

A PRELIMINARY REPORT ON

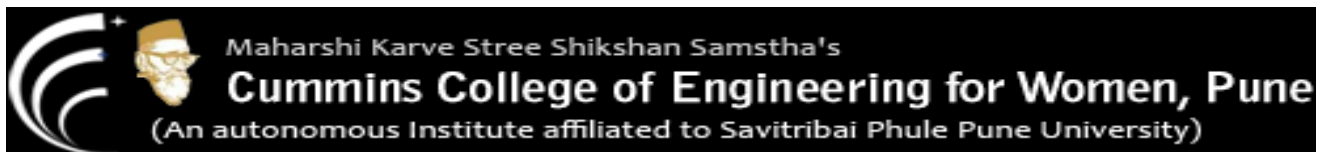
**GENERATING COMPILER PHASES FROM SPECIFICATIONS
OF INTERMEDIATE REPRESENTATIONS**

SUBMITTED TO THE SAVITRIBAI PHULE PUNE UNIVERSITY, PUNE
IN THE PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE AWARD OF THE DEGREE
OF

BACHELOR OF COMPUTER ENGINEERING

SUBMITTED BY

Sai Ghule	C22018221334
Bhagyashree Rane	C22018221311
Shravasti Deore	C22018881911



DEPARTMENT OF COMPUTER ENGINEERING

2021 -2022

CERTIFICATE

This is to certify that the project report entitled

“GENERATING COMPILER PHASES FROM SPECIFICATIONS of INTERMEDIATE REPRESENTATIONS (IRs)”

Submitted by

Sai Ghule	C22018221334
Bhagyashree Rane	C22018221311
Shravasti Deore	C22018881911

is a bonafide student of this institute and the work has been carried out by her under the supervision of **Prof. Dr. Chhaya Gosavi** and it is approved for the partial fulfillment of the requirement of Savitribai Phule Pune University, for the award of the degree of **Bachelor of Computer Engineering**.

Prof. Dr. Chhaya Gosavi
Guide
Department of Computer Engineering

Dr. Supriya Kelkar
Head,
Department of Computer Engineering

Dr. M. B. Khambete
Principal,
Cummins College of Engineering for Women Pune – 52

Place : Pune

Date : 25/1/2022

ACKNOWLEDGEMENT

We would like to express our sincere gratitude towards our project guide Prof.Dr.Chhaya Gosavi for her constant encouragement and valuable guidance during the completion of this project work.

We would like to thank our external mentor Prof.Dr.Uday Khedkar sir for his constant guidance and support.

We would also like to thank Dr. Supriya Kelkar (H.O.D.,Comp) for her continuous valuable guidance, support, valuable suggestions and her precious time in every possible way inspite of her busy schedule throughout our project activity. We take this opportunity to express our sincere thanks to all the staff members of Comp. Department for their constant help whenever required. Finally, we express our sincere thanks to all those who helped us directly or indirectly in many ways in completion of this project work.

Sai Ghule

Bhagyashree Rane

Shravasti Deore

ABSTRACT

The motivation behind the implementation of SCLP has been to create a small well-crafted code base for a UG compiler construction lessons such that it can be enhanced systematically by the students. The focus is on incremental construction with some increments coming from language features and some coming from the phases in compilation.

The project, as a whole, aims to construct a simple compiler used to teach compilers at IITB, however, it does not include production compiler complexities such as code optimization, etc. Generating Compiler Phases from Specifications of IRs is just one incremental construction toward SCLP.

A compiler generator is a program which from its input produces a compiler which translates from one programming language to another. Compiler generation is the field of Computer Science concerned with using computers as a tool for constructing compilers.

Programmers who have written compilers by hand have realized that the process of writing compilers is completely systematic to be written by hand and there should be a way to be done by a machine.

The Lexical Analysis and Parsing phases are well understood that it has been possible to produce parser generators that produce a parser for the specific language. The parser generator are produced from the description of the syntax of the programming language. For example, LEX & YACC.

To generate a code for a language, it is necessary to have a clear understanding of the semantics of the language. In order to be able to make a compiler generator, one needs a theoretical framework for expressing the semantics of the languages involved.

The formal specifications of a language provides insights and understanding of the language requirements and the language statements design.

This project is to create formal language specifications for AST(Abstract Syntax Tree), TAC(Three Address Code), RTL(Register Transfer Logic) of the SCLP compiler to generate Translation Rules from the language specifications.

The Translation Rules thus generated will be the input to the Translator Generator which in turn would generate an auto generated IRGen to translate the output of the previous phase to the next phase.

TABLE OF CONTENTS

LIST OF ABBREVIATIONS	i
LIST OF FIGURES	ii
LIST OF TABLES	iii

CHAPTER	TITLE	PAGE NO.
Sr.No.	Title of Chapter	Page No.
1	Introduction	9
1.1	Motivation	9
1.2	Problem Definition.....	9
2	Literature Survey	10
2.1	Background of domain	10
2.2	Research Papers	12
3	Requirements	14
3.1	System Requirements Specification	14
3.1.1	Scope	14
3.1.2	Not in Scope	14
3.1.3	Nonfunctional Requirements	15
3.1.3.1	Software Quality Attributes	15
4	System Design	16
4.1	System Architecture	16
4.1.1	Incremental Model	17
4.2	UML Diagrams	18
4.2.1	Data Flow Diagram	18
4.2.2	Activity Diagram	19
4.2.3	Class Diagram.....	19
4.2.4	Component Diagram	20

4.2.5 Class Diagram for RTL	21
4.2.6 Class Diagram for TAC	21
4.2.7 IR Gen USE case Diagram	22
4.2.8 SCLP Use case Diagram	23
5. IMPEMETATION ASPECTS	24
5.1 Study ScIp	24
5.2 Bug Reports.	25
5.3 SCLP web page changes.	25
5.4 Thorough study of MIPS instructions for SPIM	26
5.5 Formalized ASM specifications for SPIM.	27
5.6 Thorough study of RISC-V instructions for RIPES.	28
5.7 Formalized ASM specifications for RIPES	29
5.8 Specification Docs for RTL.	30
6. Technology	31
7. Conclusion and Future work	32
8. Appendix A	32
9. References	33

LIST OF ABBREVIATIONS

ABBREVIATION	ILLUSTRATION
AST	Abstract Syntax Tree
TAC	Three Address Code
RTL	Register Transfer Language
ASM	Assembly Language

LIST OF TABLES

TABLE	ILLUSTRATION	PAGE No.
2.2	Comparisons, Research Paper Studied/products compared	29

1. INTRODUCTION

1.1 MOTIVATION

The motivation behind the implementation of SCLP has been to create a small well-crafted code base for a UG compiler construction lessons such that it can be enhanced systematically by the students. The focus is on incremental construction with some increments coming from language features and some coming from the phases in compilation.

Generating Compiler Phases from Specifications of IRs is one such incremental construction.

A secondary goal of SCLP is to support interpretation to understand the relationship between compilation and interpretation first-hand.

1.2 PROBLEM DEFINITION

A compiler generator is a program which from its input produces a compiler which translates from one programming language to another. Compiler generation is the field of Computer Science concerned with using computers as a tool for constructing compilers.

Programmers who have written compilers by hand have realized that the process of writing compilers is completely systematic to be written by hand and there should be a way to be done by a machine. Lexical Analysis and Parsing have been understood well enough that it has been possible to produce parser generators that produce a parser for the specific language.

However, the interfaces between the subsequent intermediate representations of compilation (namely, TAC, RTL, ASM) have not yet achieved code generation on the scale scanners and parsers have.

2. LITERATURE SURVEY

2.1 BACKGROUND OF DOMAIN

- The Language Processor SCLP takes input programs written in languages that are a subset of C. Based on the command line parameters, the read program is either interpreted or compiled to generate an equivalent .spim file(assembly language program)
- Compilation of the input program through SCLP involves translating one representation into another representation (each representation is an Intermediate Representation) in a series of steps.
- The series of steps of translating an input to asm are:

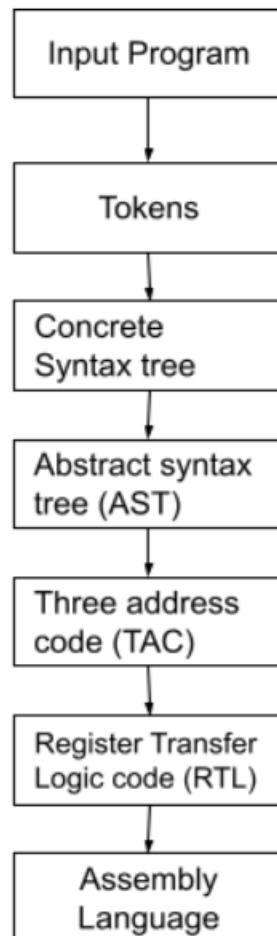


Figure 2.1- Steps for translating an input to ASM

- All these representations (other than the concrete syntax tree, which is constructed implicitly during parsing) are

available as text files produced by appropriate command line switches.

- There are six levels of the input language accepted by the SCLP compiler till date. Each level includes all the features of the previous level. Studying these levels was to be given preference.
- Some errors in the levels needed to be pointed out and studied in order to find solutions for them.
- Studying the specifications written for ASM(Assembly language) and generating the specifications for AST, TAC and RTL.
- Specification document has statements expressed in a language whose vocabulary, syntax, and semantics are formally defined. The need for a formal semantic definition means that the specification languages cannot be based on natural language; it must be based on mathematics.

For example, generating specifications for ASM

ASM:

`addi rt, rs, imm`

Specification:

$$[Add\ int] \frac{S(addi, o1, o2, o3) : \tau_{cmp}^S \quad o1 : \tau_{gpr}^S \quad o2 : \tau_{gpr}^S \quad o3 : \tau_{int}^S}{H; D; C; R \vdash S(addi, o1, o2, o3) \rightarrow H; D; C; R[o1 \mapsto R[o2] + o3] \quad R : \tau_{gpr}^S}$$

- The formal expression clearly tells the language designer that the instruction **addi** is of the phase **S**(i.e. ASM) and of type **cmp**(i.e. computational), it takes in 3 operands: **o1**, **o2**, **o3** of which **o1** and **o2** are of type **gpr**, and **o3** is of type **int**.
- **H;D;C;R** is the environment that undergoes a change after the expression **S(addi, o1, o2, o3)** is executed.
- **R**(the Register File) updates **R[o1]** to **R[o2]+o3** according to the instruction.

2.2 COMPARISONS, RESEARCH PAPERS STUDIED / PRODUCTS COMPARED

- V. Aho, Alfred V. Aho, Ravi Sethi, Jeffrey D. Ullman, J. D. Ullman, 1986. Compilers: Principles, Techniques, and Tools. Addison-Wesley.

Standard textbook about compiler construction for programming languages. Has in depth coverage of topics like lexical analysis, parsing, generation of intermediate code, etc. Also, describes development of a mini compiler.

Has level-wise production of a mini compiler.
Later the book introduces to the principal ideas in syntax-directed definitions and syntax-directed translations.

The theory of the syntax-directed definitions and syntax-directed translations is used in the later chapters to generate intermediate code for one programming language.

The book also covers code optimization and garbage collection which this project is sidelining so as to reduce the complexity and not go off the main aim of the project i.e. compiler generators from IR specifications.

- Tofte. M. 2012 compiler generators: what they can do, what they might do and what they will never do. Springer Science & Business Media.

Reports on the Compiler Generator CERES. The concept of Compiler Generator is described along with the comparison of CERES with other systems. Although, it does not say anything about intermediate representations.

Has insights about language specifications and how to use them to create compiler generators:

“A structural language definition consists of a finite number of rewrite rules that describe how source programs can be rewritten into terms of another language, the semantic language. There is one rule for each language construction of the source language. The construction in question appears on the left-hand side of the rule and the right-hand side is a term of the semantic language, the subterms of which may be expressed indirectly as the translation of the subphrases of the source language construction”

- Lee, P.. Realistic Compiler Generation. MIT Press.

Formal Specifications, their need and high-level descriptions, and generation of realistic compilers is primarily treated. Used denotational semantics.

Presents a new method for expressing the formal semantics of programming languages that allows realistic compilers to be generated automatically. The compilers thus generated are as efficient as the hand written compilers. The introduction of formal semantics make it easier to read and comprehend the semantics.

Demonstrates a working compiler generator called MESS which generates a compiler for pascal like language.

3. REQUIREMENTS

3.1. SYSTEM REQUIREMENTS SPECIFICATION

3.1.1 In Scope:

- Generation of ASM generator for SPIM
- Specifications for SPIM
- Specifications for RTL
- Machine readable form of the specs and translation rules
- Implementation of Translator Generator

- Generation of ASM generator for RIPES
- Specifications for RIPES
- Specifications for RTL
- Machine readable form of the specs and translation rules

- Generation of TAC generator
- Specifications for AST
- Specifications for TAC
- Translation rules from ASM to TAC
- Machine readable form of the specs and translation rules
- Implementation of Translator Generator

3.1.2 Not In Scope:

Inclusion of production compiler complexities like Code Optimization.

Implementation of generators that will construct interpreters.

3.3.3 Functional Requirements

- The IR generator should be generated using the Translator Generator using the machine readable form of the specifications and translation rules.
- The compiler should be able to read an input .c program and generate the expected outputs.
- SCLP should be able to generate AST, TAC, and RTL files as the outputs of the .c program.
- SCLP should be able to generate subsequent IR files for IR inputs.

3.3.4 Non-Functional Requirements

Software Quality Attributes:

- Availability
The generators should perform the tasks it is assigned to perform.
- Reliability
All levels of .c programs should be accurately executed.
- Flexibility
The system should be flexible for future development.
- Usability
The system should be easily usable for the UG courses.

3.3.5 Software Requirements

Linux - Ubuntu(Terminal)
Python 3
gcc(GNU Compiler Collection)
RIPES - RISCV simulator
SPIM - MIPS simulator
LEX & YACC

Hardware Requirements

Operating System - Linux distribution, Ubuntu

3. SYSTEM DESIGN

4.1 SYSTEM ARCHITECTURE

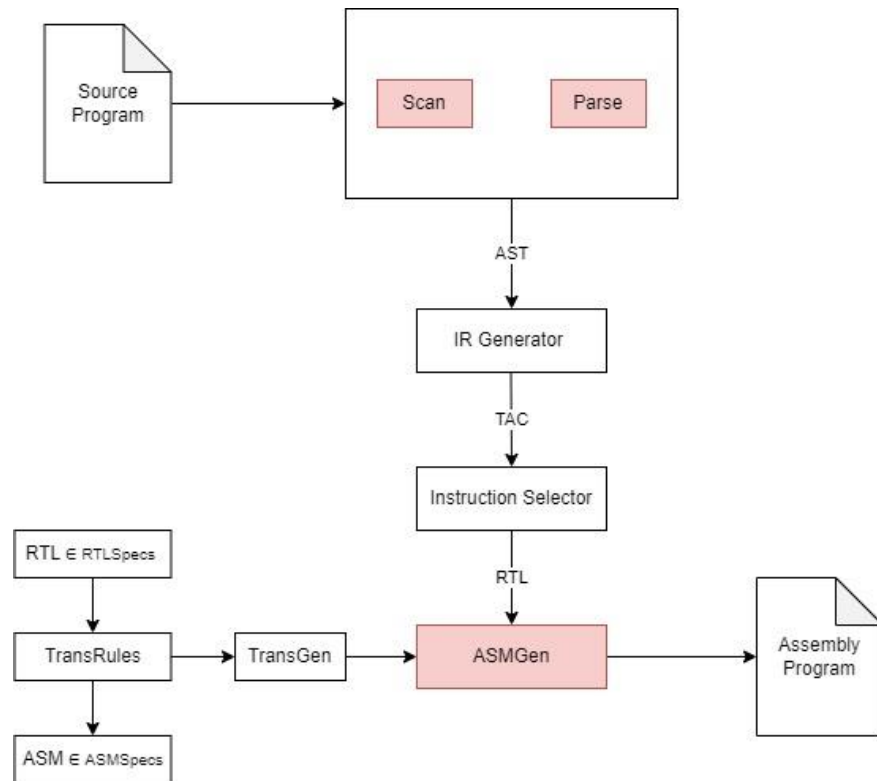


Fig 4.1- System Architecture of SCLP

Description-

- The above diagram is the previous implementation of SCLP.
- When a source program was passed to the sclp compiler, it was first scanned and parsed and the AST i.e the Abstract Syntax Tree was generated.
- The AST was then passes to the IR generator which converted the AST to a TAC i.e Three Address Code.
- The TAC was then passed to the Instruction Selector which produced the RTL i.e Register Transfer Level output.
- The RTL output was passed on to the ASM Gen which produced the Assemble Program.
- The previous group tried to automate the process of generating Assembly program by writing RTL specifications and then generating Translation Rules from them.
- Then they wrote the TransGen i.e the Translator Generator and the ASM gen i.e the ASM generator which generated the Assembly program.
- Their work made it possible to automatically generate the assembly program from RTL specifications.

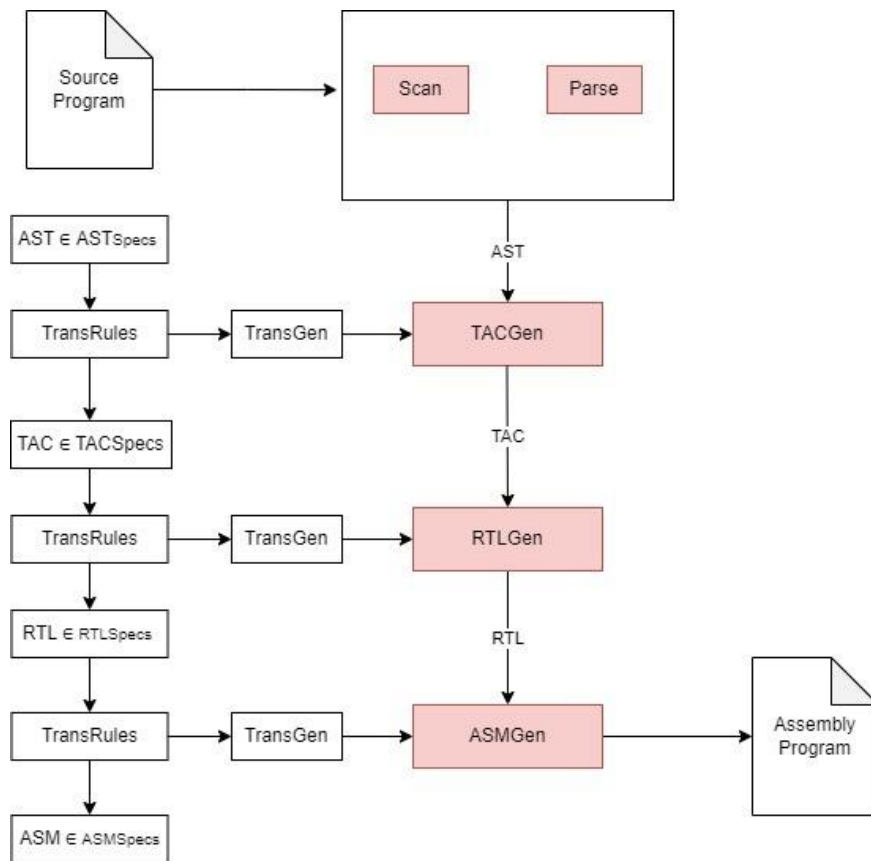


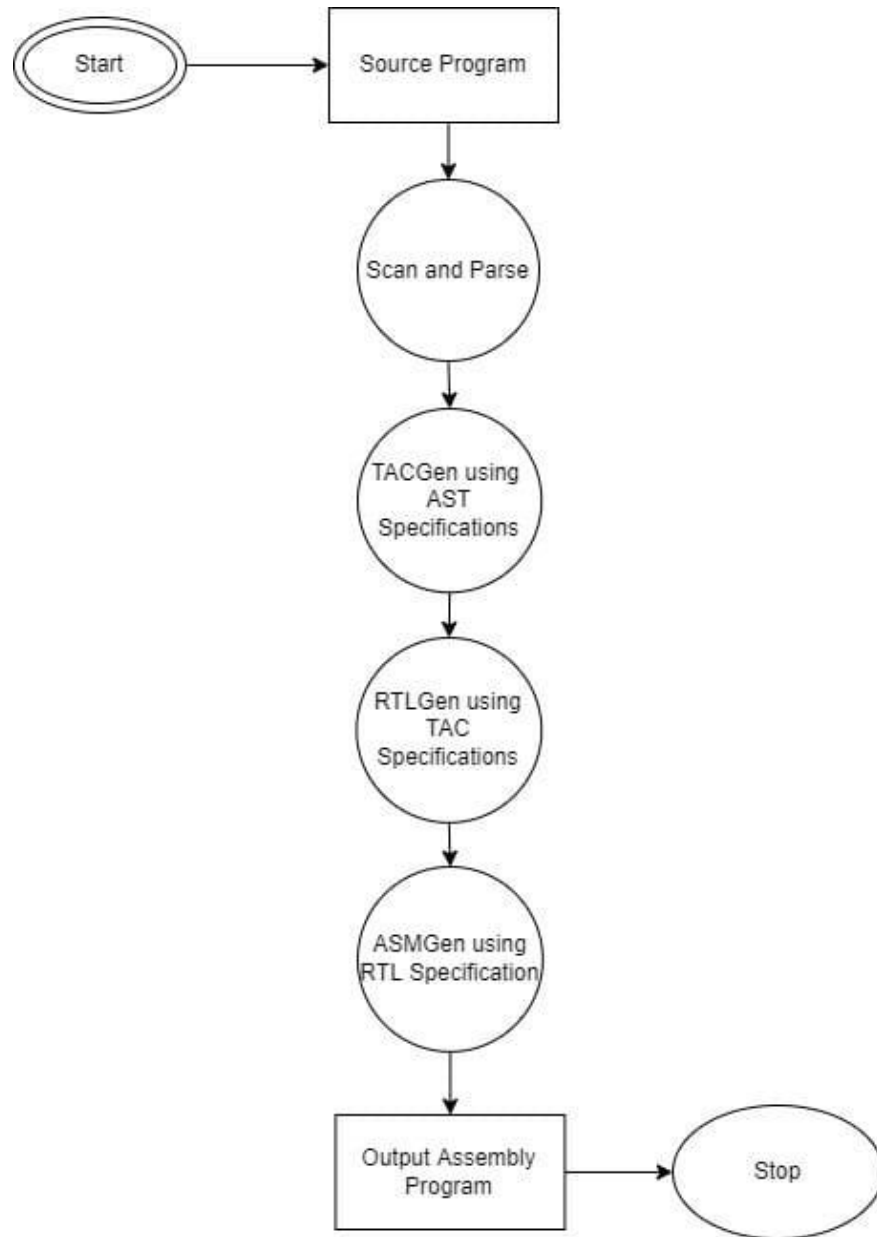
Fig 4.2- High level model

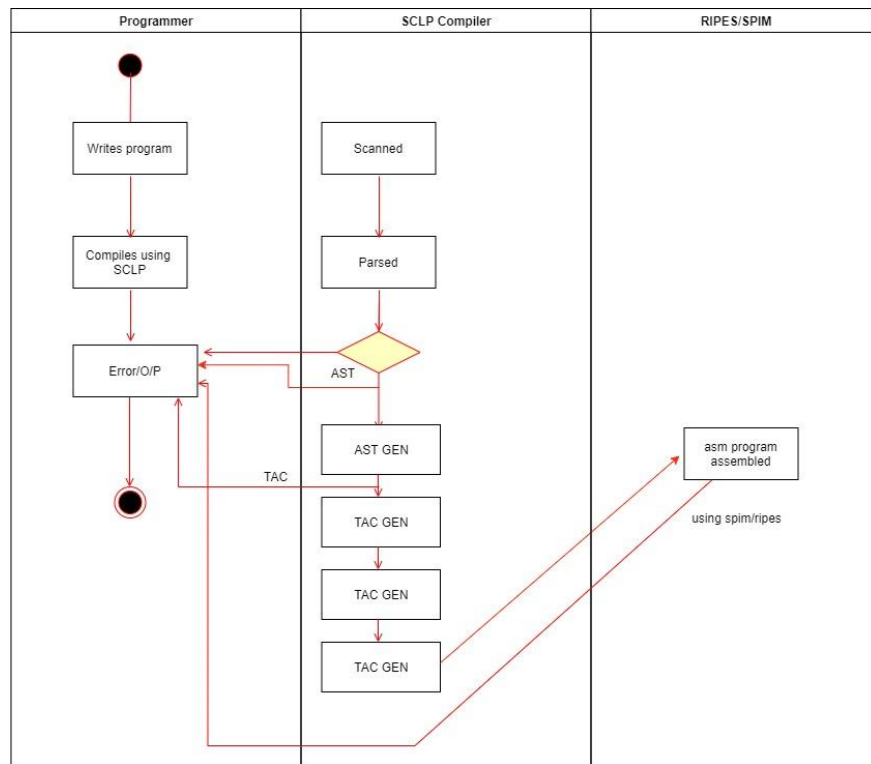
Description-

- The previous group tried to automate the process of generating Assembly program by writing RTL specifications and then generating Translation Rules from them, we are trying the same for the rest of the IR phases.
- We are going to write the AST specifications first and then generate the translation rules for AST. Then write the TransGen and TACGen and generate TAC from that.
- We are going to write the TAC specifications and then generate the translation rules for TAC. Then write the TransGen and TACGen and generate RTL from that.

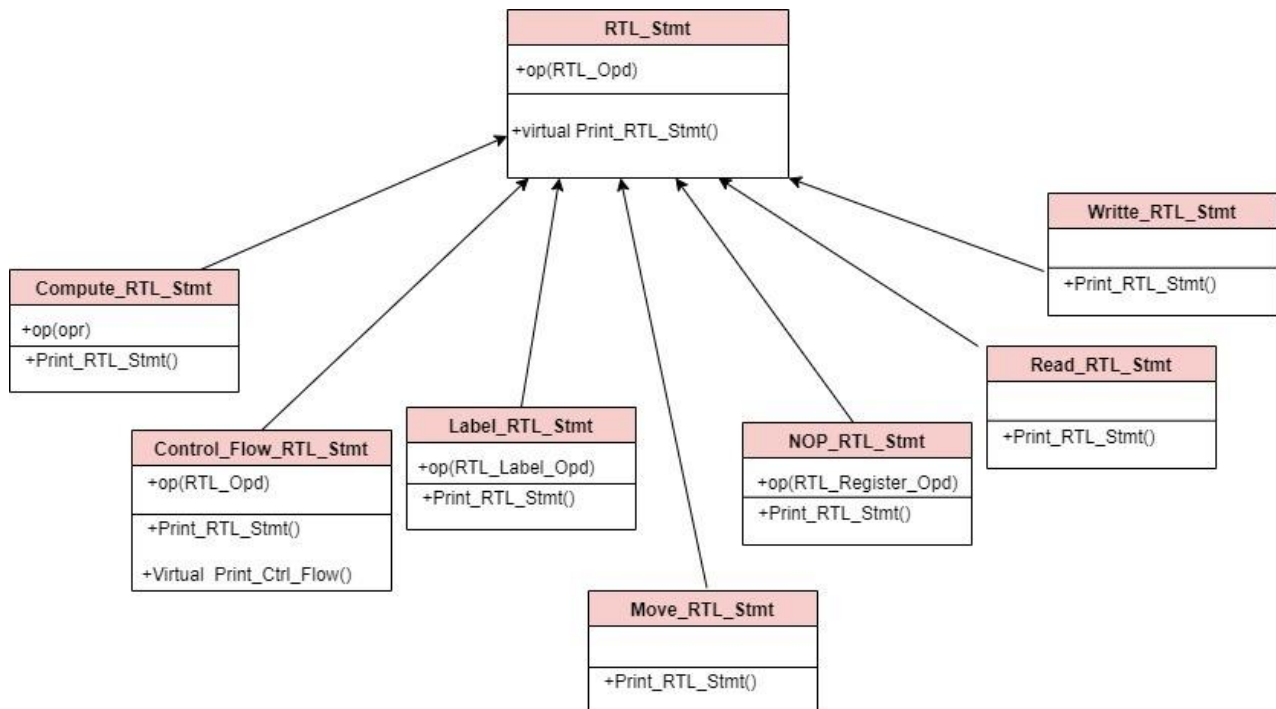
4.2 UML DIAGRAMS –

4.2.1 Data Flow Diagram

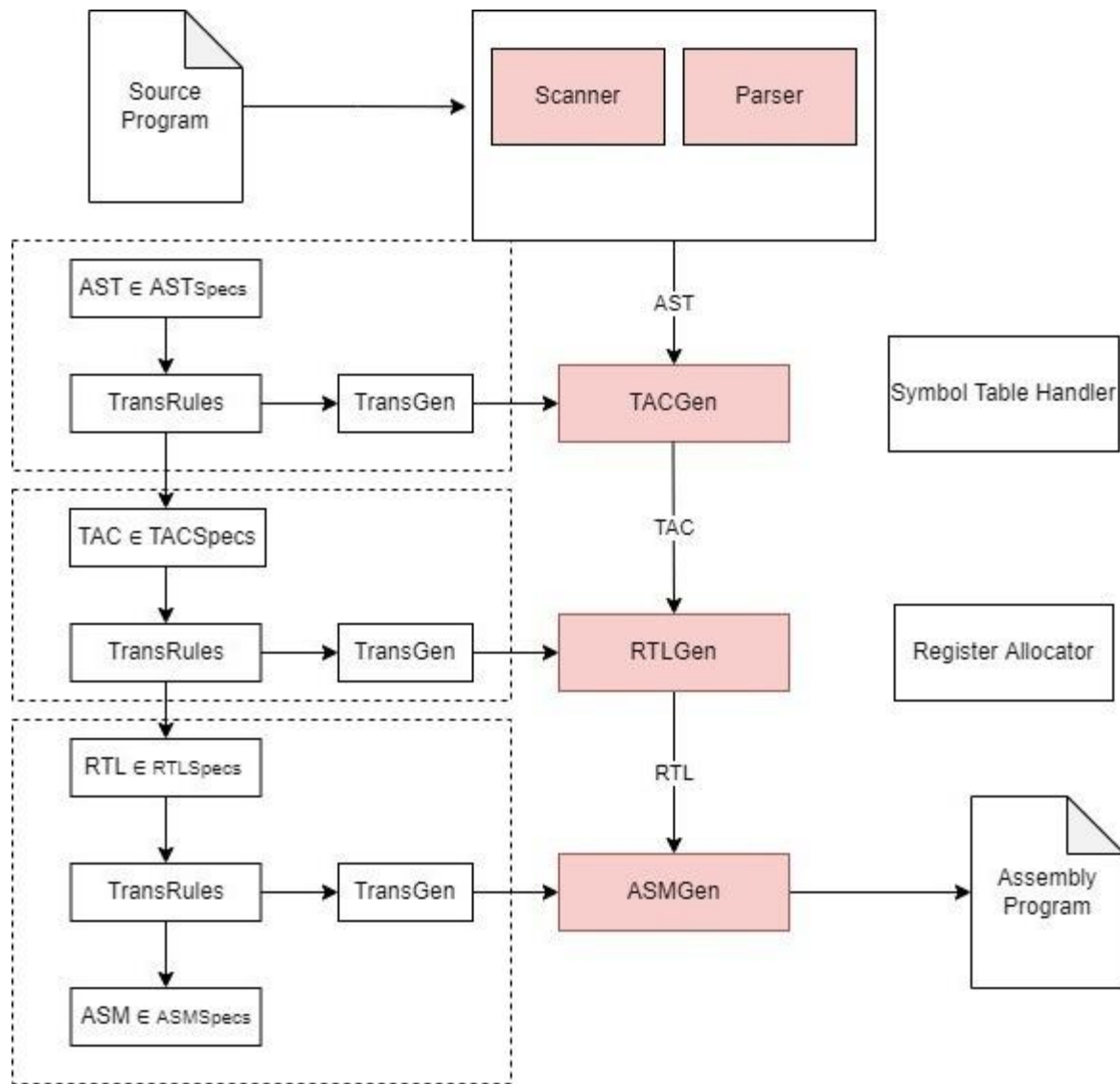




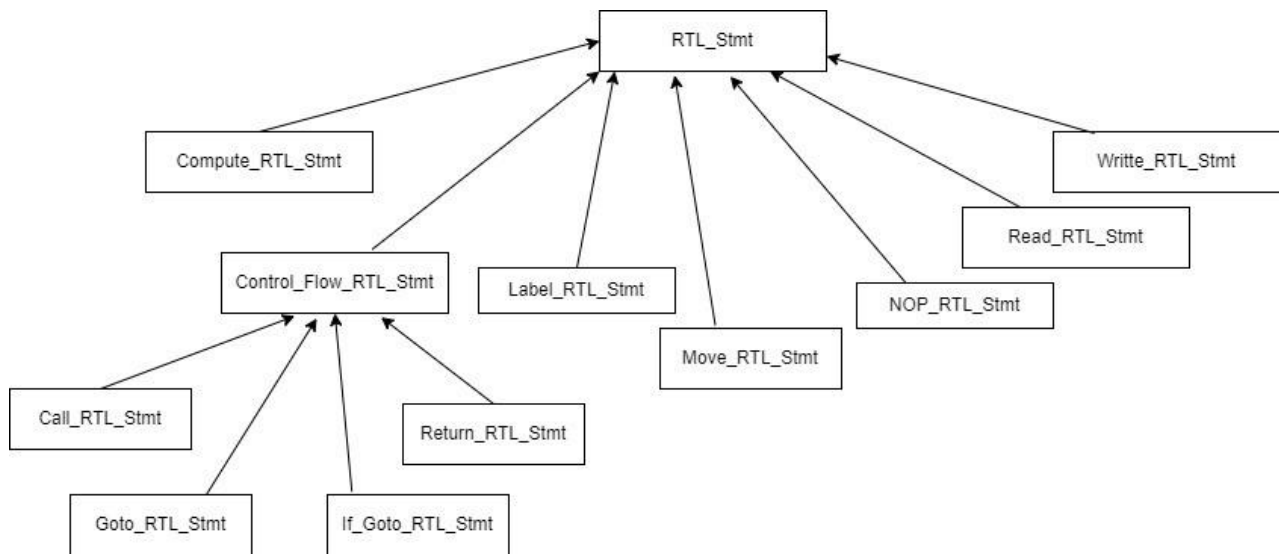
4.2.2 Activity Diagram



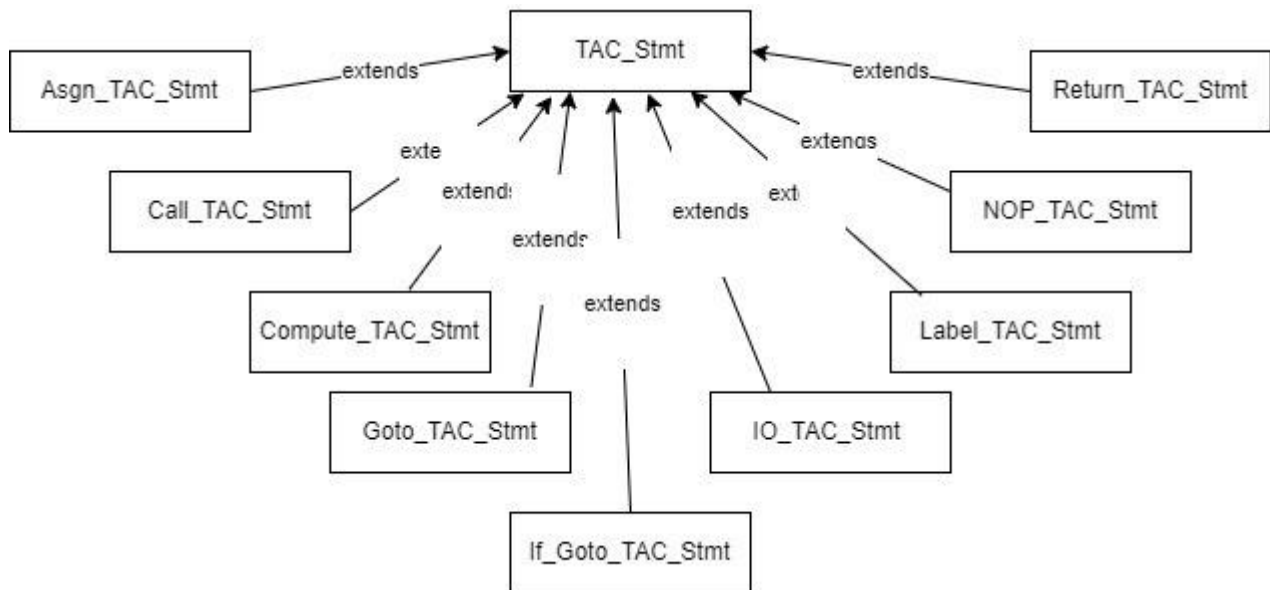
4.2.3 Class Diagram



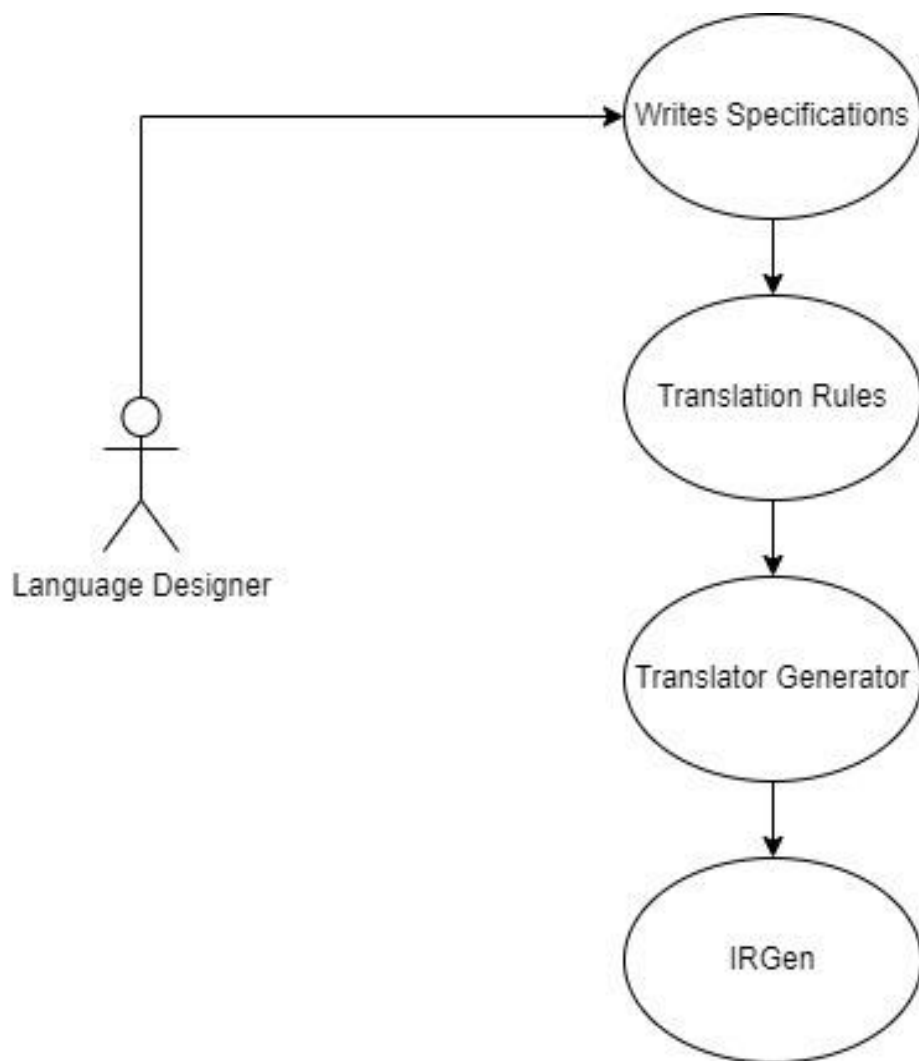
4.2.4 Component Diagram



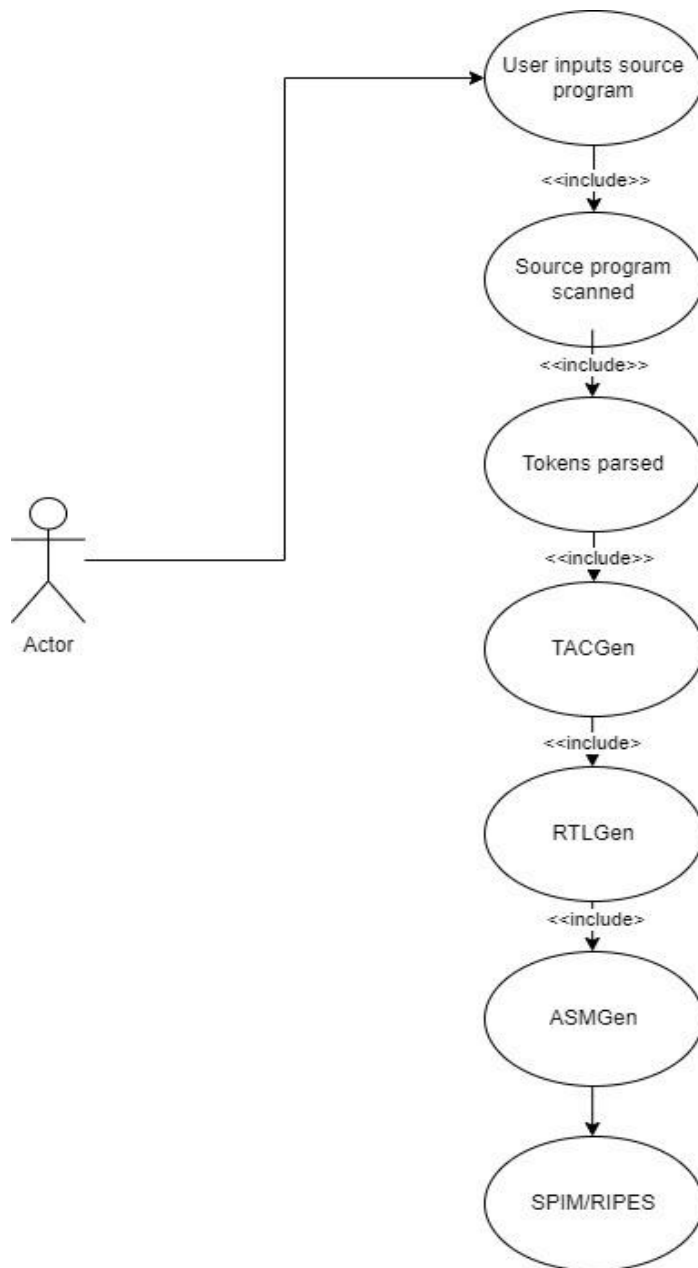
4.2.5 Class Diagram for RTL



4.2.6 Class Diagram for TAC



4.2.7 IR Generator Use Case Diagram



<<include>>

4.2.8 SCLP Use Case Diagram

5. IMPLEMENTATION ASPECTS

5.1 STUDY SCLP -

SCLP has 6 levels built atop each other.

Each level allows more and more functionalities incrementally.

Compiled each level .c programs to produce AST, TAC, RTL, ASM, Symbol Table of each code.

Eg. Level 4 code

```
1 void main()
2 {
3     int a,i;
4     a=5;
5     i=0;
6     while(i<a)
7     {
8         i=i+1;
9         print i;
10    }
11 }
```

TAC for the above code

```
bhagyashree@bhagyashree-Lenovo-Ideapad-320-15ISK:~/Documents/BTP$ ./sclp scodes/level4/code3.c -d --show-tac --sa-tac
**PROCEDURE: main
**BEGIN: Three Address Code Statements
    a_ = 5
    i_ = 0
Label0:
    temp0 = i_ < a_
    temp2 = ! temp0
    if(temp2) goto Label1
    temp1 = i_ + 1
    i_ = temp1
    write i_
    goto Label0
Label1:
**END: Three Address Code Statements
```

Executed codes of each level on each SCLP level and verified the results. Generated a report of the verification.

5.2 BUG REPORTS

The current implementation of SCLP had a handful of bugs which needed to be reported.

So far, 14 bugs have been found and reported.

Eg.

<pre>void main() { int a[5]; int b, c; a[0] = 5; a[1] = 4; a[2] = 3; a[3] = 2; a[4] = 1; c = 0; while(c < 5) { print "\n c: "; print c; print "\n"; b = a[c]; print b; print "\n"; c = c+1; } }</pre>	<pre>(spim) load "scodes/level6/code7.c.spim" (spim) run c: 0 5 c: 1 5 c: 2 2 c: 3 2 c: 4 1</pre>
---	--

The code on the left does not produce expected output. The bug was speculated and the source of the bug was found through the symbol table output and the stack of SPIM simulator.

5.3 SCLP WEB PAGE CHANGES

The SCLP web page needed updates. It wasn't up-to-date with the newer version of SCLP. It had a couple of errors which needed to be rectified.

Point - [Features of the language supported by Scip]

Line - [These sequences are limited to a length of 80]

Character sequences not limited to the length of 80. Accepts character sequences greater than the length of 80.

The changes and updates were reported, and the website was updated.

5.4 THOROUGH STUDY OF MIPS INSTRUCTIONS FOR SPIM

Studied MIPS instructions and registers.

Compiled numerous codes to investigate:

- how the registers are allocated.
- how data segment, register file, and the code segment used.

PC	=	004000c0	EPC	=	00000000	Cause	=	00000000	BadVAddr=	00000000	
Status	=	3000ff10	HI	=	00000000	LO	=	00000010			
General Registers											
R0 (r0)	=	00000000	R8 (t0)	=	00000010	R16 (s0)	=	00000000	R24 (t8)	=	00000000
R1 (at)	=	00000000	R9 (t1)	=	7ffffefe4	R17 (s1)	=	00000000	R25 (t9)	=	00000000
R2 (v0)	=	00000005	R10 (t2)	=	00000000	R18 (s2)	=	00000000	R26 (k0)	=	00000000
R3 (v1)	=	00000000	R11 (t3)	=	00000000	R19 (s3)	=	00000000	R27 (k1)	=	00000000
R4 (a0)	=	00000000	R12 (t4)	=	00000000	R20 (s4)	=	00000000	R28 (gp)	=	10008000
R5 (a1)	=	7ffff000	R13 (t5)	=	00000000	R21 (s5)	=	00000000	R29 (sp)	=	7ffffefe8
R6 (a2)	=	7ffff004	R14 (t6)	=	00000000	R22 (s6)	=	00000000	R30 (s8)	=	7ffffeff8
R7 (a3)	=	00000000	R15 (t7)	=	00000000	R23 (s7)	=	00000000	R31 (ra)	=	00400018
FIR	=	00009800	FCSR	=	00000000	FCCR	=	00000000	FEXR	=	00000000
FENR	=	00000000									
Double Floating Point Registers											
<div><div>quit</div><div>load</div><div>reload</div><div>run</div><div>step</div><div>clear</div><div>set value</div><div>print</div><div>breakpoints</div><div>help</div><div>terminal</div><div>mode</div></div>											
Text Segments											
[0x004000a4]	0x34020004	ori \$2, \$0, 4									; 42: li \$v0, 4
[0x004000a8]	0x34090004	ori \$9, \$0, 4									; 43: li \$t1, 4
[0x004000ac]	0x70494002	mul \$8, \$2, \$9									; 44: mul \$t0, \$v0, \$t1
[0x004000b0]	0x34020005	ori \$2, \$0, 5									; 45: li \$v0, 5
[0x004000b4]	0x23c9ffff	addi \$9, \$30, -4									; 46: la \$t1, -4(\$fp)
[0x004000b8]	0x01284822	sub \$9, \$9, \$8									; 47: sub \$t1, \$t1, \$t0
[0x004000bc]	0xad220000	sw \$2, 0(\$9)									; 48: sw \$v0, 0(\$t1)
[0x004000c0]	0x34020000	ori \$2, \$0, 0									; 49: li \$v0, 0
Data Segments											
DATA											
[0x10000000] ... [0x10020000]	0x00000000										
STACK											
[0x7ffffefe8]	0x00000006		0x00000007								
[0x7ffffeff0]	0x00000008		0x00000009		0x00000000		0x00400018				
KERNEL DATA											
[0x90000000]	0x78452020		0x74706563		0x206e6f69		0x636f2000				
[0x90000010]	0x72727563		0x61206465		0x6920646e		0x726f6e67				
[0x90000020]	0x000a6465		0x495b2020		0x7265746e		0x74707572				
[0x90000030]	0x2000205d		0x4c545b20		0x20005d42		0x4c545b20				
[0x90000040]	0x20005d42		0x4c545b20		0x20005d42		0x64415b20				
[0x90000050]	0x73657264		0x72652073		0x20726f72		0x69206e69				
[0x90000060]	0x2f74736e		0x61746164		0x74656620		0x205d6863				

Fig 5.1- MIPS Instructions Working

5.5 FORMALIZED ASM INSTRUCTIONS FOR SPIM

The predecessors of the project had created specifications document for ASM which needed cleaning and deciding one proper syntax for all the specifications.

One universal syntax was decided to be used in all the specifications documents and ASM was formalized.

A few Load Instructions from the specifications document:

$$\begin{array}{lllll}
I \in \{mv, cmp, cf\} & \{mv, cmp, cf\} : \tau_C^I & & & \\
O \in \{int, dub, wrd, addr, lab, gpr, fpr, off\} & \{lab\} : \tau_C^O & \{gpr\} : \tau_R^O & \{fpr\} : \tau_{FPR}^O &
\end{array}$$

2 Load Instructions

$$[Load\ Integer\ Immediate] \frac{S(li, o1, o2) : \tau_{mv}^S \quad o1 : \tau_{gpr}^S \quad o2 : \tau_{int}^S}{H; D; C; R \vdash S(li, o1, o2) \rightarrow H; D; C; R[o1 \mapsto o2]} R : \tau_{gpr}^S \quad (1)$$

$$[Load\ Float\ Immediate] \frac{S(li.d, o1, o2) : \tau_{mv}^S \quad o1 : \tau_{fpr}^S \quad o2 : \tau_{dub}^S}{H; D; C; R \vdash S(li.d, o1, o2) \rightarrow H; D; C; R[o1 \mapsto o2]} R : \tau_{fpr}^S \quad (2)$$

$$[Load\ Word] \frac{S(lw, o1, o2) : \tau_{mv}^S \quad o1 : \tau_{gpr}^S \quad o2 : \tau_{addr}^S}{H; D; C; R \vdash S(lw, o1, o2) \rightarrow H; D; C; R[o1 \mapsto D[o2 : o2 + 4]]} R : \tau_{gpr}^S \quad (3)$$

$$[Load\ Double] \frac{S(ld, o1, o2) : \tau_{mv}^S \quad o1 : \tau_{fpr}^S \quad o2 : \tau_{off}^S \quad o3 : \tau_{gpr}^S}{H; D; C; R \vdash S(ld, o1, o2) \rightarrow H; D; C; R[o1 \mapsto D[R[o3] + o2 : R[o3] + o2 + 8]]} R : \tau_{fpr}^S \quad (4)$$

$$[Load\ Address] \frac{S(la, o1, o2) : \tau_{mv}^S \quad o1 : \tau_{gpr}^S \quad o2 : \tau_{addr}^S}{H; D; C; R \vdash S(la, o1, o2) \rightarrow H; D; C; R[o1 \mapsto o2]} R : \tau_{gpr}^S \quad (5)$$

5.6 THOROUGH STUDY OF RISC-V INSTRUCTIONS FOR RIPES

Studied the instructions and compared them with the MIPS instructions. Discussed the mapping of MIPS register and RISCV registers.

RIPES simulator showing the assembly editor, the assembled code.

```

1 # This example demonstrates an implementation of the multiplication of two
2 # complex numbers z = 1 + 3i, w = 5 + 4i.
3
4 .data
5 aa: .word 1 # Real part of z
6 bb: .word 3 # Imag. part of z
7 cc: .word 5 # Real part of w
8 dd: .word 4 # Imag part of w
9 str: .string " + i* "
10
11 .text
12 main:
13     lw a0, aa
14     lw a1, bb
15     lw a2, cc
16     lw a3, dd
17
18     # Do complex multiplication of numbers a0-a3
19     jal complexMul
20     mv t0, a1 # Move imaginary value to t0
21     mv a0, a0 # Move real value to a1
22
23     # Print real value (in a0) by setting ecall argument to 1
24     li a7, 1
25     ecall
26
27     # Print delimiter string (pointer in a0) by setting ecall argument to 4
28     la a0, str
29     li a7, 4
30     ecall
31
32     # Print imaginary value (in a0) by setting ecall argument to 1
33     mv a0, t0 # Move imaginary value to a1
34     li a7, 1
35     ecall
36
37     # Exit program
38     li a7, 10
39     ecall
40
41 myMult:
42     li t0, 32 # Iteration variable
43     li t3, 0 # initialize temporary product register to 0
44
45 start:
46     mv t1, a1 # move multiplier to temporary register
47     andi t1, t1, 1 # mask first bit
48     beq t1, x0, shift
49     add t3, t3, a0
50
51 shift:
52     slli a0, a0, 1
53     srai a1, a1, 1 # make an arithmetic right shift for signed multiplication
54     addi t0, t0, -1 # decrement loop index
55     bnez t0, start # branch if loop index is not 0
56     mv a0, t3 # move final product to result register

```

```

0: 10000517 auipc x10 0x10000
4: 00052503 lw x10 0 x10
8: 10000597 auipc x11 0x10000
c: ffc5a583 lw x11 -4 x11
10: 10000617 auipc x12 0x10000
14: ff862603 lw x12 -8 x12
18: 10000697 auipc x13 0x10000
1c: ff46a683 lw x13 -12 x13
20: 008000ef jal x1 104 <complexMul>
24: 00058293 addi x5 x11 0
28: 00050513 addi x10 x10 0
2c: 00100893 addi x17 x0 1
30: 00000073 ecall
34: 10000517 auipc x10 0x10000
38: fdc50513 addi x10 x10 -36
3c: 00400893 addi x17 x0 4
40: 00000073 ecall
44: 00028513 addi x10 x5 0
48: 00100893 addi x17 x0 1
4c: 00000073 ecall
50: 000a0893 addi x17 x0 10
54: 00000073 ecall

00000058 <myMult>:
58: 02000293 addi x5 x0 32
5c: 00000e13 addi x28 x0 0

00000060 <start>:
60: 00053313 addi x6 x11 0
64: 00137313 andi x6 x6 1
68: 00030463 beq x6 x0 8 <shift>
6c: 00ae0e33 add x28 x28 x10

00000070 <shift>:
70: 00151513 slli x10 x10 1
74: 4015d593 srai x11 x11 1
78: fff28293 addi x5 x5 -1
7c: fe0292e3 bne x5 x0 -28 <start>
80: 000e0513 addi x10 x28 0
84: 00000067 jalr x0 x1 0

00000088 <complexMul>:
88: fe410113 addi x2 x2 -28
8c: 00012c23 sw x0 24 x2
90: 00012a23 sw x0 20 x2
94: 00112823 sw x1 16 x2
98: 00a12623 sw x10 12 x2
9c: 00b12423 sw x11 8 x2
a0: 00c12223 sw x12 4 x2
a4: 00d12023 sw x13 0 x2
a8: 00060593 addi x11 x12 0
ac: fadff0ef jal x1 -84 <myMult>
b0: 00a12a23 sw x10 20 x2
b4: 00812503 lw x10 8 x2
b8: 00012503 lw x11 0 x2
bc: f9dff0ef jal x1 -100 <myMult>

```

Name	Alias	Value
x0	zero	0x00000000
x1	ra	0x00000000
x2	sp	0x7fffffff0
x3	gp	0x10000000
x4	tp	0x00000000
x5	t0	0x00000000
x6	t1	0x00000000
x7	t2	0x00000000
x8	s0	0x00000000
x9	s1	0x00000000
x10	a0	0x00000000
x11	a1	0x00000000
x12	a2	0x00000000
x13	a3	0x00000000
x14	a4	0x00000000
x15	a5	0x00000000
x16	a6	0x00000000
x17	a7	0x00000000
x18	s2	0x00000000
x19	s3	0x00000000
x20	s4	0x00000000
x21	s5	0x00000000
x22	s6	0x00000000
x23	s7	0x00000000
x24	s8	0x00000000
x25	s9	0x00000000
x26	s10	0x00000000
x27	s11	0x00000000

Display type:

Fig 5.2, 5.3- Ripes Simulator showing the memory and the registers used.

Memory viewer						Memory map		
Address	Word	Byte 0	Byte 1	Byte 2	Byte 3	Name	Size	Range
0x00000070	0x00151513	0x13	0x15	0x15	0x00	.text	260	0x00000000 - 0x00000104
0x0000006c	0x00a0e333	0x33	0x0e	0xae	0x00	.data	23	0x10000000 - 0x10000017
0x00000068	0x00030463	0x63	0x04	0x03	0x00	.bss	0	0x11000000 - 0x11000000
0x00000064	0x00137313	0x13	0x73	0x13	0x00			
0x00000060	0x00058313	0x13	0x83	0x05	0x00			
0x0000005c	0x00000e13	0x13	0x0e	0x00	0x00			
0x00000058	0x02000293	0x93	0x02	0x00	0x02			
0x00000054	0x00000073	0x73	0x00	0x00	0x00			
0x00000050	0x00a00893	0x93	0x08	0xae	0x00			
0x0000004c	0x00000073	0x73	0x00	0x00	0x00			
0x00000048	0x00100893	0x93	0x08	0x10	0x00			
0x00000044	0x00028513	0x13	0x85	0x02	0x00			
0x00000040	0x00000073	0x73	0x00	0x00	0x00			
0x0000003c	0x00400893	0x93	0x08	0x40	0x00			
0x00000038	0xfdc50513	0x13	0x05	0xc5	0xfd			

5.7 FORMALIZED ASM SPECIFICATIONS FOR RIPES

Created specifications document for RISC-V instructions for RIPES

Formalized specifications few compute instructions

Compute Instructions

$$[Add\ ints] \frac{S(add, o1, o2, o3) : \tau_{cmp}^S \quad o1 : \tau_{gpr}^S \quad o2 : \tau_{gpr}^S \quad o3 : \tau_{gpr}^S}{H; D; C; R \vdash S(add, o1, o2) \rightarrow H; D; C; R[o1 \mapsto R[o2] + R[o3]]} \quad R : \tau_{gpr}^S \quad (6)$$

$$[Add\ ints] \frac{S(addi, o1, o2, o3) : \tau_{cmp}^S \quad o1 : \tau_{gpr}^S \quad o2 : \tau_{gpr}^S \quad o3 : \tau_{int}^S}{H; D; C; R \vdash S(addi, o1, o2) \rightarrow H; D; C; R[o1 \mapsto R[o2] + o3]} \quad R : \tau_{gpr}^S \quad (7)$$

$$[Sub\ ints] \frac{S(sub, o1, o2, o3) : \tau_{cmp}^S \quad o1 : \tau_{gpr}^S \quad o2 : \tau_{gpr}^S \quad o3 : \tau_{gpr}^S}{H; D; C; R \vdash S(sub, o1, o2, o3) \rightarrow H; D; C; R[o1 \mapsto R[o2] - R[o3]]} \quad R : \tau_{gpr}^S \quad (8)$$

$$[Sub\ ints] \frac{S(sub, o1, o2, o3) : \tau_{cmp}^S \quad o1 : \tau_{gpr}^S \quad o2 : \tau_{gpr}^S \quad o3 : \tau_{int}^S}{H; D; C; R \vdash S(sub, o1, o2, o3) \rightarrow H; D; C; R[o1 \mapsto R[o2] - o3]} \quad R : \tau_{gpr}^S \quad (9)$$

$$[Multiply\ ints] \frac{S(mul, o1, o2, o3) : \tau_{cmp}^S \quad o1 : \tau_{gpr}^S \quad o2 : \tau_{gpr}^S \quad o3 : \tau_{gpr}^S}{H; D; C; R \vdash S(mul, o1, o2, o3) \rightarrow H; D; C; R[o1 \mapsto R[o2] \times R[o3]]} \quad R : \tau_{gpr}^S \quad (10)$$

$$[Divide\ ints] \frac{S(div, o1, o2, o3) : \tau_{cmp}^S \quad o1 : \tau_{gpr}^S \quad o2 : \tau_{gpr}^S \quad o3 : \tau_{gpr}^S}{H; D; C; R \vdash S(div, o1, o2, o3) \rightarrow H; D; C; R[o1 \mapsto R[o2] \div R[o3]]} \quad R : \tau_{gpr}^S \quad (11)$$

5.8 SPECIFICATIONS DOCUMENTS FOR RTL

Currently working on the specifications document for RTL for SPIM and for RIPES.

Eg. RTL specification of Li (Load Immediate) Instruction

$$[Load\ Integer\ Immediate] \frac{R(iLoad, o1, o2) : \tau_{mv}^R \quad o1 : \tau_{reg}^R \quad o2 : \tau_{int}^R}{R \vdash S(iLoad, o1, o2) \rightarrow R[o1 \mapsto o2]} \quad R : \tau_{gpr}^S \quad (12)$$

6. TECHNOLOGY

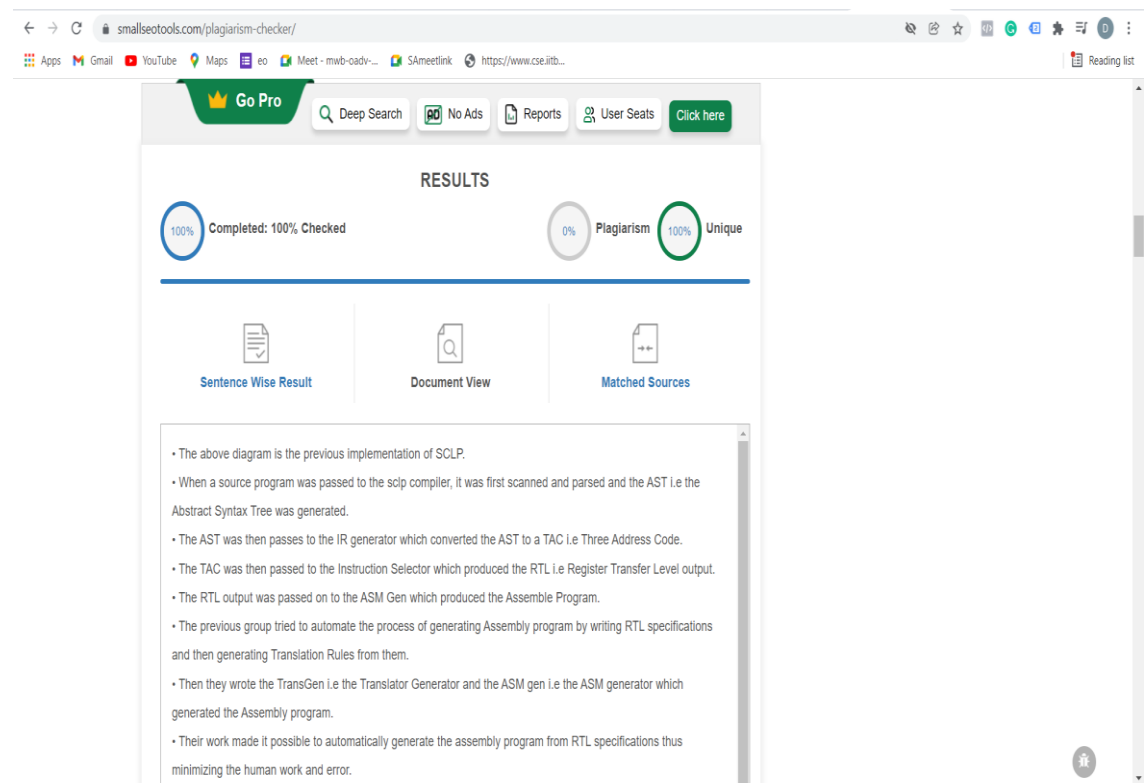
- To write the translator generators we are using the languages C++ and python.
- We are using lex and yacc as lexical analyzers and parser generators.
- Yacc is a computer program for the Unix operating system and Lex is a computer program that generates lexical analyzers.
- The current processor being used is MIPS processor and the simulator is SPIM.
- We are going to change the processor to RISC-V and the simulator to RIPS.
- The software system LaTeX is used for document preparation.

7. CONCLUSIONS & FUTURE WORK

Thus, we conclude that our project is the development of SCLP compiler by automating the generation of IR phases like AST, TAC and RTL. The future work would be implementing generators that will construct interpreters and thus providing virtual machines for the IR phases i.e AST, TAC and RTL.

8. APPENDIX

A: PLAGIARISM REPORT



The screenshot shows a web browser window with the URL smallseotools.com/plagiarism-checker/. The page features a 'Go Pro' button, a search bar, and several icons for 'Deep Search', 'No Ads', 'Reports', 'User Seats', and a 'Click here' button. The main section is titled 'RESULTS' and displays two circular progress indicators: 'Completed: 100% Checked' and '0% Plagiarism 100% Unique'. Below these indicators are three tabs: 'Sentence Wise Result', 'Document View', and 'Matched Sources'. The 'Sentence Wise Result' tab is selected, showing a list of sentences with bullet points. The sentences are as follows:

- The above diagram is the previous implementation of SCLP.
- When a source program was passed to the scp compiler, it was first scanned and parsed and the AST i.e the Abstract Syntax Tree was generated.
- The AST was then passes to the IR generator which converted the AST to a TAC i.e Three Address Code.
- The TAC was then passed to the Instruction Selector which produced the RTL i.e Register Transfer Level output.
- The RTL output was passed on to the ASM Gen which produced the Assemble Program.
- The previous group tried to automate the process of generating Assembly program by writing RTL specifications and then generating Translation Rules from them.
- Then they wrote the TransGen i.e the Translator Generator and the ASM gen i.e the ASM generator which generated the Assembly program.
- Their work made it possible to automatically generate the assembly program from RTL specifications thus minimizing the human work and error.

9. REFERENCES

- [1] A. V. Aho, Alfred V. Aho, Ravi Sethi, Jeffrey D. Ullman, J. D. Ullman, 1986. Compilers: Principles, Techniques, and Tools. Addison-Wesley.
- [2] Tofte. M. 2012 compiler generators : what they can do , what they might do and what they will never do. Springer Science & Business Media.
- [3] William Waite, “A Complete Specification of a Simple Compiler” Available- http://eli-project.sourceforge.net/pascal_html/pascal-.html#s2.1
- [4] Dr.Uday Khedkar, “Sclp: A Language Processor for a Small C-like Language” Available- <https://www.cse.iitb.ac.in/~uday/sclp-web/>
- [5] Central Connecticut State University, “Programmed Introduction to MIPS Assembly Language” Available- <https://chortle.ccsu.edu/AssemblyTutorial/>
- [6] “SPIM Quick Reference”, Available- http://www.cse.iitm.ac.in/~krishna/cs3300/spim_ref.html#traps
- [7] Charles Prince, “MIPS IV Instruction Set”, Available- <https://www.cs.cmu.edu/afs/cs/academic/class/15740-f97/public/doc/mips-isa.pdf>
- [8] James.R.Larus, “SPIM Documentation”, Available- <https://www.dsi.unive.it/~archit/LAB/spim.htm>
- [9] Portland State University, “Instruction Set Architecture”, Available- <https://web.cecs.pdx.edu/~harry/riscv/RISCV-Summary.pdf>
- [10] Andrew Waterman, Krste Asanovi´, “The RISC-V Instruction Set Manual”, Available- <https://riscv.org/wp-content/uploads/2017/05/riscv-spec-v2.2.pdf>
- [11] “GitHub - mortbopet/Ripes: A graphical processor simulator and assembly editor for the RISC-V ISA”, Available- <https://github.com/mortbopet/Ripes>