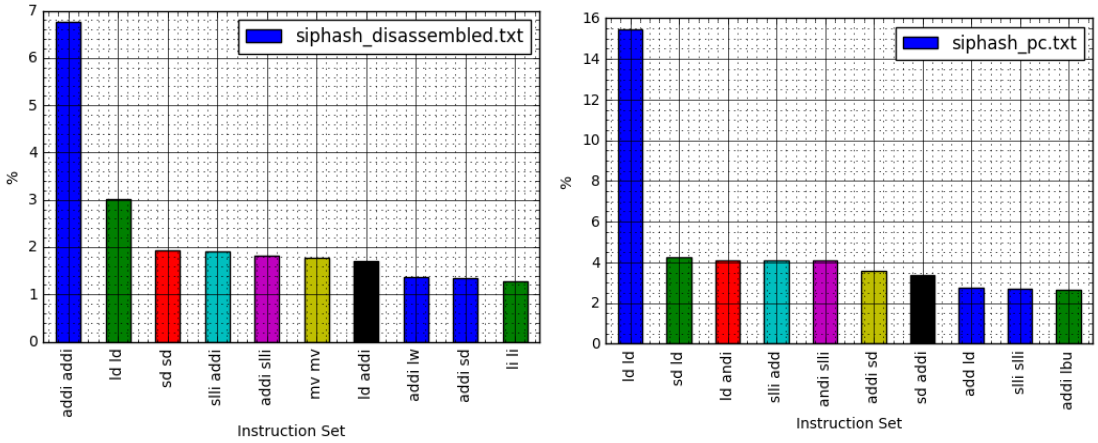
Figure 4.14: RSA:percentage of set of two instructions in terms of memory(static) and execution time(runtime)



(a) Static

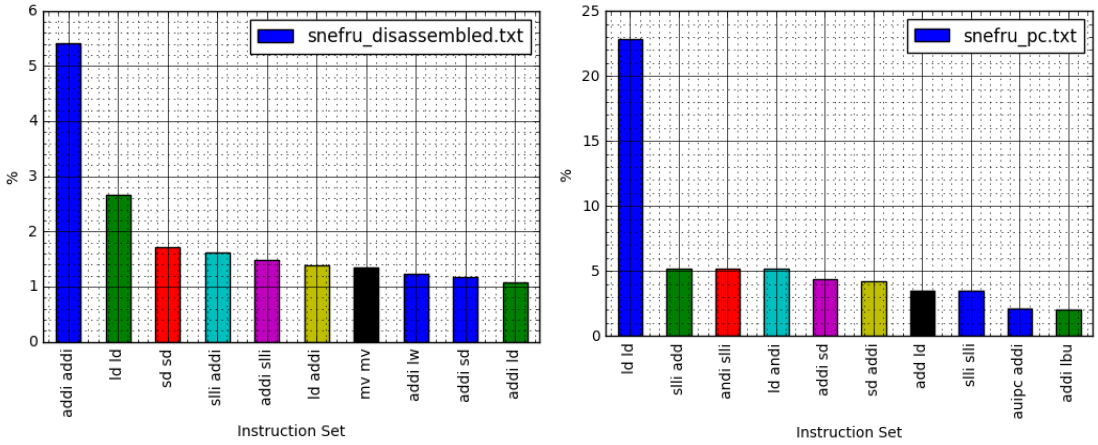
(b) Runtime

Figure 4.15: ECC:percentage of set of two instructions in terms of memory(static) and execution time(runtime)



(a) Static (b) Runtime

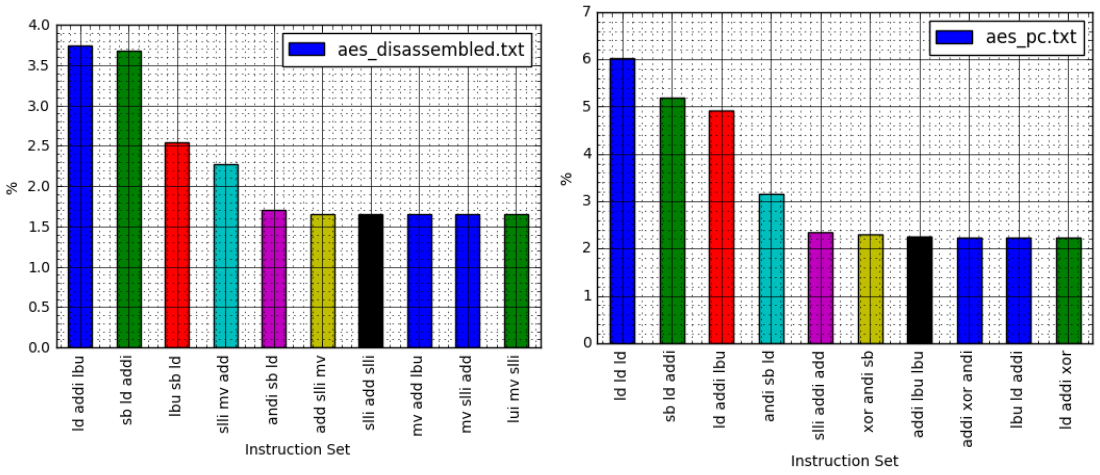
Figure 4.16: SIPHASH:percentage of set of two instructions in terms of memory(static) and execution time(runtime)



(a) Static (b) Runtime

Figure 4.17: SNEFRU:percentage of set of two instructions in terms of memory(static) and execution time(runtime)

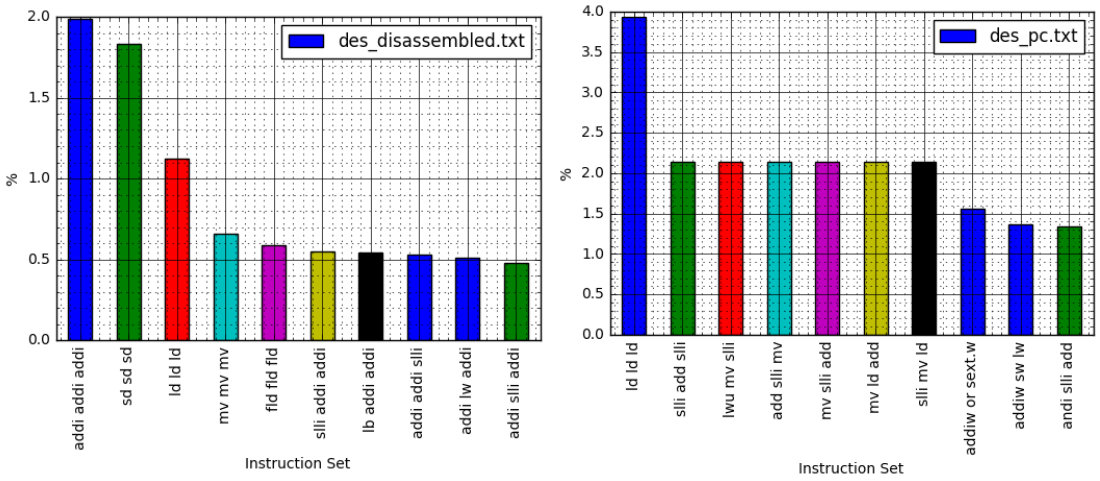
4.1.2 For Set of three insturctions



(a) Static

(b) Runtime

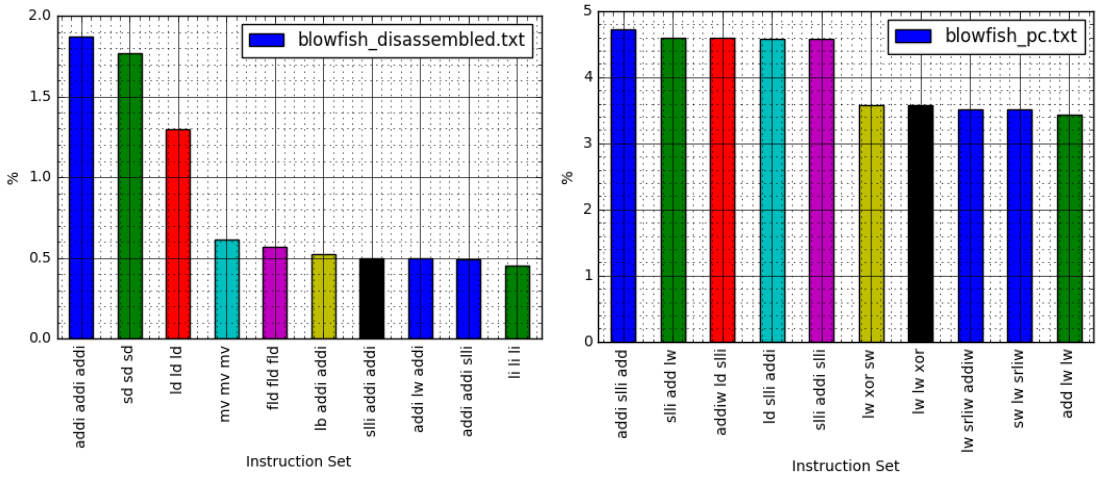
Figure 4.18: AES:percentage of set of three instructions in terms of memory(static) and execution time(runtime)



(a) Static

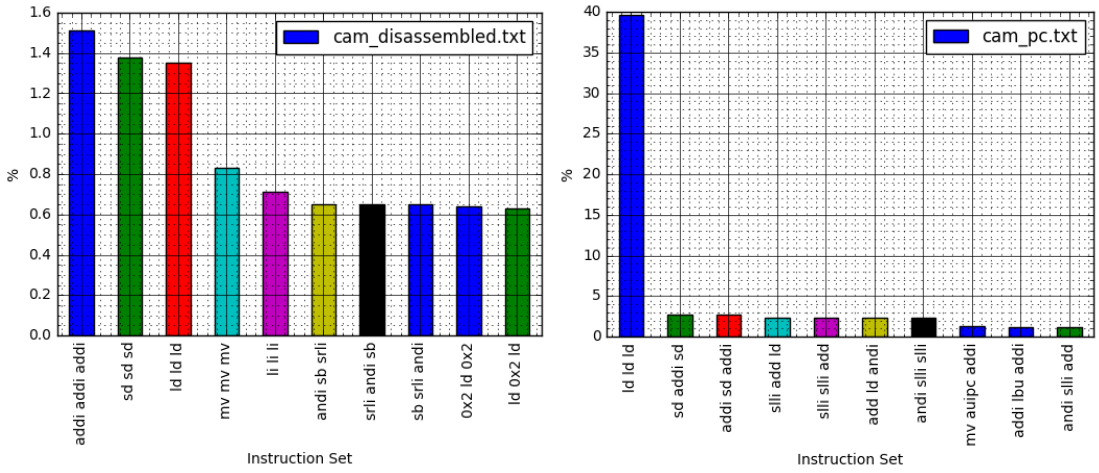
(b) Runtime

Figure 4.19: DES:percentage of set of three instructions in terms of memory(static) and execution time(runtime)



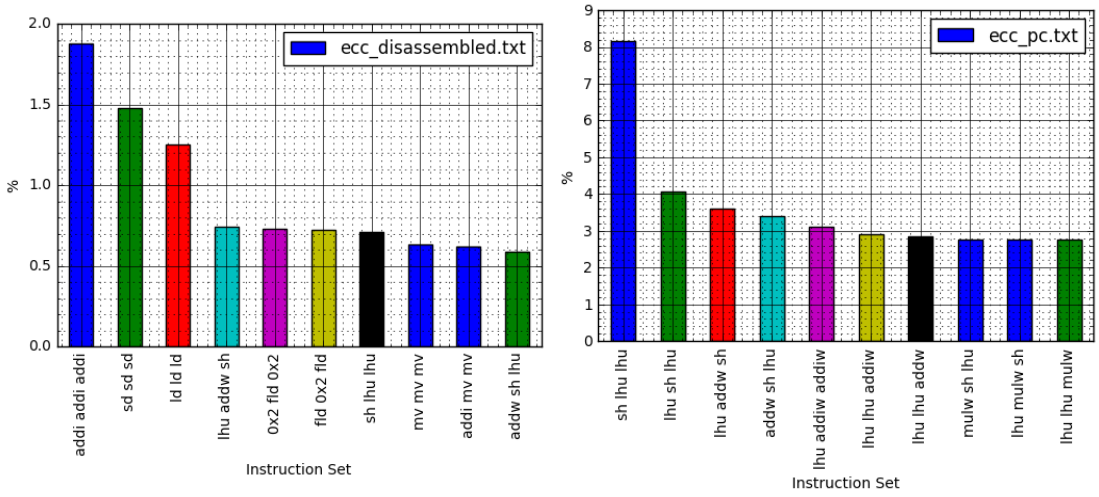
(a) Static (b) Runtime

Figure 4.20: BLOWFISH:percentage of set of three instructions in terms of memory(static) and execution time(runtime)



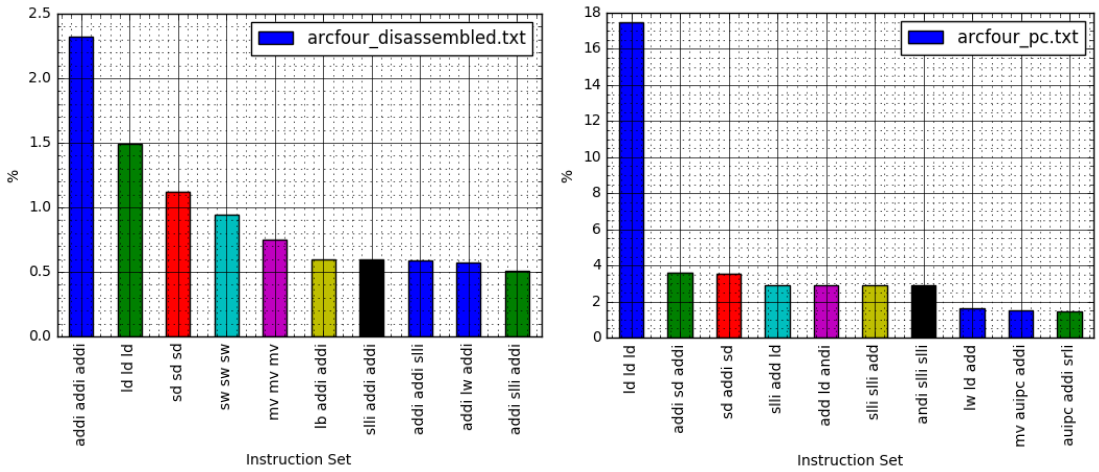
(a) Static (b) Runtime

Figure 4.21: CAMELLIA:percentage of set of three instructions in terms of memory(static) and execution time(runtime)



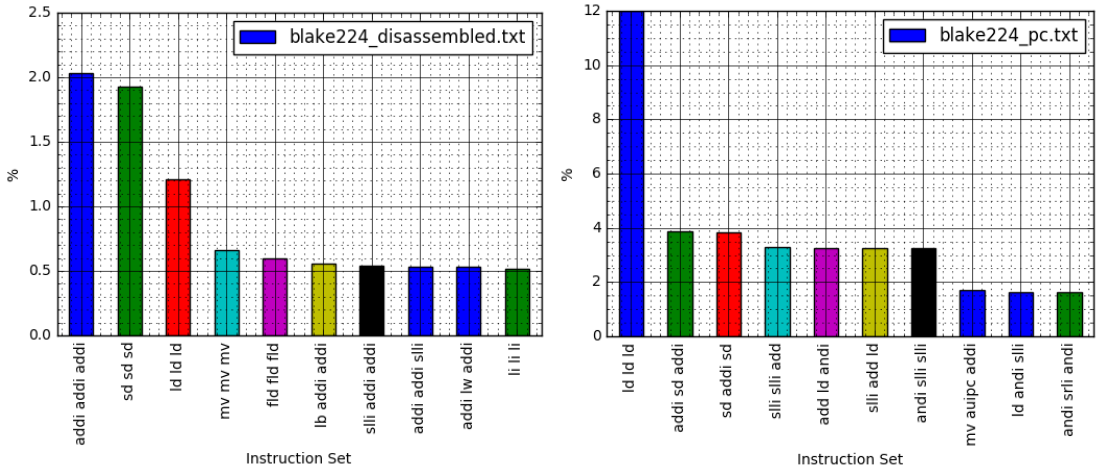
(a) Static (b) Runtime

Figure 4.29: ECC:percentage of set of three instructions in terms of memory(static) and execution time(runtime)



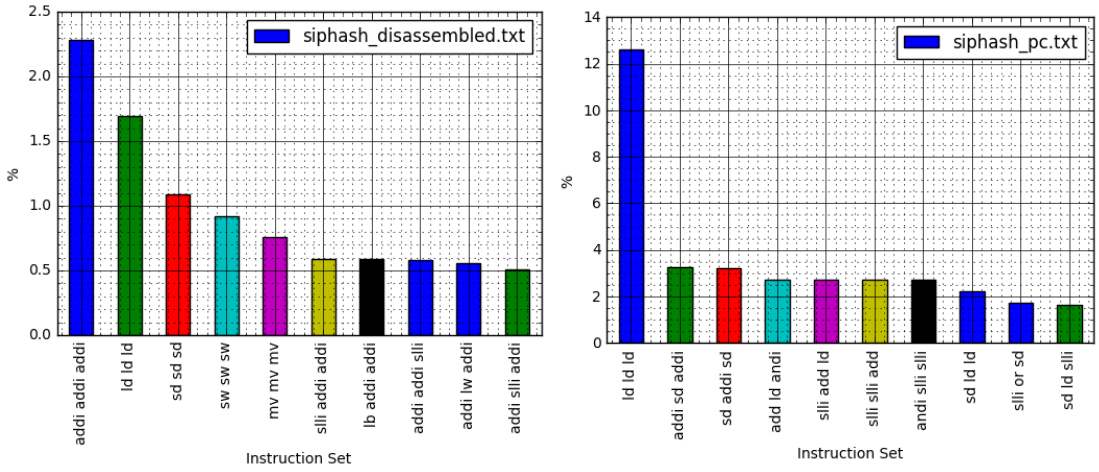
(a) Static (b) Runtime

Figure 4.22: ARCFOUR:percentage of set of three instructions in terms of memory(static) and execution time(runtime)



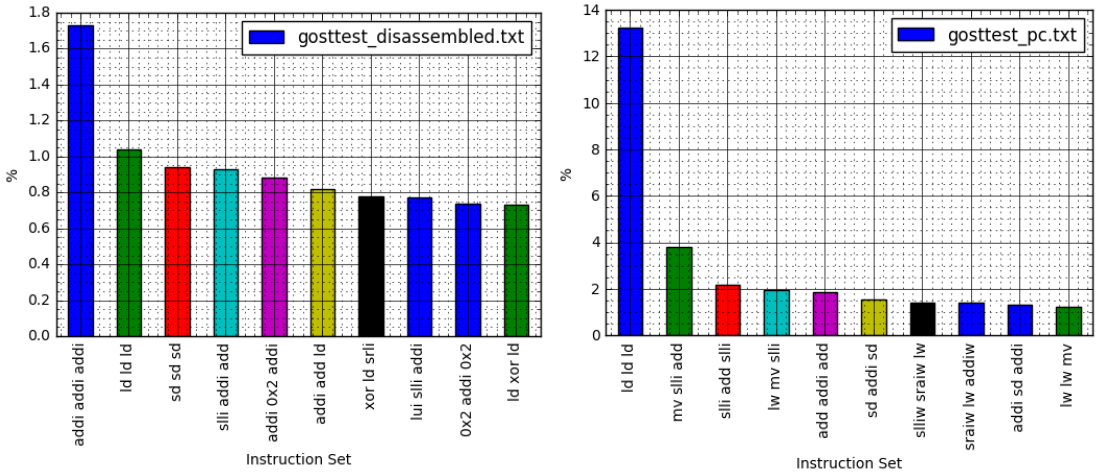
(a) Static (b) Runtime

Figure 4.23: BLAKE224:percentage of set of three instructions in terms of memory(static) and execution time(runtime)



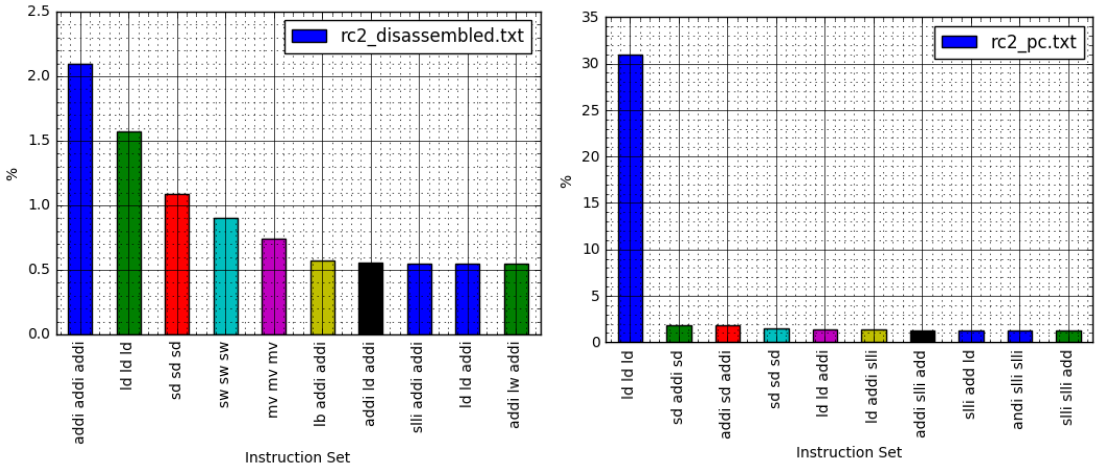
(a) Static (b) Runtime

Figure 4.30: SIPHASH:percentage of set of three instructions in terms of memory(static) and execution time(runtime)



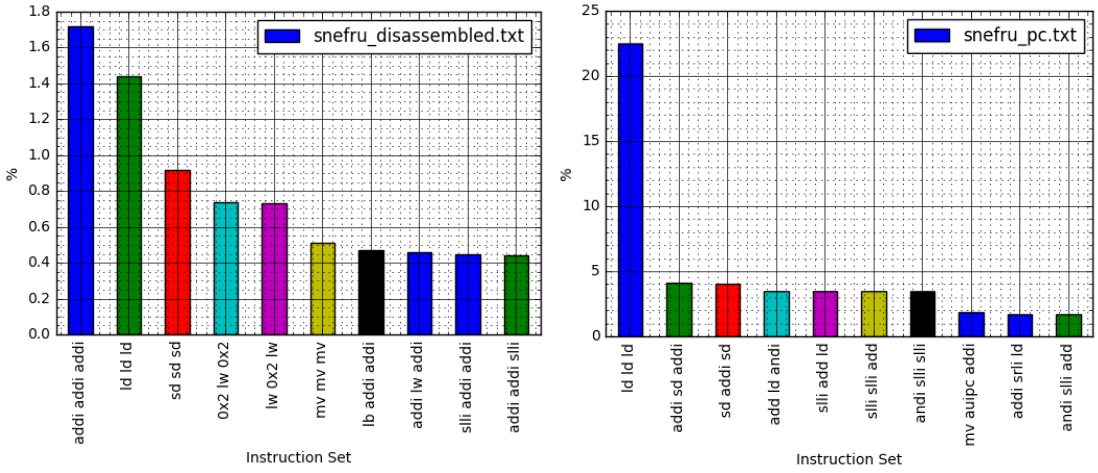
(a) Static (b) Runtime

Figure 4.24: GOST:percentage of set of three instructions in terms of memory(static) and execution time(runtime)



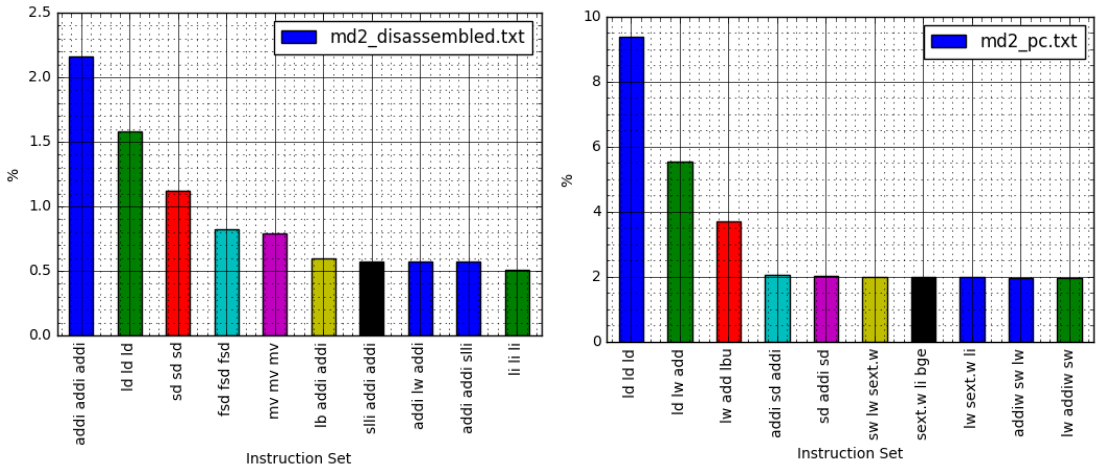
(a) Static (b) Runtime

Figure 4.25: RC2:percentage of set of three instructions in terms of memory(static) and execution time(runtime)



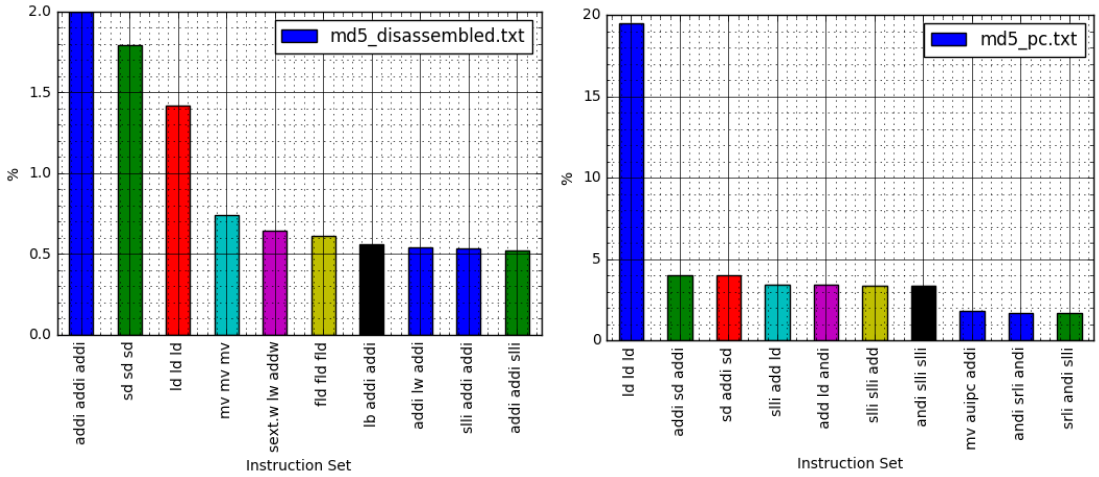
(a) Static (b) Runtime

Figure 4.31: SNEFRU:percentage of set of three instructions in terms of memory(static) and execution time(runtime)



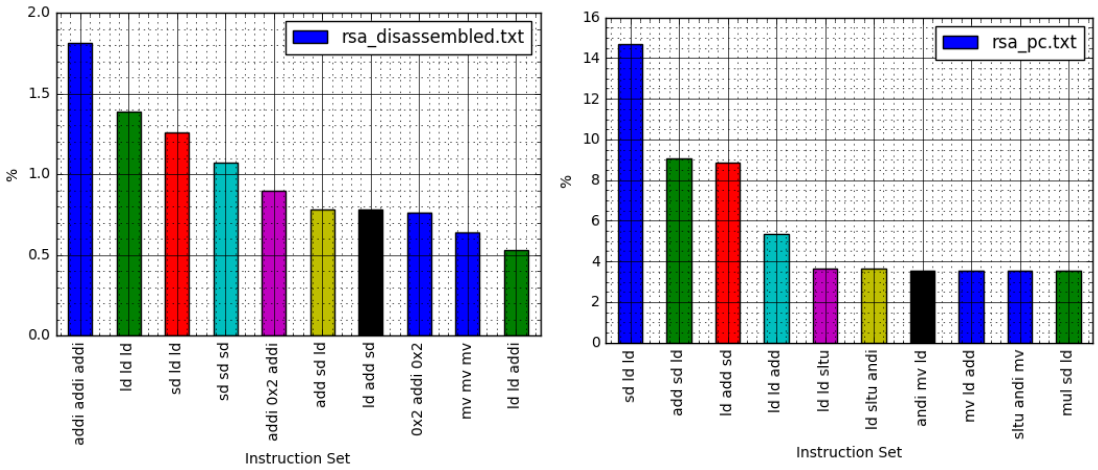
(a) Static (b) Runtime

Figure 4.26: MD2:percentage of set of three instructions in terms of memory(static) and execution time(runtime)



(a) Static (b) Runtime

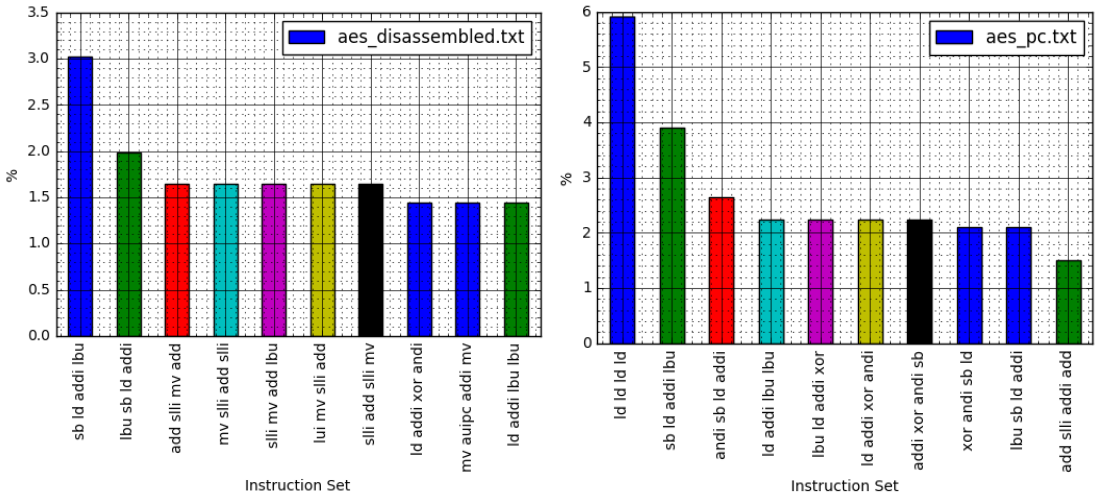
Figure 4.27: MD5:percentage of set of three instructions in terms of memory(static) and execution time(runtime)



(a) Static (b) Runtime

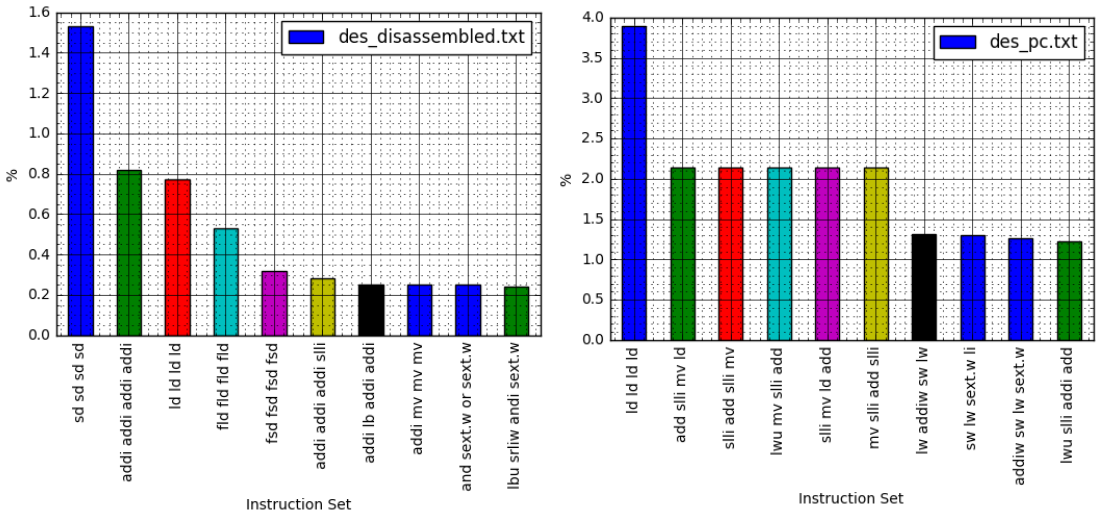
Figure 4.28: RSA:percentage of set of three instructions in terms of memory(static) and execution time(runtime)

4.1.3 For Set of four insturctions



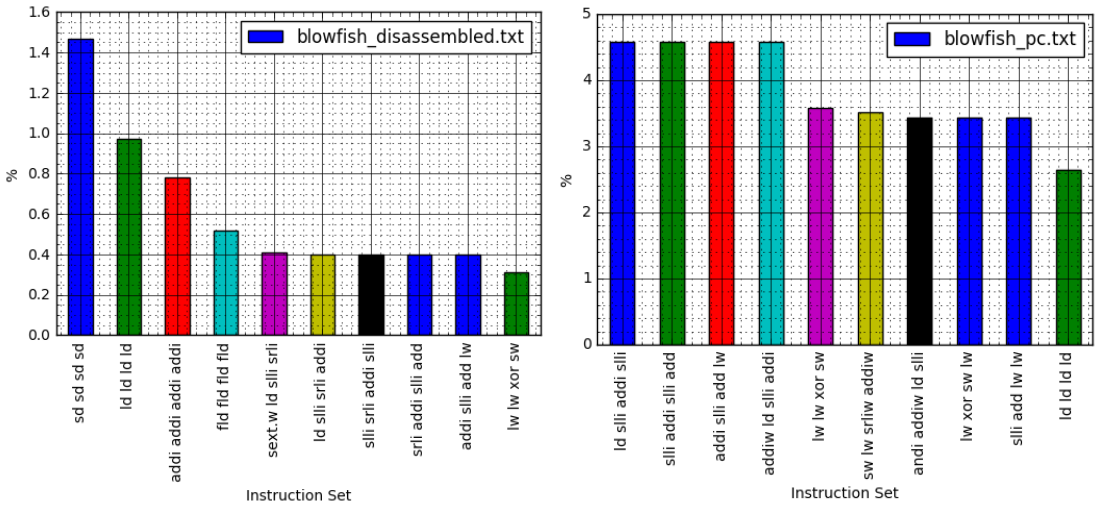
(a) Static (b) Runtime

Figure 4.32: AES:percentage of set of four instructions in terms of memory(static) and execution time(runtime)



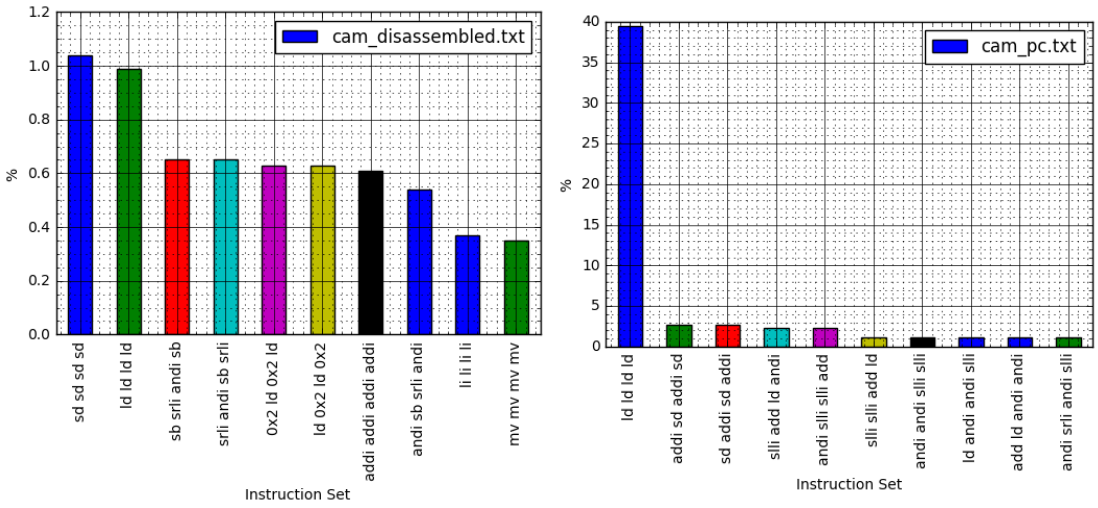
(a) Static (b) Runtime

Figure 4.33: DES:percentage of set of four instructions in terms of memory(static) and execution time(runtime)



(a) Static (b) Runtime

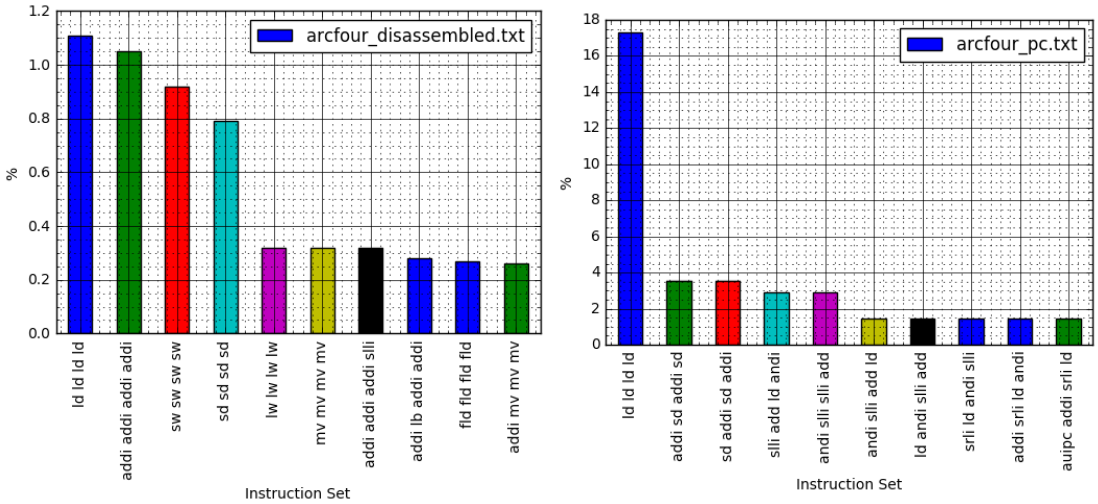
Figure 4.34: BLOWFISH:percentage of set of four instructions in terms of memory(static) and execution time(runtime)



(a) Static

(b) Runtime

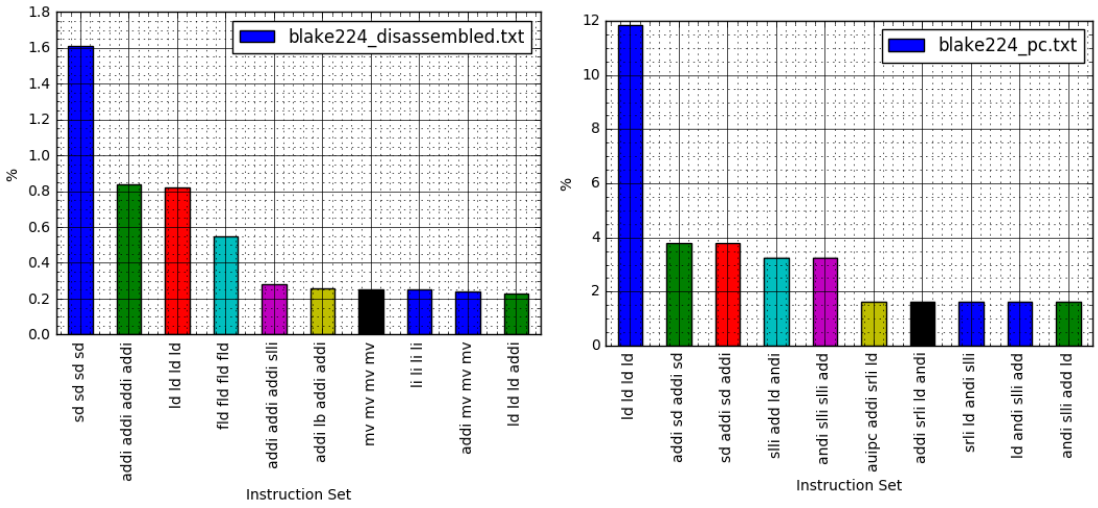
Figure 4.35: CAMELLIA:percentage of set of four instructions in terms of memory(static) and execution time(runtime)



(a) Static

(b) Runtime

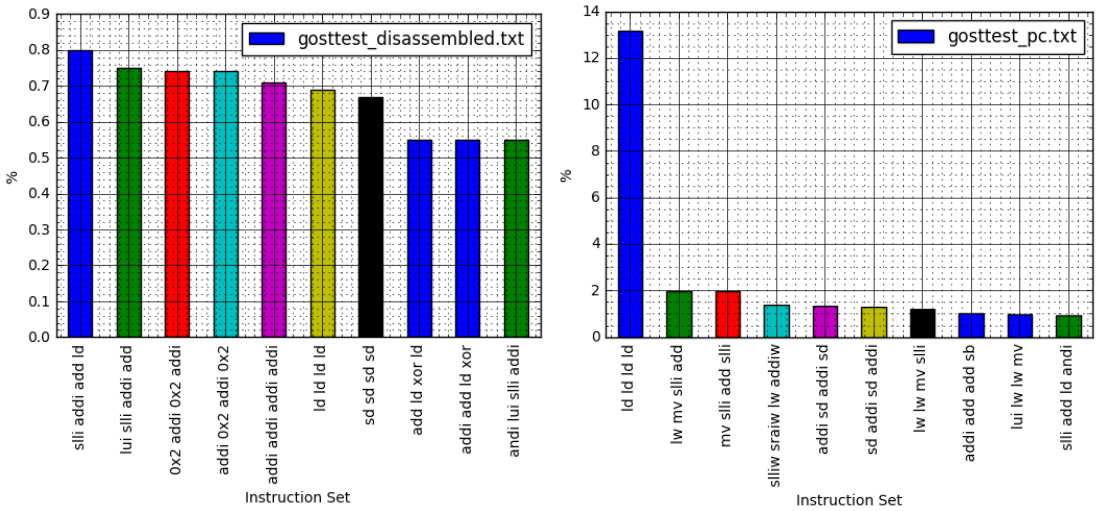
Figure 4.36: ARCFOUR:percentage of set of four instructions in terms of memory(static) and execution time(runtime)



(a) Static

(b) Runtime

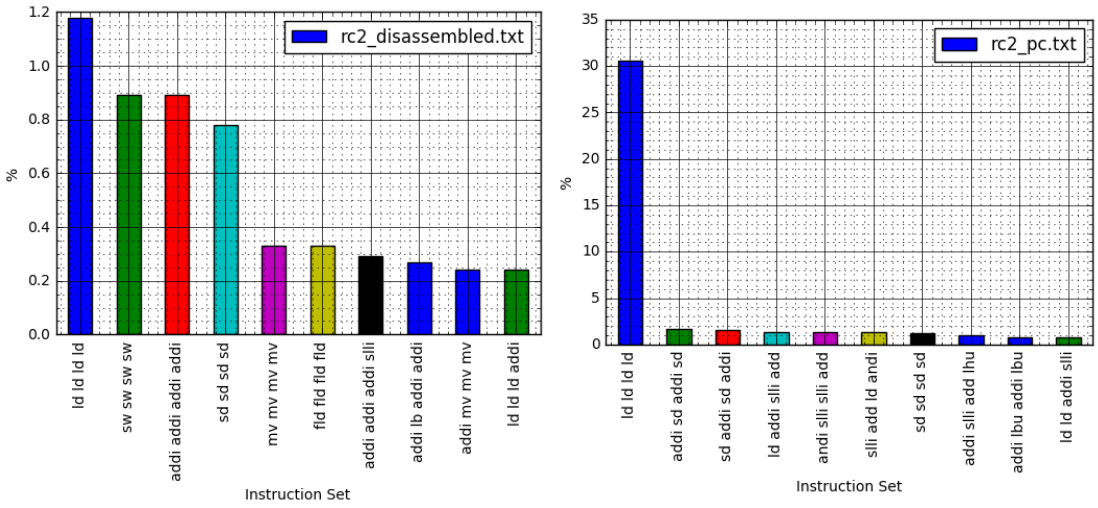
Figure 4.37: BLAKE224:percentage of set of four instructions in terms of memory(static) and execution time(runtime)



(a) Static

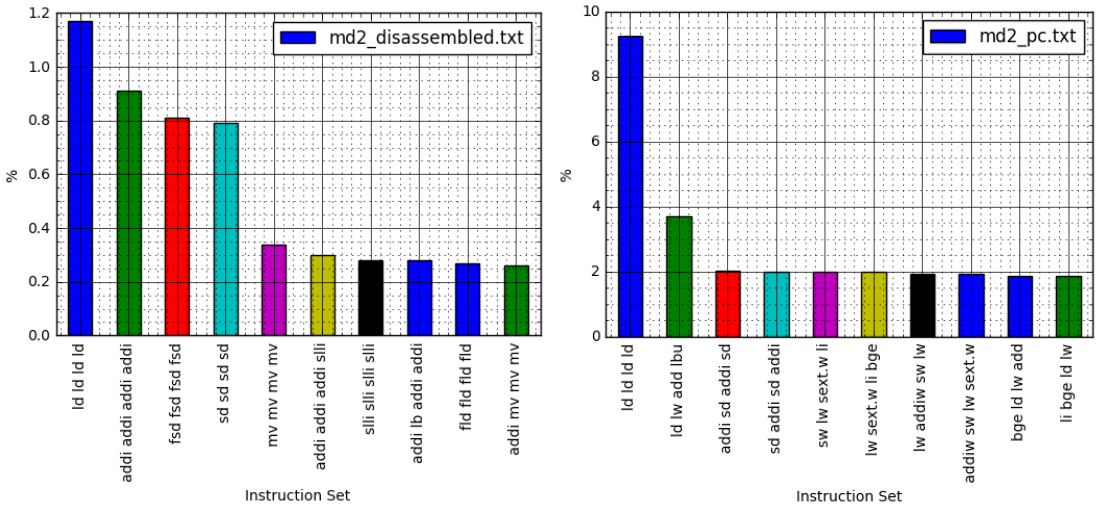
(b) Runtime

Figure 4.38: GOST:percentage of set of four instructions in terms of memory(static) and execution time(runtime)



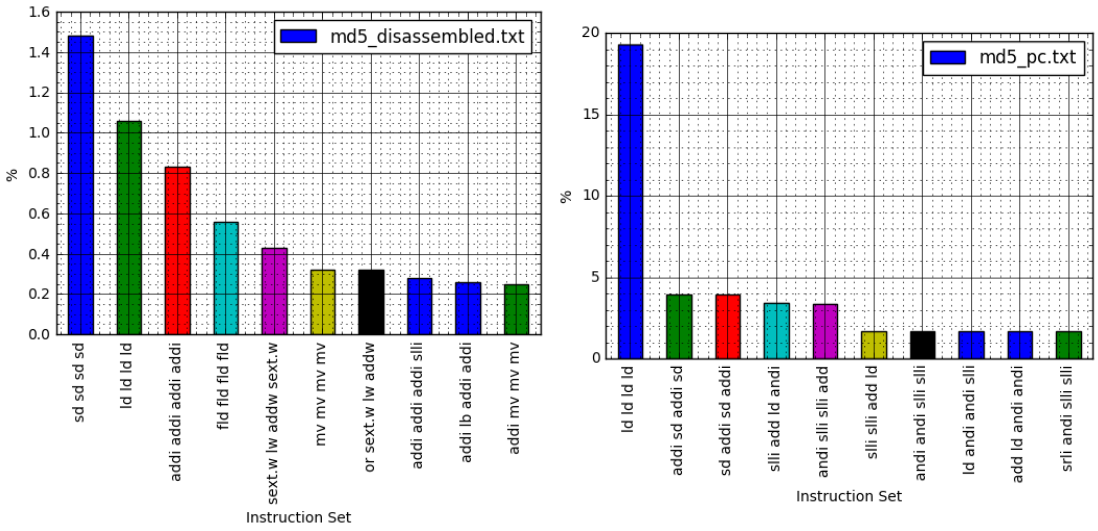
(a) Static (b) Runtime

Figure 4.39: RC2:percentage of set of four instructions in terms of memory(static) and execution time(runtime)



(a) Static (b) Runtime

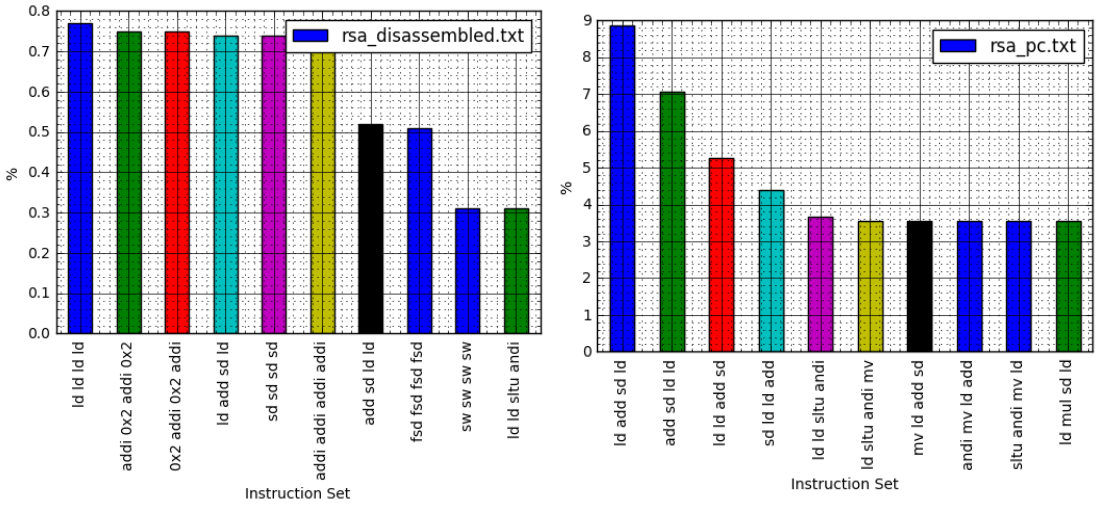
Figure 4.40: MD2:percentage of set of four instructions in terms of memory(static) and execution time(runtime)



(a) Static

(b) Runtime

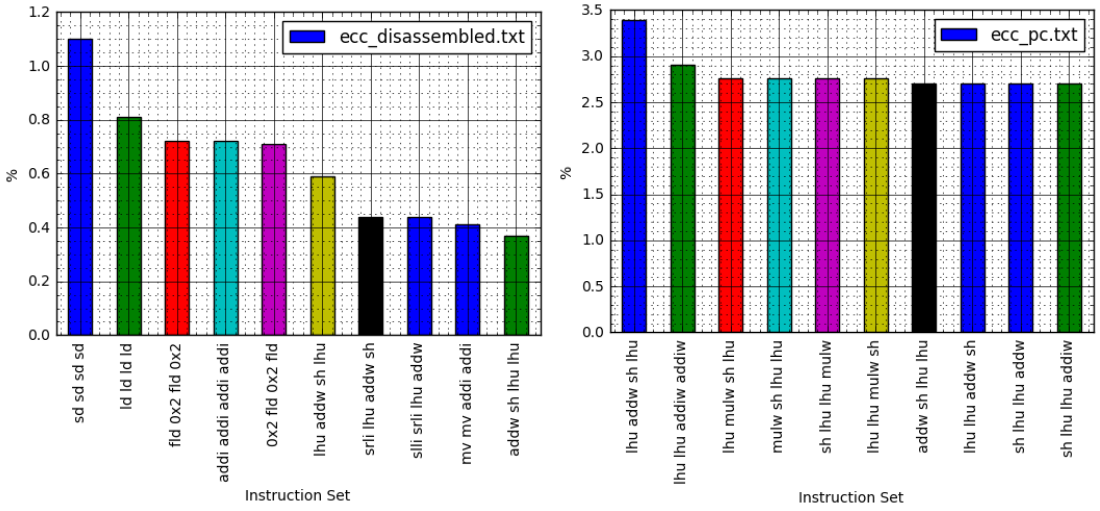
Figure 4.41: MD5:percentage of set of four instructions in terms of memory(static) and execution time(runtime)



(a) Static

(b) Runtime

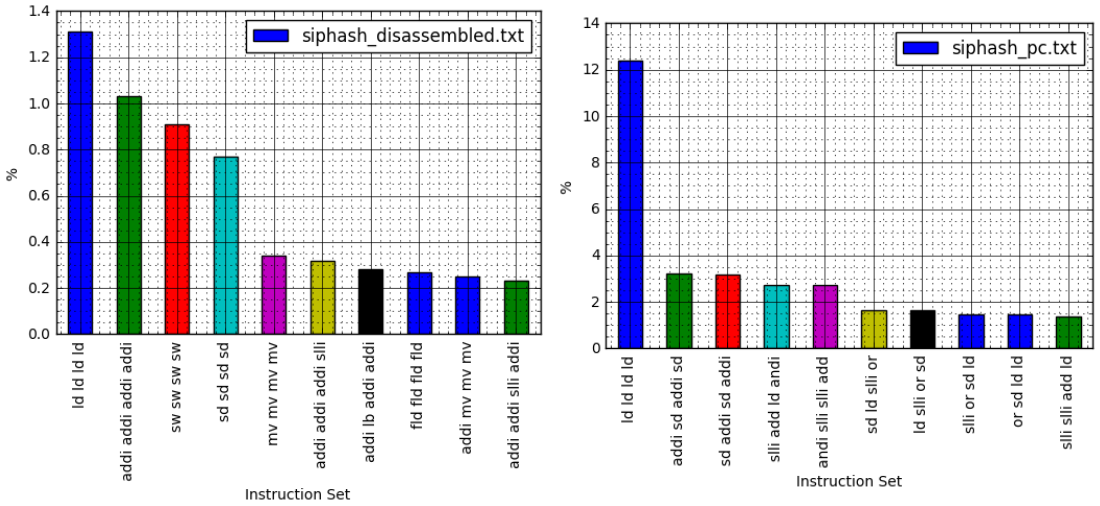
Figure 4.42: RSA:percentage of set of four instructions in terms of memory(static) and execution time(runtime)



(a) Static

(b) Runtime

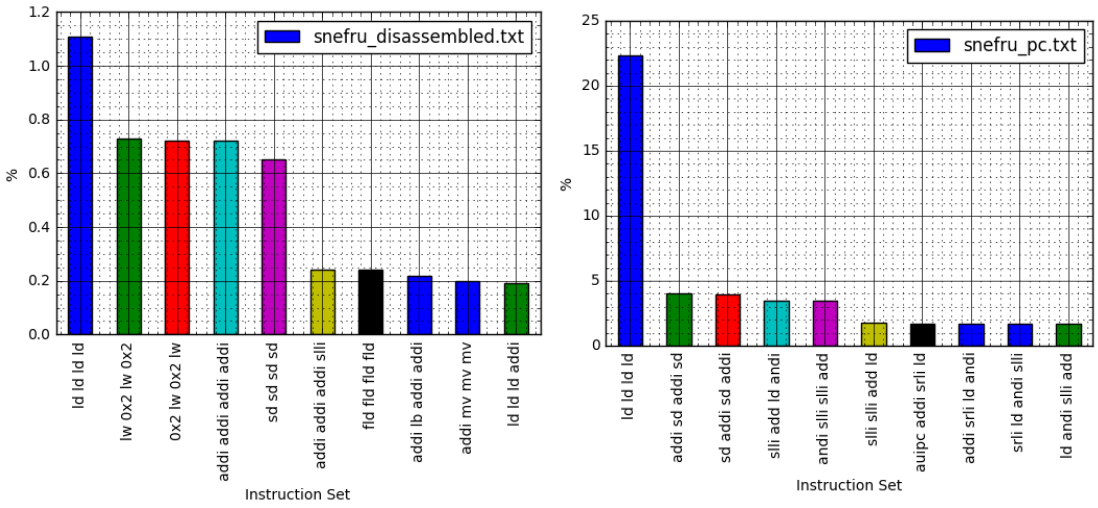
Figure 4.43: ECC:percentage of set of four instructions in terms of memory(static) and execution time(runtime)



(a) Static

(b) Runtime

Figure 4.44: SIPHASH:percentage of set of four instructions in terms of memory(static) and execution time(runtime)

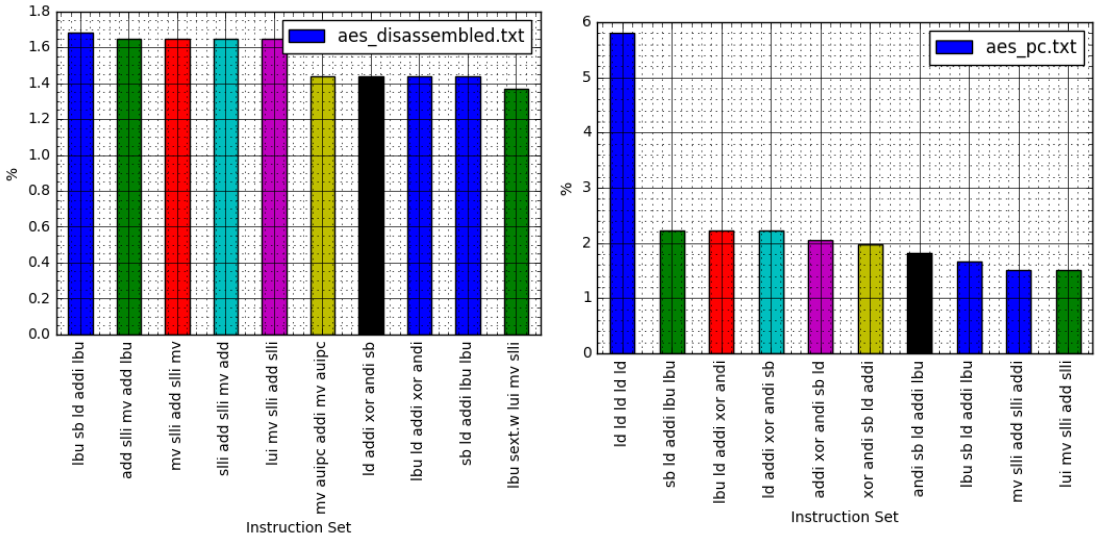


(a) Static

(b) Runtime

Figure 4.45: SNEFRU:percentage of set of four instructions in terms of memory(static) and execution time(runtime)

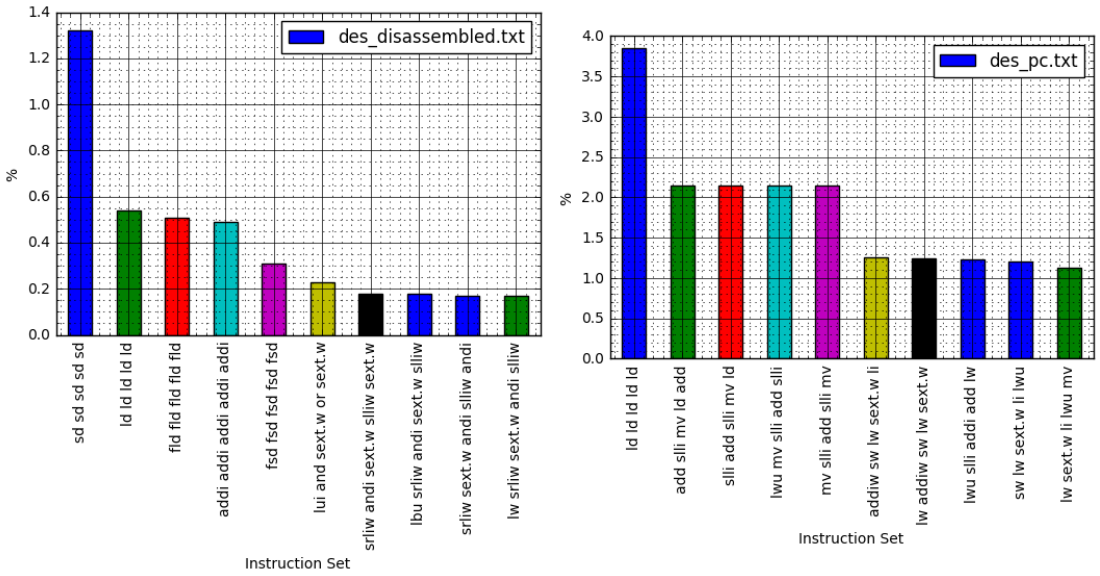
4.1.4 For Set of five insturctions



(a) Static

(b) Runtime

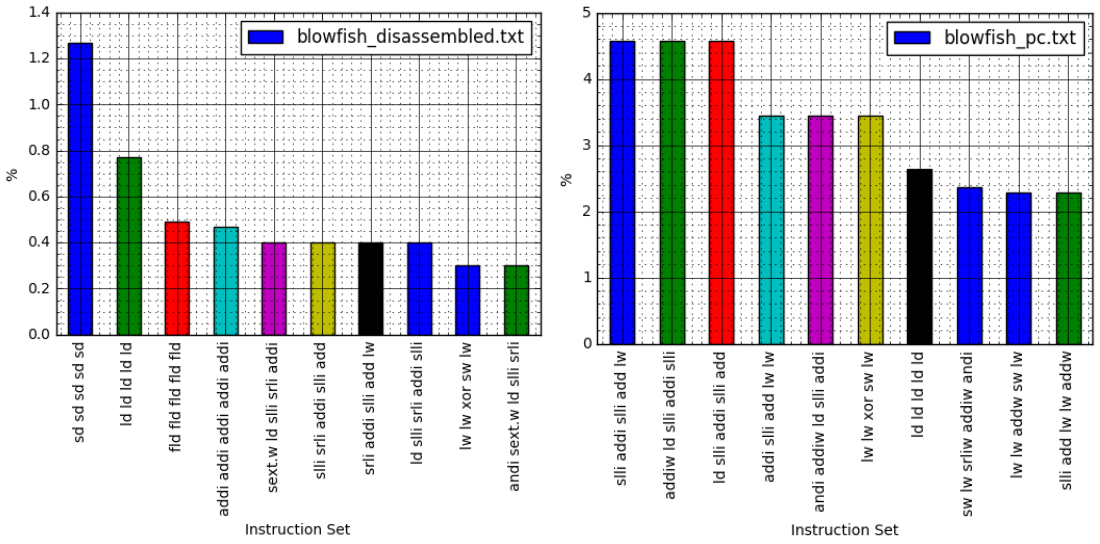
Figure 4.46: AES:percentage of set of five instructions in terms of memory(static) and execution time(runtime)



(a) Static

(b) Runtime

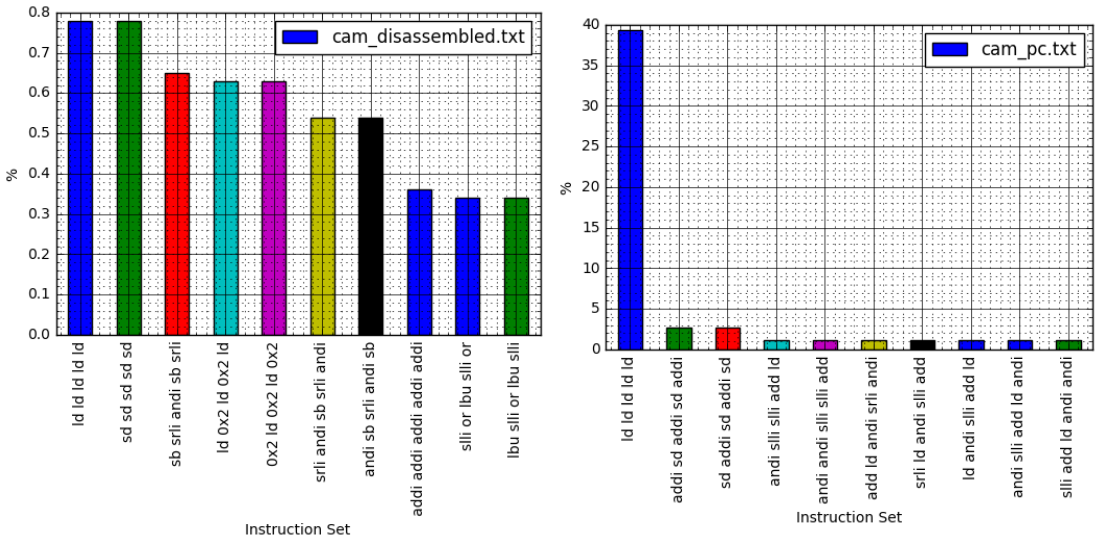
Figure 4.47: DES:percentage of set of five instructions in terms of memory(static) and execution time(runtime)



(a) Static

(b) Runtime

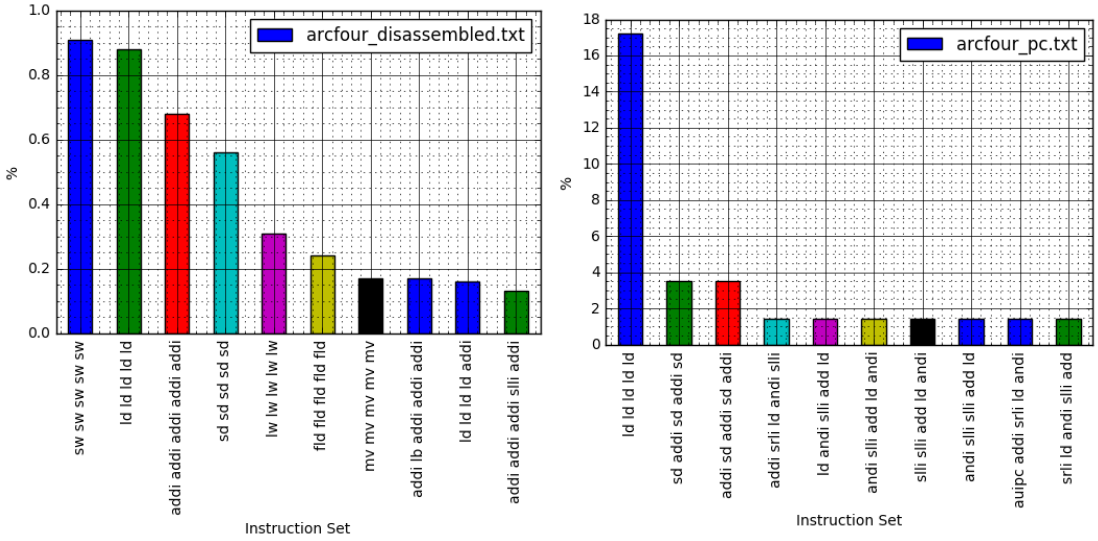
Figure 4.48: BLOWFISH:percentage of set of five instructions in terms of memory(static) and execution time(runtime)



(a) Static

(b) Runtime

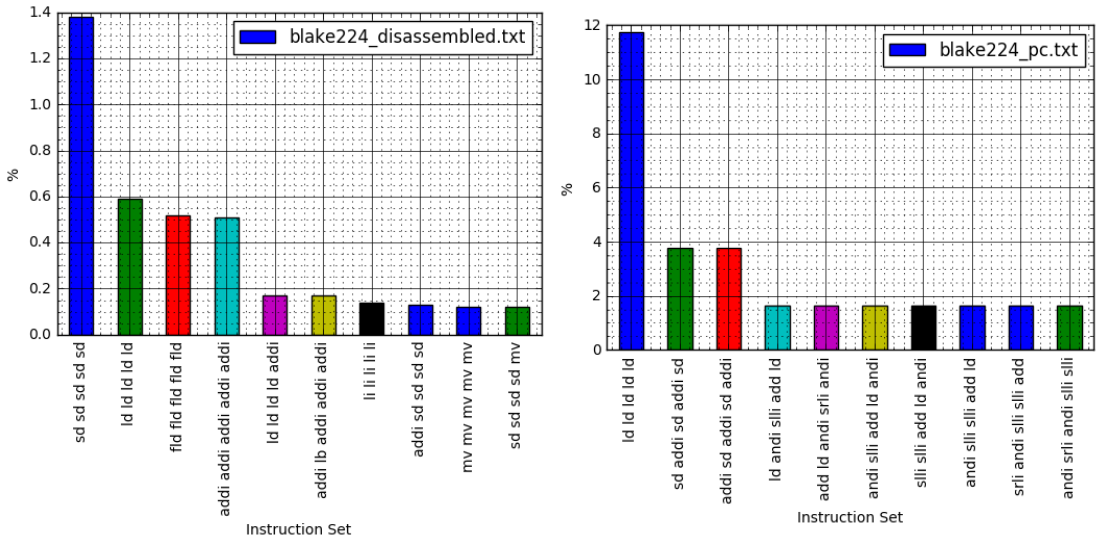
Figure 4.49: CAMELLIA:percentage of set of five instructions in terms of memory(static) and execution time(runtime)



(a) Static

(b) Runtime

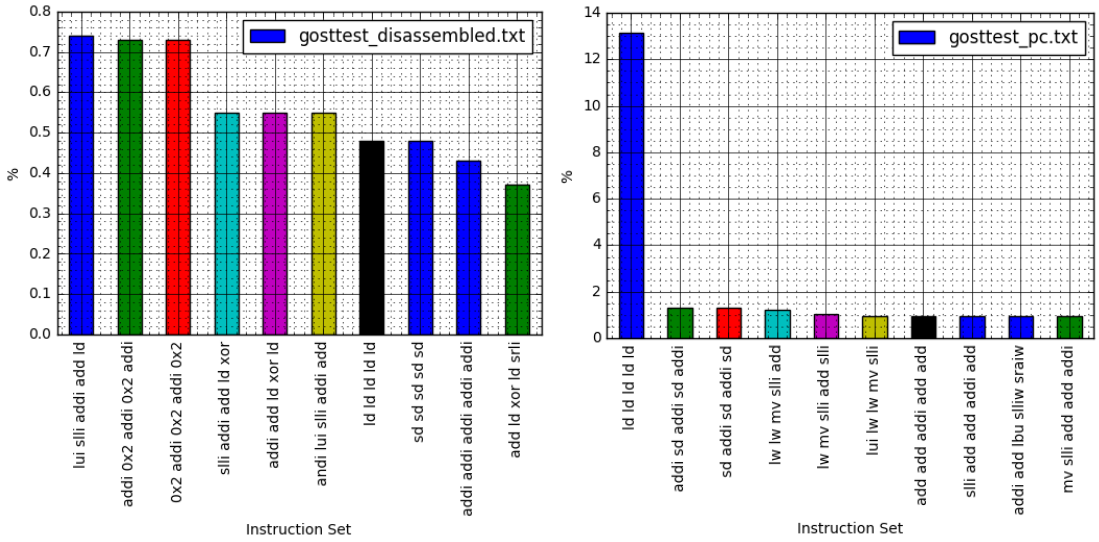
Figure 4.50: ARCFOUR:percentage of set of five instructions in terms of memory(static) and execution time(runtime)



(a) Static

(b) Runtime

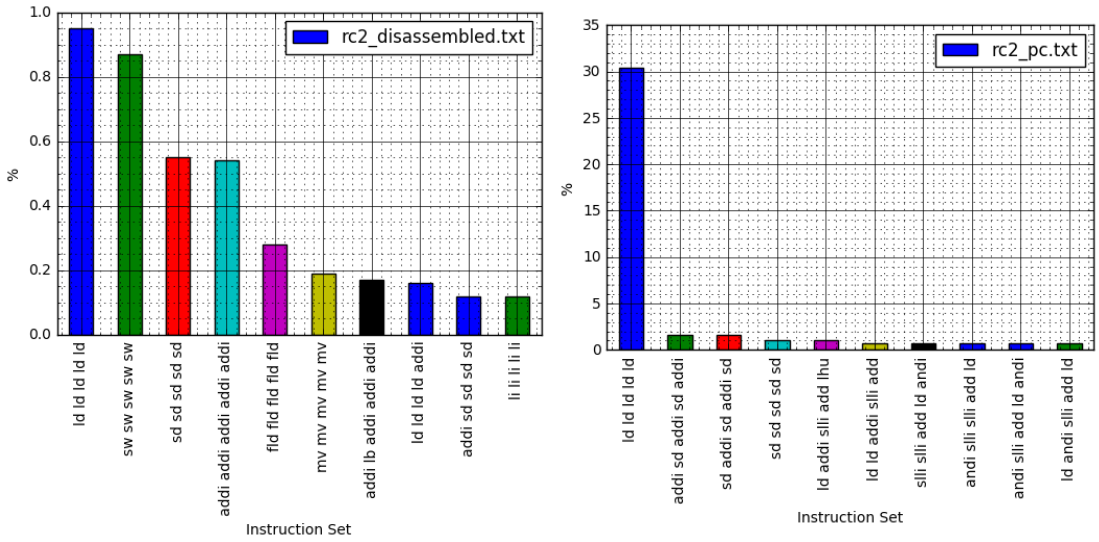
Figure 4.51: BLAKE224:percentage of set of five instructions in terms of memory(static) and execution time(runtime)



(a) Static

(b) Runtime

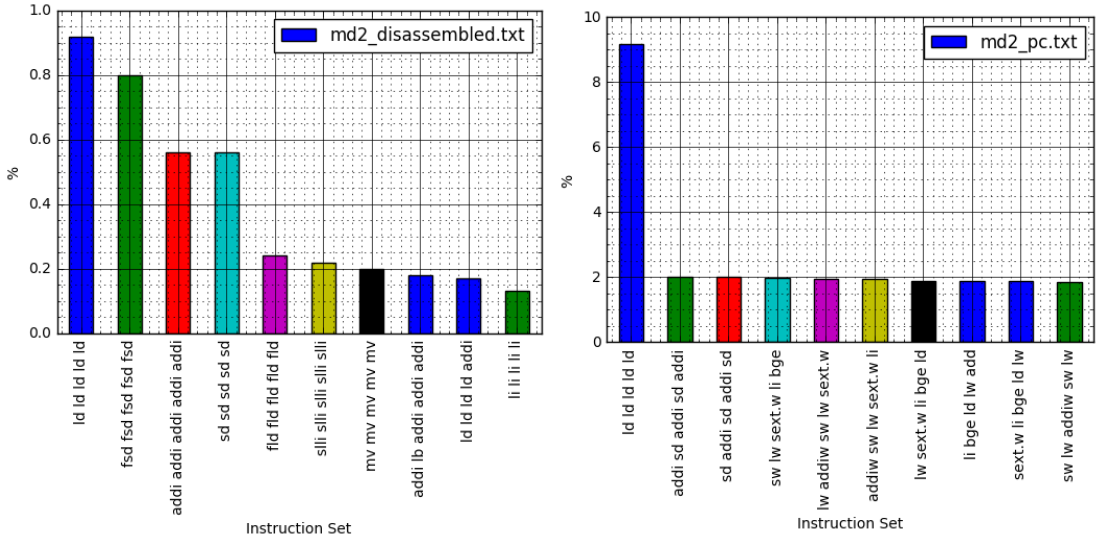
Figure 4.52: GOST:percentage of set of five instructions in terms of memory(static) and execution time(runtime)



(a) Static

(b) Runtime

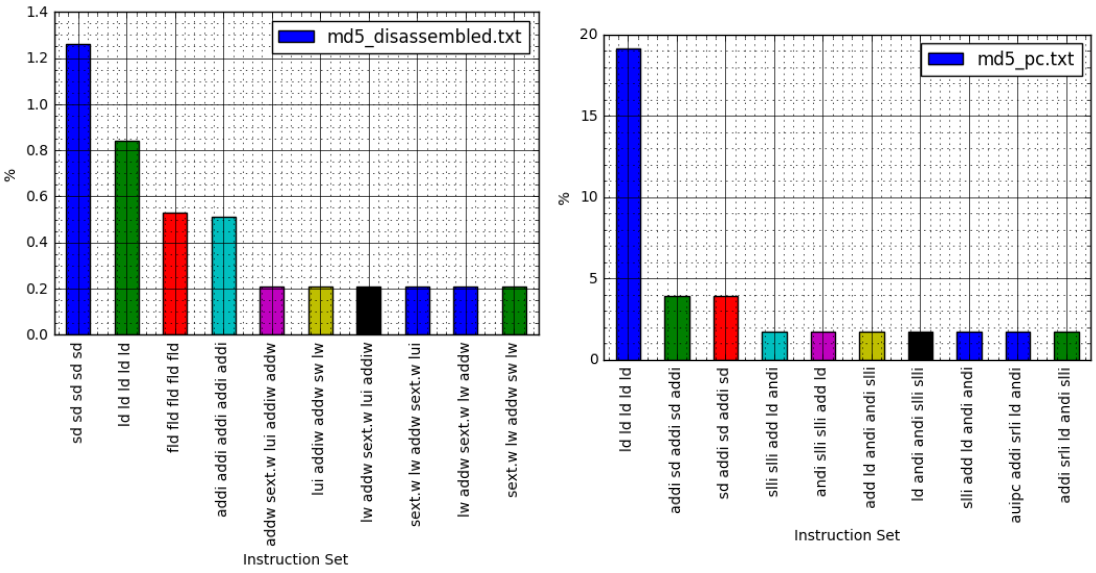
Figure 4.53: RC2:percentage of set of five instructions in terms of memory(static) and execution time(runtime)



(a) Static

(b) Runtime

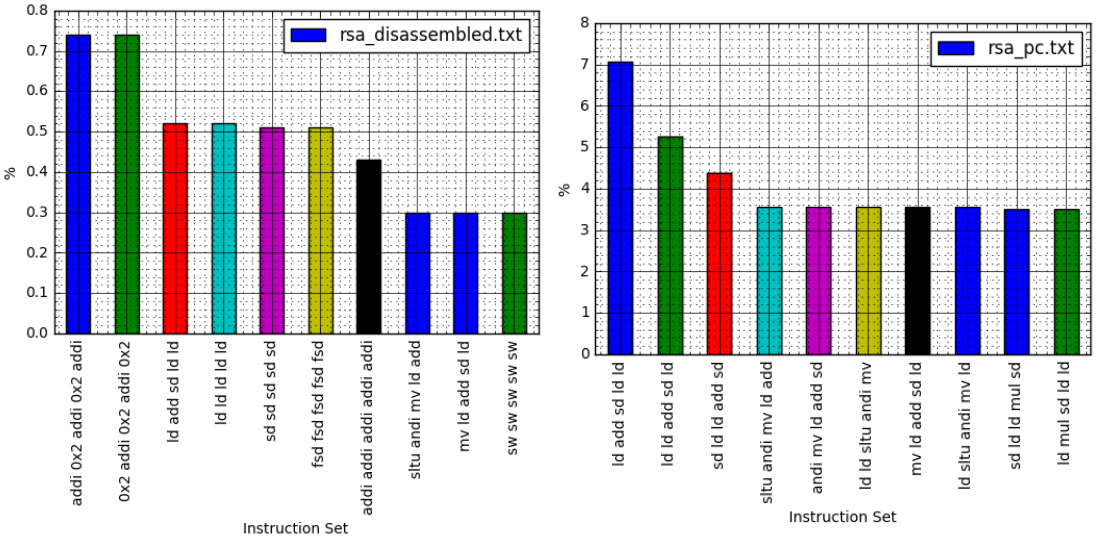
Figure 4.54: MD2:percentage of set of five instructions in terms of memory(static) and execution time(runtime)



(a) Static

(b) Runtime

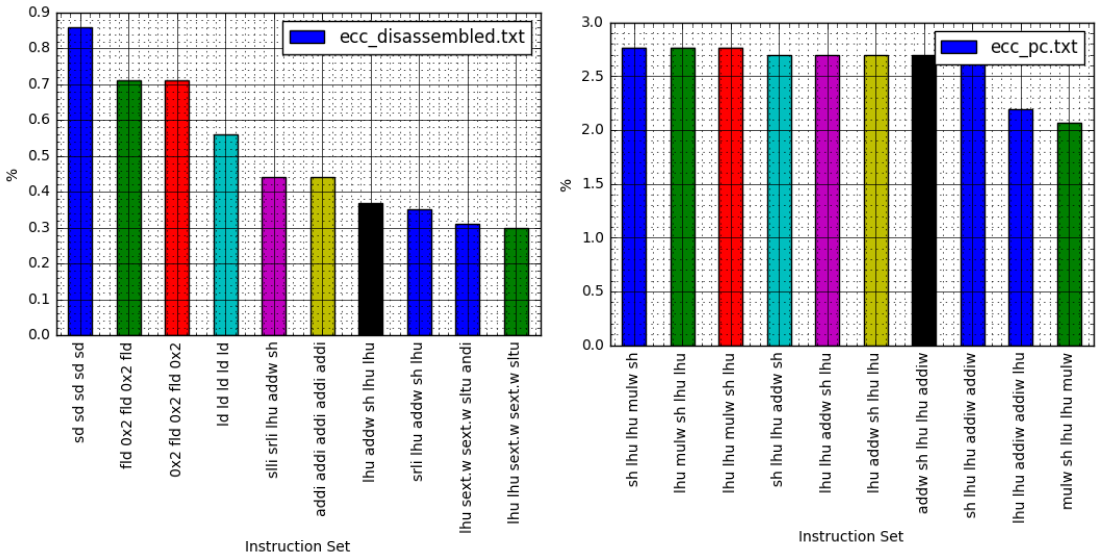
Figure 4.55: MD5:percentage of set of five instructions in terms of memory(static) and execution time(runtime)



(a) Static

(b) Runtime

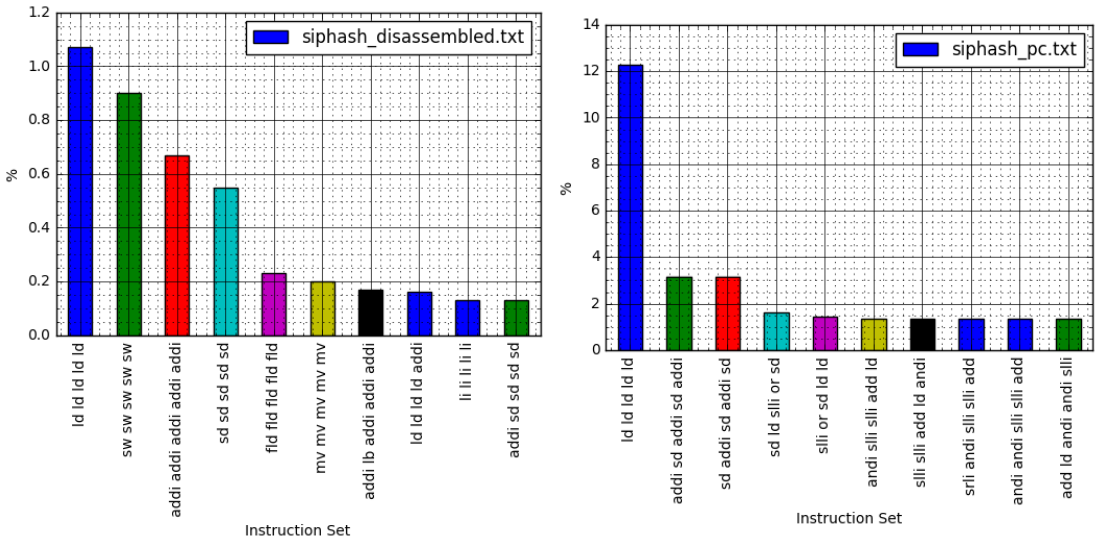
Figure 4.56: RSA:percentage of set of five instructions in terms of memory(static) and execution time(runtime)



(a) Static

(b) Runtime

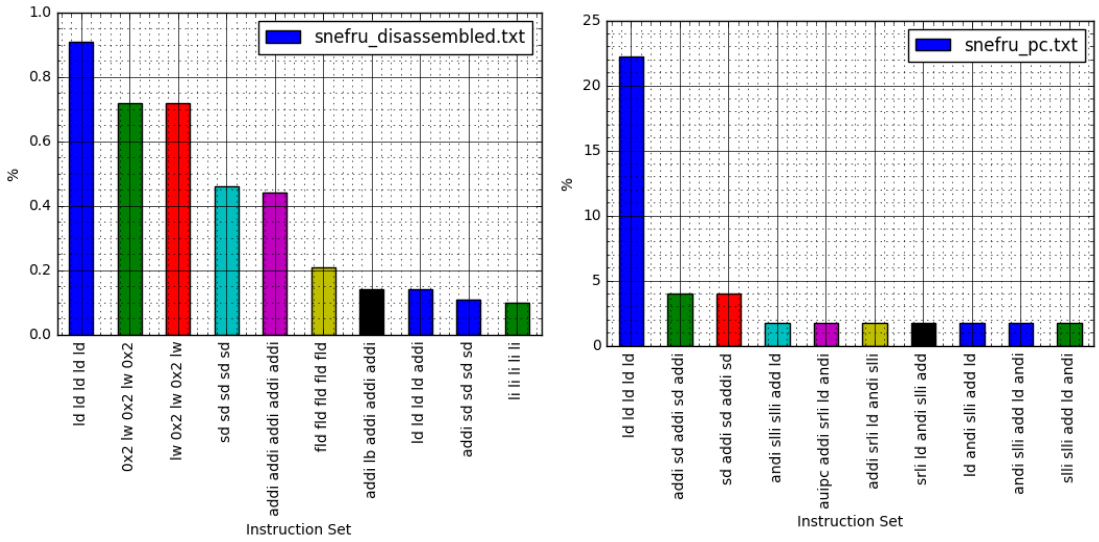
Figure 4.57: ECC:percentage of set of five instructions in terms of memory(static) and execution time(runtime)



(a) Static

(b) Runtime

Figure 4.58: SIPHASH:percentage of set of five instructions in terms of memory(static) and execution time(runtime)



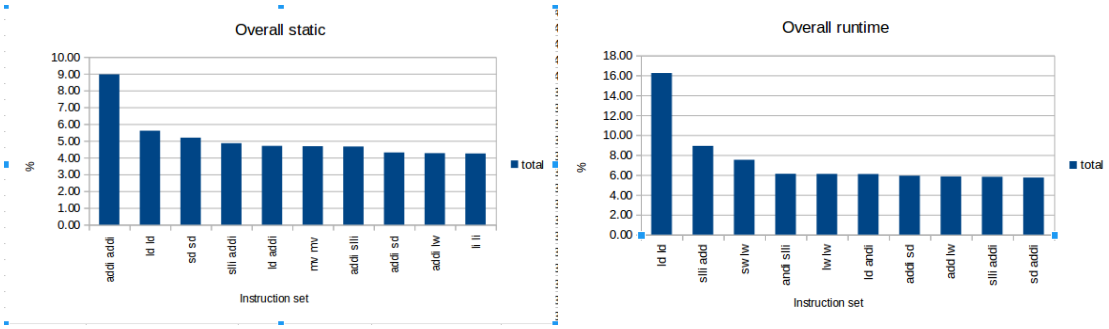
(a) Static

(b) Runtime

Figure 4.59: SNEFRU:percentage of set of five instructions in terms of memory(static) and execution time(runtime)

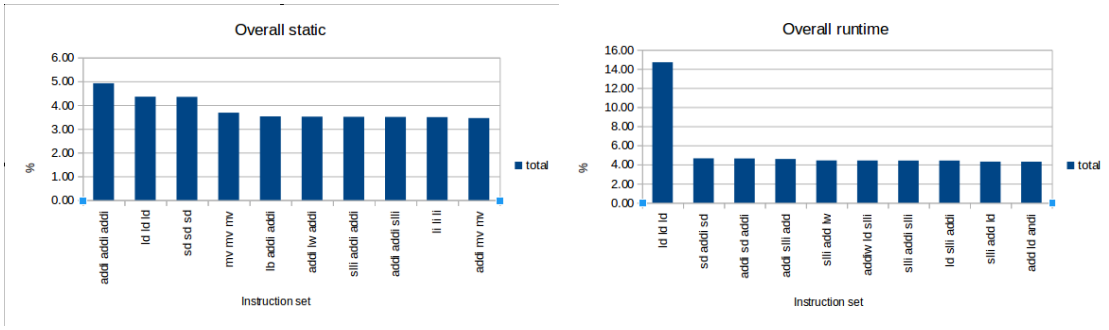
4.2 Overall results

Histogram of top ten instruction sets overall in terms of memory and execution time are plotted below:



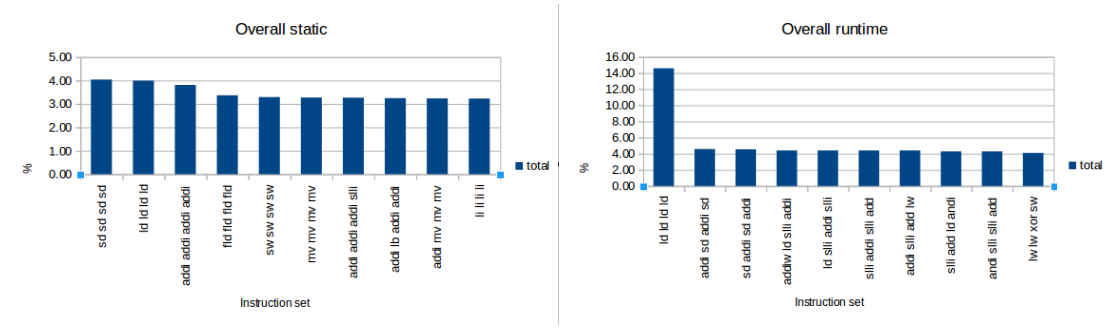
(a) Static (b) Runtime

Figure 4.60: overall percentage of set of two instructions in terms of memory(static) and execution time(runtime)



(a) Static (b) Runtime

Figure 4.61: overall percentage of set of three instructions in terms of memory(static) and execution time(runtime)

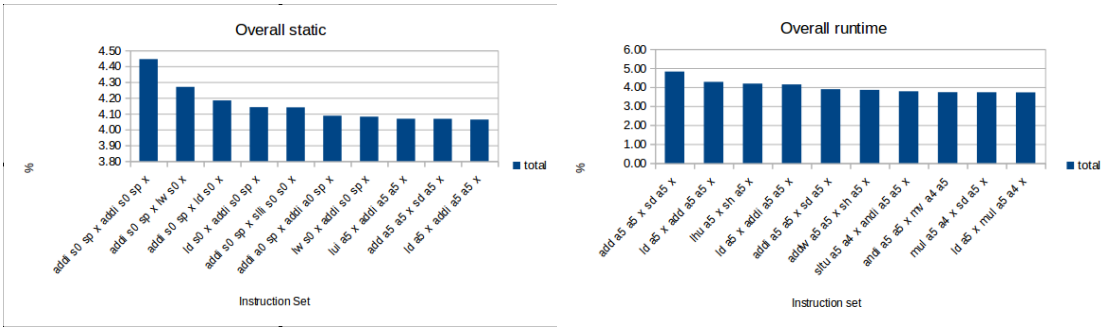


(a) Static (b) Runtime

Figure 4.62: overall percentage of set of four instructions in terms of memory(static) and execution time(runtime)

4.3 Further analysis

Further we have looked at potential instruction set with interdependency.Source register operands are kept as don't care. Instruction starting with load operation and arithmetic operation on values loaded will have more significance.



(a) Static (b) Runtime

Figure 4.63: overall percentage of set of interdependent two instructions in terms of memory(static) and execution time(runtime)

4.4 Amdahl's Law

Amdahl's law is fundamental theorem of computer architecture.It is a formula used to find the maximum improvement possible by improving a particular part of a system.

If we can speed up x percentage of the program by S times Amdahl's law gives us total speed up by below formulaHennessy and Patterson (2011).

$$Speedup_{total} = \frac{1}{(1-x)+\frac{x}{S}}$$

From the analysis in 4.62 we can see that set of four consecutive loads are taking around 14 % of total execution time overall.So if we can propose a new instruction names MLD to load four values in one instruction, 2.25 times speed up. Using Amdahl's law overall speed will be 1.08.

4.5 Future Scope

1. New instruction can be proposed for Multiple load/store operation for block data transfer
2. Application specific potential instruction set extension can be tried on harware based on analysis results
3. Dependent instruction set block for further speed up in cryptography applications

REFERENCES

1. **Grabher, P.**, **J. Großschädl**, and **D. Page**, Light-weight instruction set extensions for bit-sliced cryptography. *In International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2008.

2. **Hennessy, J. L.** and **D. A. Patterson**, *Computer architecture: a quantitative approach*. Elsevier, 2011.

3. **Katz, J.**, **A. J. Menezes**, **P. C. Van Oorschot**, and **S. A. Vanstone**, *Handbook of applied cryptography*. CRC press, 1996.

4. **Waterman, A.**, **Y. Lee**, **D. A. Patterson**, and **K. Asanovic** (2011). The risc-v instruction set manual, volume i: Base user-level isa. *EECS Department, UC Berkeley, Tech. Rep. UCB/EECS-2011-62*.