TOWARDS UNIVERSAL COSMIC SIZE MEASUREMENT AUTOMATION

H.Soubra, Y.Abufrikha et A. Abran

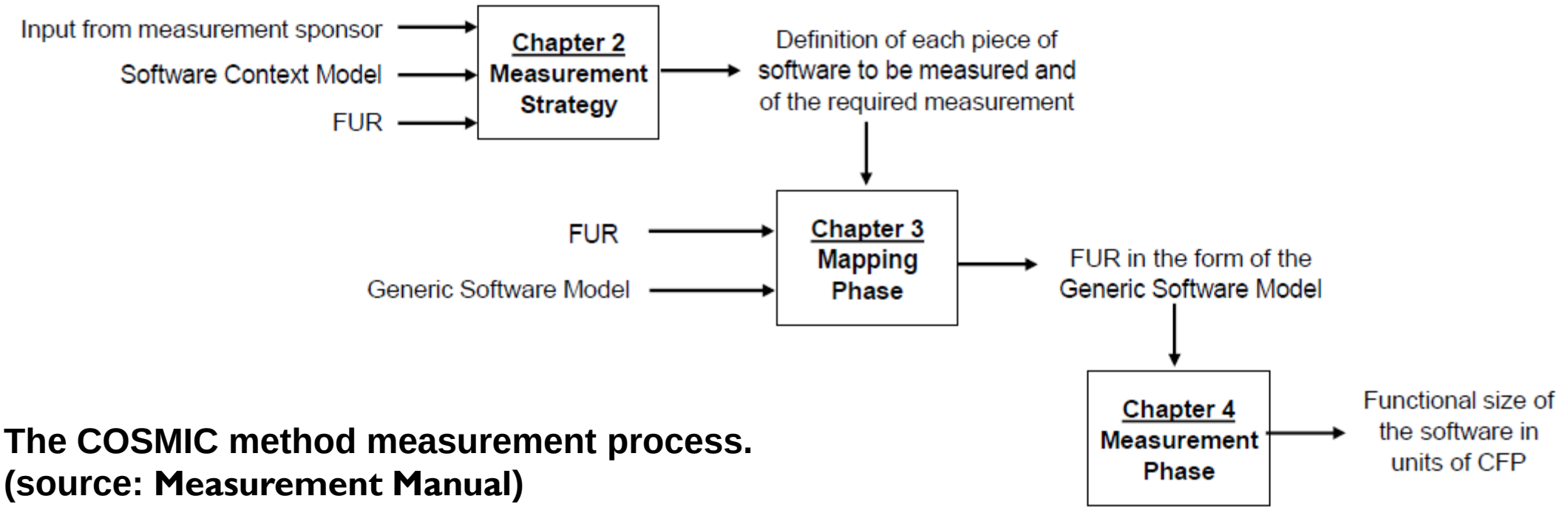
INTRODUCTION

- There are 8945 different programming languages!
- Automating COSMIC-based FSM: complete mapping between the principles of the COSMIC method and the notation of a programming language.
- And different programming languages may produce different results when implementing the same set of requirements.
- This is a feasibility study of an approach to a 'universal' tool based on COSMIC ISO 19761 and MIPS to automate the measurement of software written in different programming languages.

AGENDA

- Introduction
- COSMIC Overview
- Related Work
- Proposed Approach
  - MIPS Overview
  - Mapping COSMIC to MIPS
  - Automation Prototype
- Conclusion

COSMIC



The COSMIC method measurement process.
(source: Measurement Manual)

AGENDA

- Introduction
- COSMIC Overview
- Related Work
- Proposed Approach
  - MIPS Overview
  - Mapping COSMIC to MIPS
  - Automation Prototype
- Conclusion

RELATED WORK

- Examples of COSMIC-based automation tools from the literature:

Tool	Year	Author
Tool using Simulink model	2011	Soubra et al. [4]
ScopeMaster	2018	Hammond et al. [5]
A tool for the automation of functional size measurement with RUP	2004	Azzouz and Abran [9]
μcROSE	2005	Diab et al. [10]
A procedure that measures the functional size	2015	Gonultas and Tarhan [11]
A tool using SCADE model	2015	Soubra et al. [12]
UML profile tool	2011	Lind and Heldal [13]

- None of the proposed tools can be considered universal since they require specific types of input languages/models.

- The concept of a universal tool is a tool that is applicable to all types of input languages/models.

AGENDA

- Introduction
- COSMIC Overview
- Related Work
- Proposed Approach
  - MIPS Overview
  - Mapping COSMIC to MIPS
  - Automation Prototype
- Conclusion

PROPOSED APPROACH

Once a program is translated into machine code, it becomes independent of the original language it was written in.

Map and use machine code as Input for COSMIC-FSM

Use MIPS as POC and generalize approach

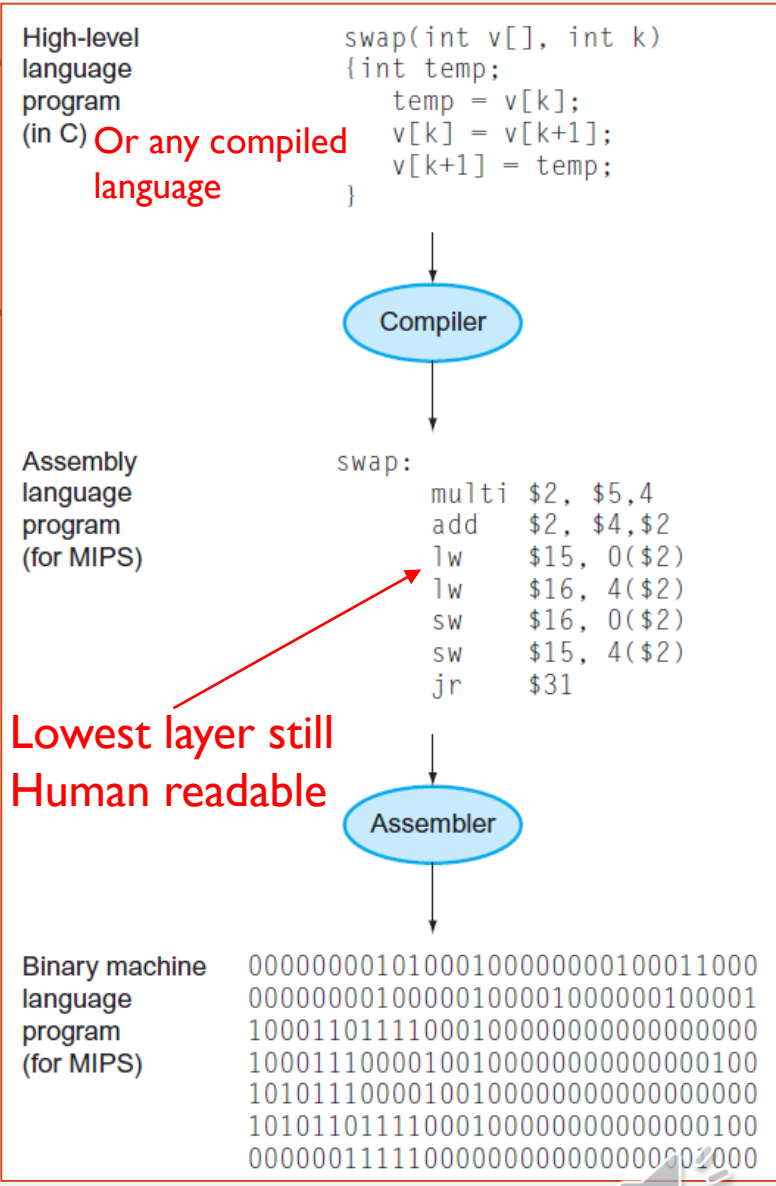


Image source: Patterson, David A., and John L. Hennessy. *Computer Organization and Design: The Hardware Software Interface*.

PROPOSED APPROACH: MIPS OVERVIEW

- MIPS-Microprocessor without Interlocked Pipelined Stages: is a reduced instruction set computer (RISC)

- Widely used in Computer Architecture courses and has been used in some implementations in game consoles : Nintendo 64 and PlayStation.

PROPOSED APPROACH: MIPS OVERVIEW

- Five-stage execution pipeline: fetch, decode, execute, memory-access, write-result.
- Regular instruction set, all instructions are 32-bit.
- Three-operand arithmetical and logical instructions.
- 32 general-purpose registers of 32-bits each.
- …

- The components of MIPS architecture are:

scope of our study

  - MIPS instruction set architecture (ISA)
  - MIPS privileged resource architecture (PRA)
  - MIPS modules and application-specific extensions (ASEs)
  - MIPS user defined instructions (UDIs)

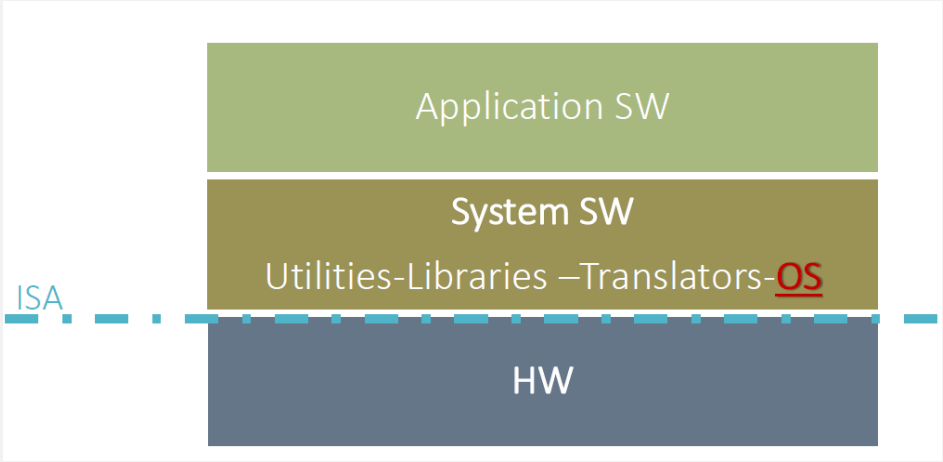
ISA AND MACHINE CODE

- Two sides of the same coin!
- ISA serves as the boundary between the software and hardware.

MIPS code example (file)

```
    addi $s6 , $0 , 23
    addi $t5 , $0 , 5
    sw $t5 , 0 ( $ s6 )
    addi $s6 , $0 , 8
Loop:
    sll $t1 , $s3 , 2
    add $t1 , $t1 , $s6
    lw $t0 , 0 ( $t1 )
    bne $t0 , $zero , Exit
    addi $s3 , $s3 , 1
    j Loop
Exit:
```

add $s1,$s2,$s3

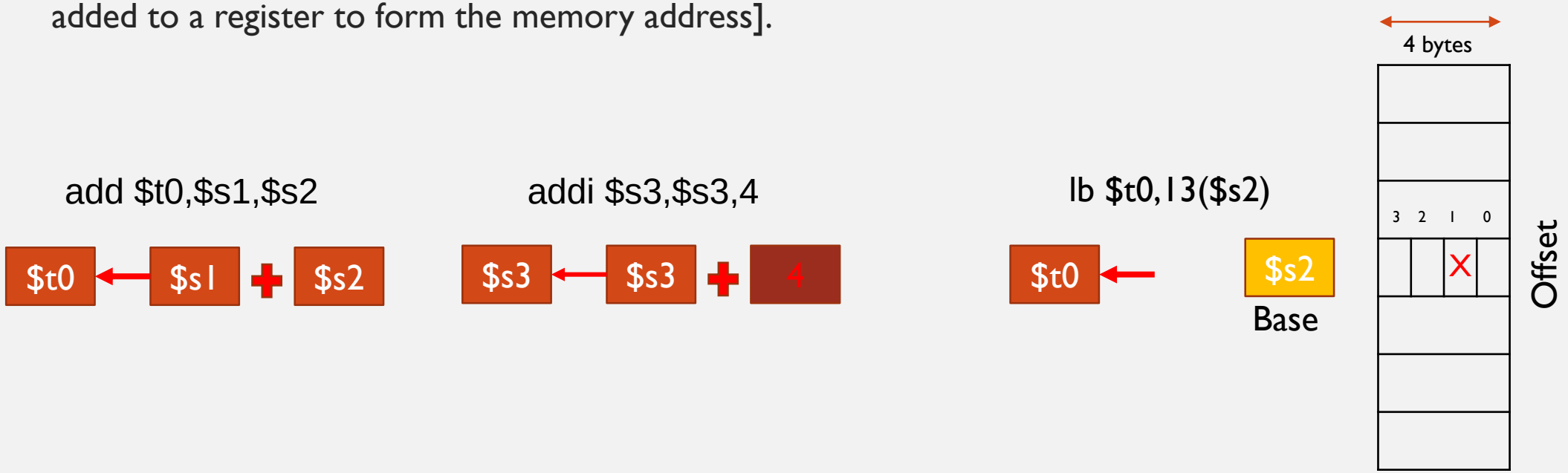


ISA- ADDRESSING MODES AND MEMORY

MIPS uses byte addressing to access memory operands

Objects must be aligned

- MIPS addressing modes are Register, Immediate, and Displacement [where a constant offset is added to a register to form the memory address].



ISA- OPERATIONS/CONTROL

Register operation expl

Memory operation expl

Mnemonic	Meaning	Type
ADD	Add	R
ADDI	Add Immediate	I
ADDIU	Add Unsigned Immediate	I
ADDU	Add Unsigned	R
AND	Bitwise AND	R
ANDI	Bitwise AND Immediate	I
BEQ	Branch if Equal	I
BLEZ	Branch if Less Than or Equal to Zero	I
BNE	Branch if Not Equal	I
BGTZ	Branch on Greater Than Zero	I
DIV	Divide	R
DIVU	Unsigned Divide	R
j	Jump to Address	J
JAL	Jump and Link	J
JR	Jump to Address in Register	R
LB	Load Byte	I
LBU	Load Byte Unsigned	I
LHU	Load Halfword Unsigned	I
LUI	Load Upper Immediate	I
LW	Load Word	I
MFHI	Move from HI Register	R
MTHI	Move to HI Register	R
MFLO	Move from LO Register	R
MTLO	Move to LO Register	R
MFC0	Move from Coprocessor 0	R
MULT	Multiply	R
MULTU	Unsigned Multiply	R
NOR	Bitwise NOR (NOT-OR)	R
XOR	Bitwise XOR (Exclusive-OR)	R
OR	Bitwise OR	R
ORI	Bitwise OR Immediate	I
SB	Store Byte	I
SH	Store Halfword	I
SLT	Set to 1 if Less Than	R
SLTI	Set to 1 if Less Than Immediate	I
SLTIU	Set to 1 if Less Than Unsigned Immediate	I
SLTU	Set to 1 if Less Than Unsigned	R
SLL	Logical Shift Left	R
SRL	Logical Shift Right (0-extended)	R
SRA	Arithmetic Shift Right (sign-extended)	R
SUB	Subtract	R
SUBU	Unsigned Subtract	R
SW	Store Word	I

AGENDA

- Introduction
- COSMIC Overview
- Related Work
- Proposed Approach
  - MIPS Overview
  - Mapping COSMIC to MIPS
  - Automation Prototype
- Conclusion

MAPPING COSMIC TO MIPS

Rule number	COSMIC element	Rule description
1	Functional Process (FP)	Identify 1 functional process for each subroutine in the file.
2	Data movement	Identify 1 Entry (E) for each source register in each instruction in a FP.
3	Data movement	Identify 1 Entry (E) for each immediate each instruction in a FP.
4	Data movement	Identify 1 Exit (X) for a destination register in each instruction in a FP.
5	Data movement	Identify 1 Exit (X) for the new PC value after branch and jump instructions.
6	Data movement	Identify 1 Exit (X) for the return value after branch and link and jump and link instructions.
7	Data movement	Identify 1 Read (R) for each Load instruction inside a FP.
8	Data movement	Identify 1 Write (W) for each Store instruction inside a FP.
9	Functional process size	Aggregate the COSMIC Function Point (CFP) for each data movement in a FP. to obtain the size of the process.
10	Size of the software	Aggregate the CFP of each FP. to obtain the size of the whole software.

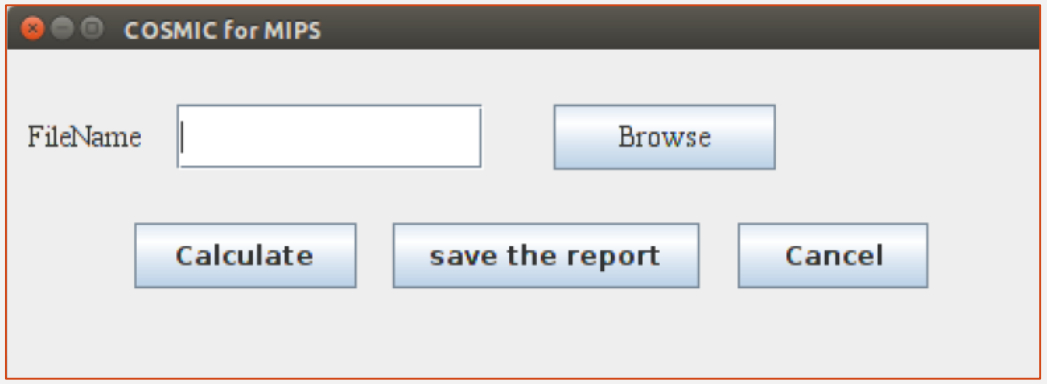
AGENDA

- Introduction
- COSMIC Overview
- Related Work
- Proposed Approach
  - MIPS Overview
  - Mapping COSMIC to MIPS
  - Automation Prototype
- Conclusion

Opcode= MIPS operation code
(Unique instruction ID)

AUTOMATION PROTOTYPE

Parse Input MIPS code file → store it in an ArrayList → perform string manipulation → obtain the Opcode + labels (Subroutines)



Identify E, X, R, W according to the rules → Yield measurement result

Proto Developed in JAVA

AUTOMATION PROTOTYPE

MIPS code Input file: TestFile.asm

```
    addi $s6 , $0 , 23
    addi $t5 , $0 , 5
    sw $t5 , 0 ( $ s6 )
    addi $s6 , $0 , 8
Loop:
    sll $t1 , $s3 , 2
    add $t1 , $t1 , $s6
    lw $t0 , 0 ( $t1 )
    bne $t0 , $zero , Exit
    addi $s3 , $s3 , 1
    j Loop
Exit:
```

Output file:

```
This is the report coresponding to TestFile.asm

The file had 14 lines of code.
The total number of instructions in the file = 10 instruction(s).

COSMIC calculation:

The file had 3 functional processes.
    main
        Loop
        Exit
The functional size = 33 CFP.

The total number of entries = 22.
 The instructions that caused the number of Entries:
            addi $s6 , $0 , 23------> 2 entries.

            addi $t5 , $0 , 5------> 2 entries.

            sw $t5 , 0 ( $ s6 ) ------> 3 entries.

            addi $s6 , $0 , 8------> 2 entries.

            sll $t1 , $s3 , 2 ------> 2 entries.

            add $t1 , $t1 , $s6 ------> 2 entries.

            lw $t0 , 0 ( $t1 ) ------> 2 entries.

            bne $t0 , $zero , Exit------> 4 entries.

            addi $s3 , $s3 , 1------> 2 entries.

            j Loop------> 1 entry.
```

Output file (ctnd):

```
The total number of exits = 9.
 The instructions that caused the number of Exits:
            addi $s6 , $0 , 23------> 1 exit.

            addi $t5 , $0 , 5------> 1 exit.

            addi $s6 , $0 , 8------> 1 exit.

            sll $t1 , $s3 , 2 ------> 1 exit.

            add $t1 , $t1 , $s6 ------> 1 exit.

            lw $t0 , 0 ( $t1 ) ------> 1 exit.

            bne $t0 , $zero , Exit------> 1 exit.

            addi $s3 , $s3 , 1------> 1 exit.

            j Loop------> 1 exit.

The total number of read = 1.
 The instructions that caused the number of Reads:
            lw $t0 , 0 ( $t1 ) ------> 1 read.

The total number of write = 1.
 The instructions that caused the number of Writes:
            sw $t5 , 0 ( $ s6 ) ------> 1 write.
```

AUTOMATION PROTOTYPE

Limitations:

- This is a feasibility prototype
- Not all MIPS instructions -including pseudo-instructions- are included.
- Accuracy and the precision of the tool should be analyzed with more test cases.

AGENDA

- Introduction
- COSMIC Overview
- Related Work
- Proposed Approach
  - MIPS Overview
  - Mapping COSMIC to MIPS
  - Automation Prototype
- Conclusion

CONCLUSION

1. The goal of this study: to propose an approach for a `universal' tool based on COSMIC ISO 19761 to ensure that the measurement of all types of input software written in different programming languages is correctly automated.

2. Outcomes of the study: Mapping rules and a feasibility prototype tool based on COSMIC and MIPS was developed using Eclipse Java based on the latest version of the MIPS architecture.

3. Promising results: the 'universal' tool may be achieved by generalizing the approach proposed in this study to cover, and use, the machine code (ISA-Instruction Set Architecture) generated by any compiler/Assembler to get the COSMIC functional size of a program.

Q&A

hassan.soubra@guc.edu.eg

yomna.abufrihka@student.guc.edu.eg

alain.abran@etsmtl.ca