Group 14

Generating Compiler Phases from Specifications of Intermediate Representations (IRs)

Sponsored By : IIT Bombay

Internal mentor : Dr. Chhaya Gosavi
External mentor : Dr. Uday Khedkar (IITB Professor)

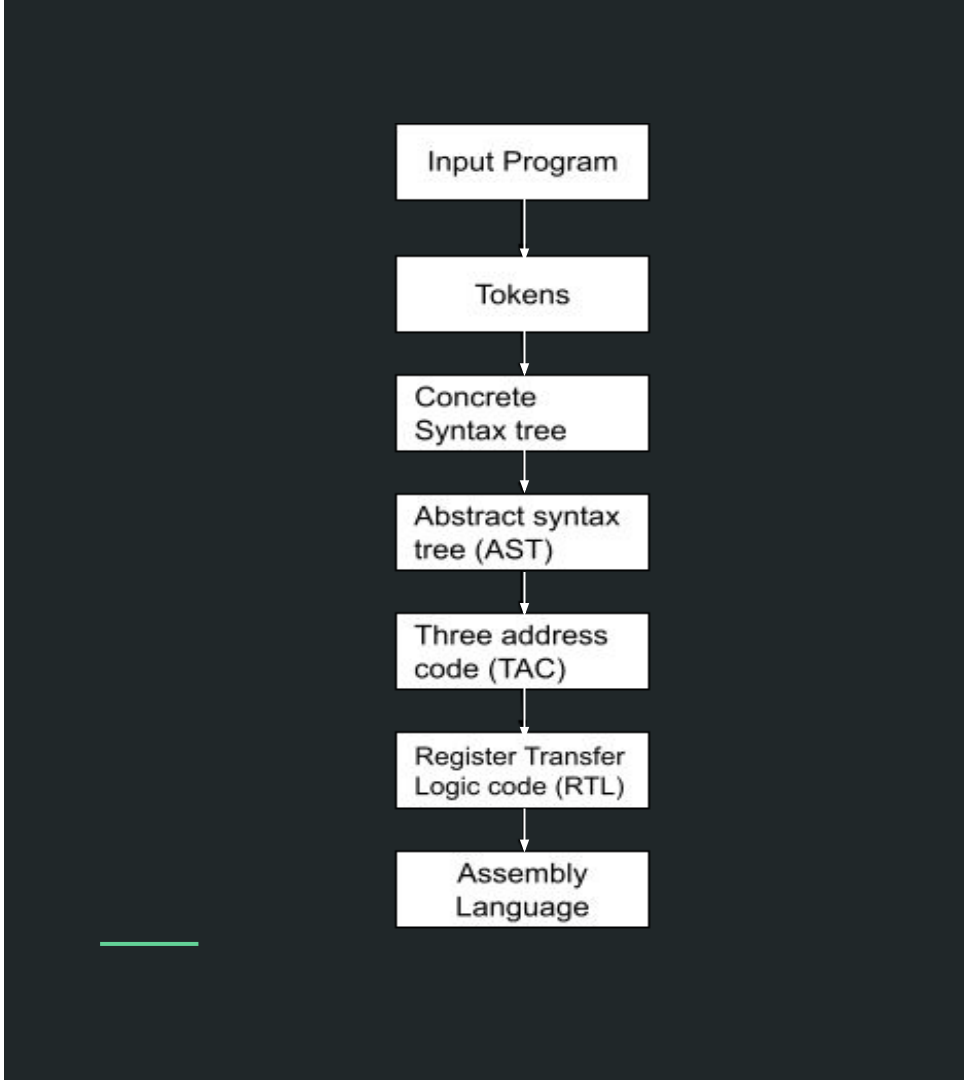
Group members - Sai Ghule (4332), Bhagyashree Rane (4371), Shravasti Deore (4916)

Agenda

- Problem Statement
- What is SCLP?
- Objectives for the Project
- System Architecture
- Implementation Aspects
- Translation Rules
- Division of work done
- Progress
- Technology
- Future Enhancements
- Documents and Process Followed
- References

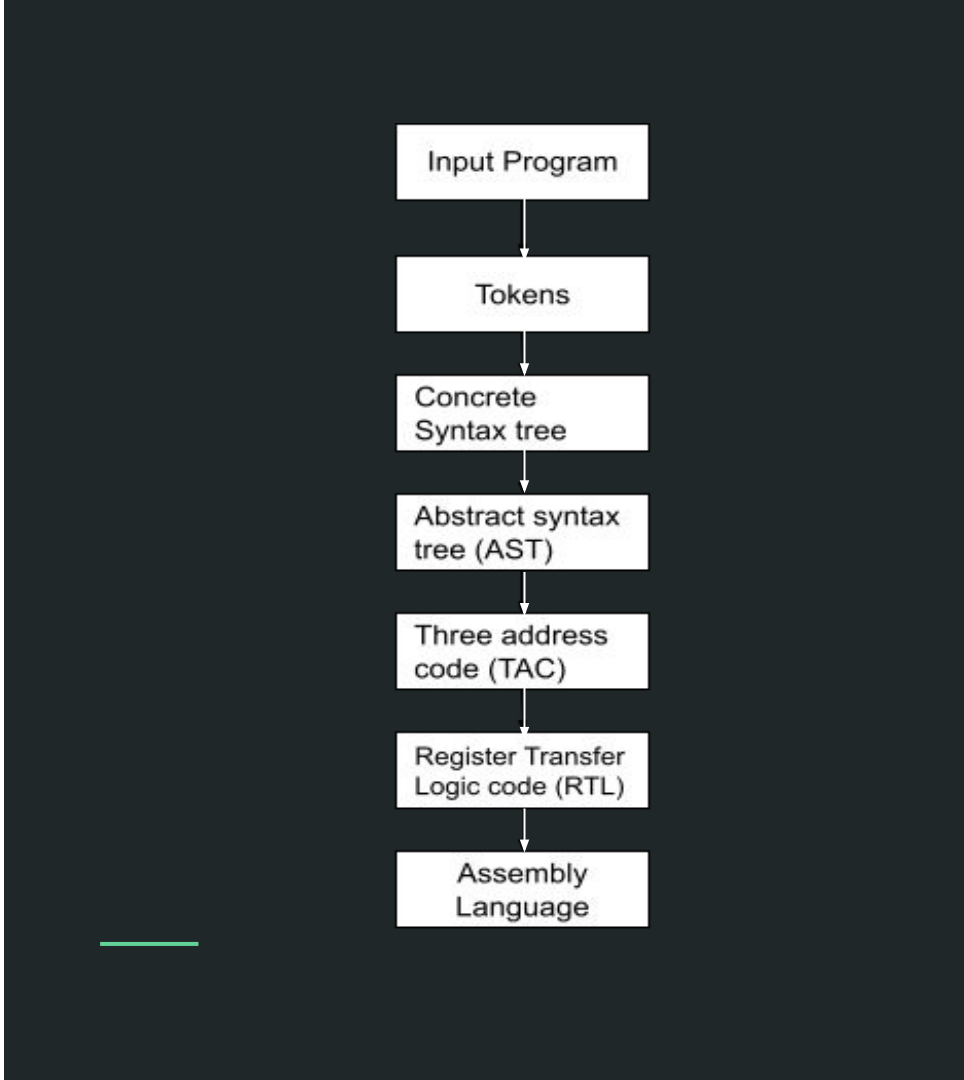
Problem Statement

Generating Compiler Phases from Specifications of Intermediate Representations of SCLP



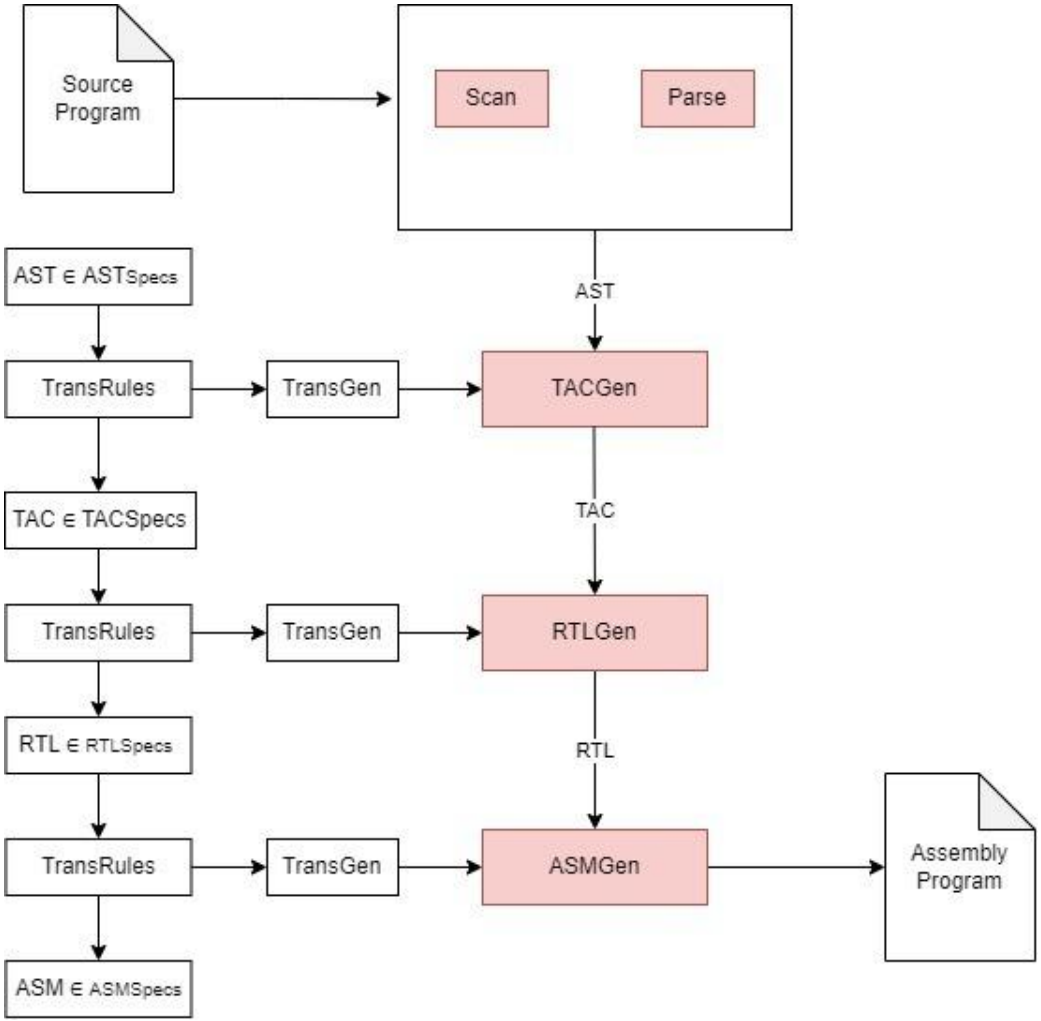
SCLP Compiler

SCLP is a language processor for a small C-like language implemented for UG courses at IIT Bombay.



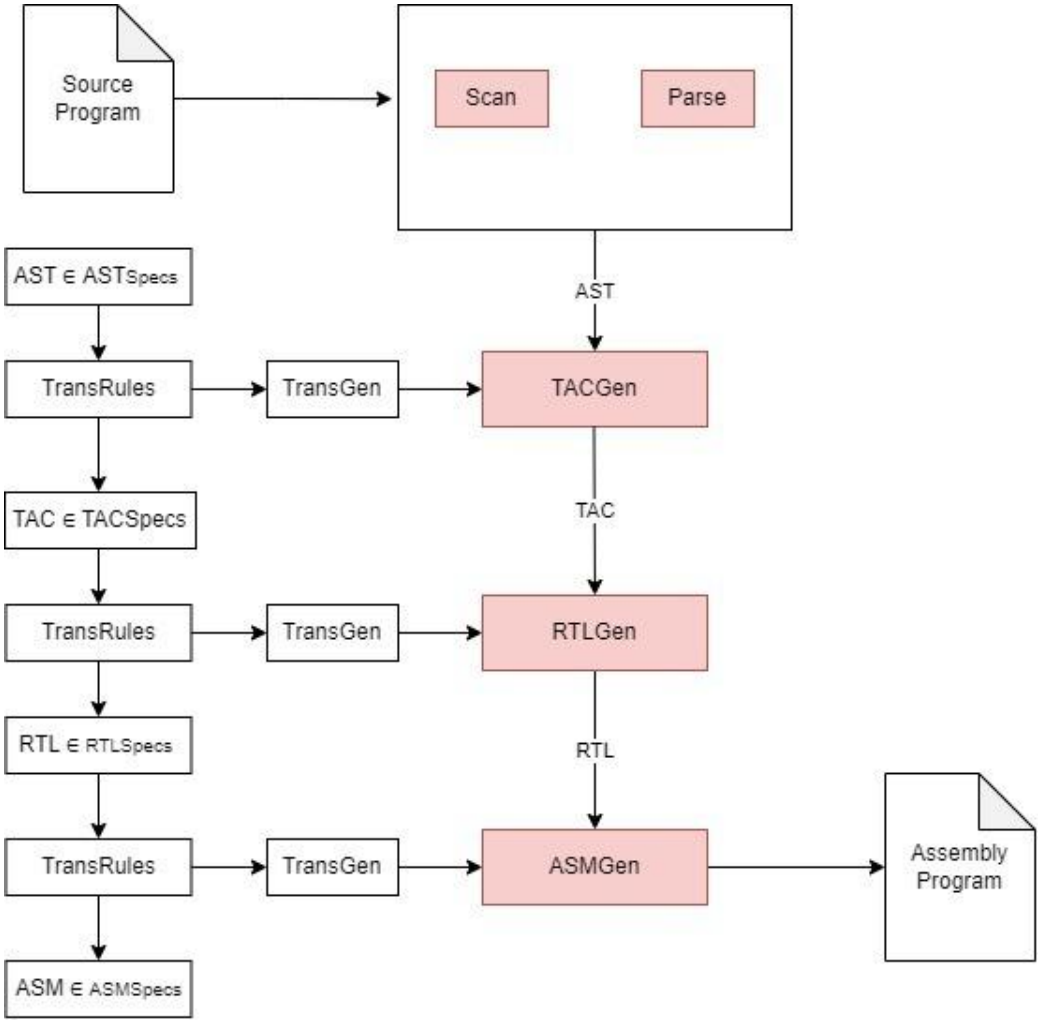
Objectives

- To make the already existing ASM specification documents cleaner.
- Create specification documents for AST and TAC.
- Generate Translation Rules from specifications of AST and TAC.
- Write TransGen for TACGen and RTLGen.
- ASMGen generated for a RISC processor using RIPES simulator.
- Integrate the work to current SCLP implementation.



System Architecture

High Level Design



Low level Design

- Change the simulator from spim to ripes
  - Specifications for SPIM
  - Specifications for RIPES
  - Specifications for RTL (need to study how RTL will change for ripes, can we keep the same RTL?)
  - Translation rules from RTL to ASM
  - Machine readable form of the specifications and translation rules
  - Implementation of the translator generator

Low level Design

- Generate TAC generator
  - Specifications for AST
  - Specifications for TAC
  - Translation rules from AST to TAC
  - Machine readable form of the specs and translation rules
  - Implementation of the translator generator

Low level Design

- Generate RTL generator
  - Specifications for TAC
  - Specifications for RTL
  - Translation rules for converting TAC to RTL
  - Machine readable form of the specs and translation rules
  - Implementation of the translator generator

Implementation Aspects

- Studied the SCLP compiler, various IRs and levels.
- Delivered bug reports, analysed SCLP and the SCLP web page and thus suggested changes.
- Studied the formalised ASM instructions written by the previous group.
- Suggested ways to improve the said formalization, making it more precise.
- Changed the specifications syntax of instructions to our suggested syntax.
- Studies RIPES simulator
- Researched on the differences between RISCV and MIPS processor which helped in studying the differences between spim and ripes simulator.
- Researched on the ways to write RTL specifications using Ripes.
- Studied and rectified specification translator provided by previous group
- Researched for ways to change the syntax of translation rules
- Came up with multiple ideas to do the same

Original Translation Rules Syntax

```
fun gen_asm (rstmt: Compute_RTL_Stmt ) astmts: ASM_Stmt list =
        astmts : ASM_Stmt list = []
        case rstmt.op of
                | rtl_and | rtl_or | rtl_add | rtl_imm_add | rtl_sub | rtl_mult
                | rtl_div | rtl_add_d | rtl_sub_d | rtl_mult_d |  rtl_div_d
                | rtl_slt_d | rtl_sle_d | rtl_sgt_d | rtl_sge_d | rtl_sne_d | rtl_seq_d  =>
                        astmts.append(ASM(get_asm_op(rstmt.op), (opd)rstmt.opd1, (opd)rstmt.opd2, (opd)rstmt.res))

                | rtl_slt | rtl_sle | rtl_sgt | rtl_sge | rtl_sne | rtl_seq =>
                        astmts.append(ASM(get_asm_op(rstmt.op), (opd)rstmt.opd1, (opd)rstmt.opd2, NULL))

                | rtl_not | rtl_uminus | rtl_uminus_d =>
                        astmts.append(ASM(get_asm_op(rstmt.op), (opd)rstmt.opd1, NULL, (opd)rstmt.res))
```

Original Translation Rules Syntax

- Written in SML(Standard Meta Language)
- A newer, flexible syntax was expected
- The translation rules supposed to be closer in syntax to the paper specifications
- The types of operands and other details to be mentioned in the translation rules itself

A few suggested changes in the syntax

```
move:
  rule {rtl_move}

  antecedent
  {
    RTL(iLoad, o1, o2, o3):Move_RTL_Stmt
    o1 : reg
    o2 : null
    o3 : reg
  }

  consequent
  {
    ASM(li, o1, o2, o3):Move_ASM_Stmt
    o1 : reg
    o2 : null
    o3 : reg
  }
```

A few suggested changes in the syntax

```
gen_asm compute <astmts: ASM_Stmt list>:
  rule {rtl_and | rtl_or | rtl_add | rtl_imm_add | rtl_sub | rtl_mult
      | rtl_div | rtl_add_d | rtl_sub_d | rtl_mult_d |  rtl_div_d
      | rtl_slt_d | rtl_sle_d | rtl_sgt_d | rtl_sge_d | rtl_sne_d | rtl_seq_d}
    antecedant {rstmt: Compute_RTL_Stmt}
    consequent {astmts.append(ASM(get_asm_op(rstmt.op), (opd)rstmt.opd1, (opd)rstmt.opd2, (opd)rstmt.res))}

  rule {rtl_slt | rtl_sle | rtl_sgt | rtl_sge | rtl_sne | rtl_seq}
    antecedant {rstmt: Compute_RTL_Stmt}
    consequent {astmts.append(ASM(get_asm_op(rstmt.op), (opd)rstmt.opd1, (opd)rstmt.opd2, NULL))}

  rule {rtl_not | rtl_uminus | rtl_uminus_d}
    antecedant {rstmt: Compute_RTL_Stmt}
    consequent {astmts.append(ASM(get_asm_op(rstmt.op), (opd)rstmt.opd1, NULL, (opd)rstmt.res))}
```

A few suggested changes in the syntax

```
rule {rtl_and | rtl_or | rtl_add | rtl_imm_add | rtl_sub | rtl_mult
            | rtl_div | rtl_add_d | rtl_sub_d | rtl_mult_d |  rtl_div_d
            | rtl_slt_d | rtl_sle_d | rtl_sgt_d | rtl_sge_d | rtl_sne_d | rtl_seq_d}

    antecedent
    {
        RTL(iLoad, o1, o2, o3) = rstmt : Compute_RTL_Stmt
        o1 = rstmt.opd1 : RTL_Reg
        o2 = rstmt.opd2 : RTL_Reg
        o3 = rstmt.opd3 : RTL_Reg
    }

    consequent
    {
        ASM(li, o1, o2, o3) = astmt : ASM_Stmt
        o1 = rstmt.opd1 : ASM_Reg
        o2 = rstmt.opd2 : ASM_Reg
        o3 = rstmt.opd3 : ASM_Reg
    }
```

A few suggested changes in the syntax

```
{

    "move":{
        "rtl_move":{
            "antecedent":{
                "RTL(move, o1, o2)":"Move_RTL_Stmt",
                "o1":"RTL_Reg",
                "o2":"RTL_Reg"
            },
            "consequent":{
                "conse":{
                    "arrow":{
                        "RTL(move, o1, o2)":"Move_RTL_Stmt",
                        "ASM(move, o1, o2)":"ASM_Stmt"
                    }
                },
                "quent":[
                    {
                        "Reg_File[o2]":"Reg_File[o1]"
                    }
                ]
            }
        }
    }

}
```

Does JSON solve the syntax issue?

- Closer to paper specifications
- Each instruction is divided into antecedent and consequent as in the paper specifications
- Operand types explicitly mentioned
- Parsing JSON becomes easier as it is a well known format for carrying data
- Easily readable translation rules
- But, has overhead, for eg., the double quotes
- Translation and interpretation is not separated

Improvement in the previous syntax

```
{

    "move":{
        "rtl_move":{
            "antecedent":{
                "RTL(move, o1, o2)":"Move_RTL_Stmt",
                "o1":"RTL_Reg",
                "o2":"RTL_Reg"
            },
            "consequent":{
                    "arrow":{
                        "RTL(move, o1, o2)":"Move_RTL_Stmt",
                        "ASM(move, o1, o2)":"ASM_Stmt"
                    }
            }
        }
    }

}
```

Improvement in the previous syntax

- Differentiates between compiler state and program state
- Translation Rules, thus, corresponds to compilation
- IR Rules correspond to execution(or interpretation)

LEX script to parse Translation Rules

```
1
2  digit1to9 [1-9]
3  digit [0-9]
4  digits {digit}+
5  int {digit}|{digit1to9}{digits}|-{digit}|-{digit1to9}{digits}
6  frac [.]{digits}
7  exp {E}{digits}
8  E [eE][+-]?
9  hex_digit [0-9a-f]
10 number {int}|{int}{frac}|{int}{exp}|{int}{frac}{exp}
11 unescapedchar [ -!#-\[\]-~]
12 escapedchar \\["\\bnt/]
13 unicodechar \\u{hex_digit}{hex_digit}{hex_digit}{hex_digit}
14 char {unescapedchar}|{escapedchar}|{unicodechar}
15 chars {char}+
16 dbl_quote ["]
17
```

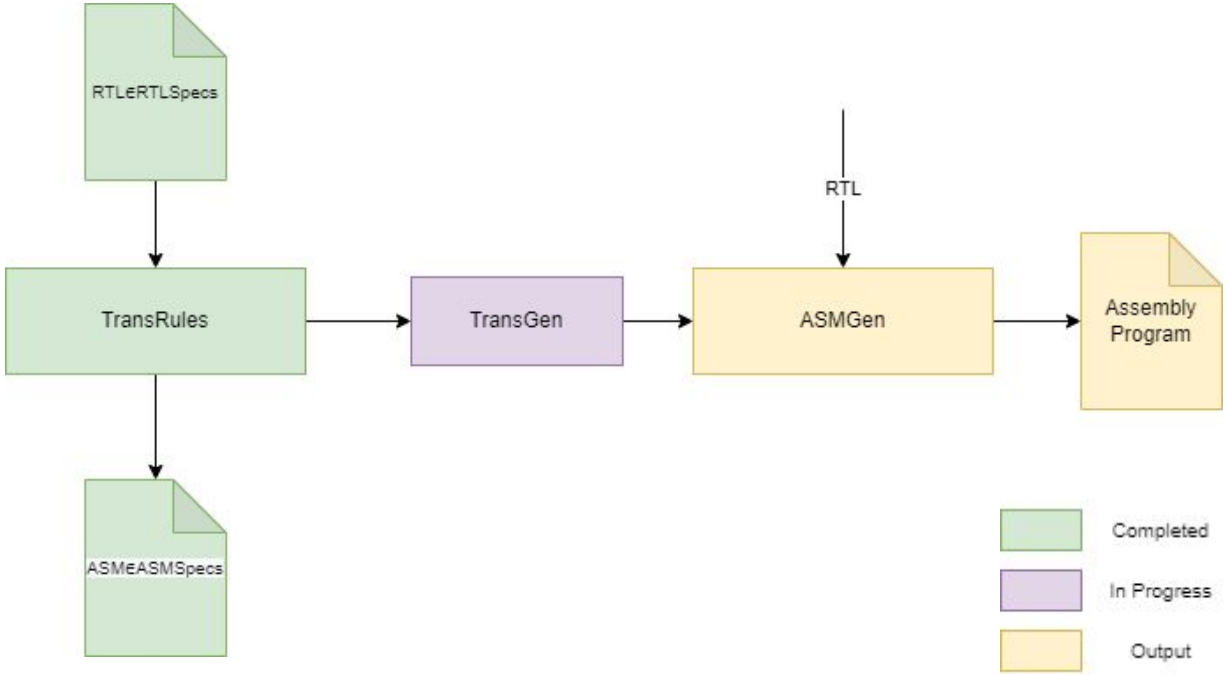
LEX script to parse Translation Rules

```
bhagyashree@bhagyashree-Lenovo-ideapad-320-15ISK:~/Documents/BTP/spec$ ./a.out
{
        "move" : {
                        "rtl_move" : {
                                        "antecedent" : {"RTL(move, o1, o2)":"Move_RTL_Stmt",
                                                                                                        "o1":"RTL_Reg",
                                                                                                        "o2":"RTL_Reg"},

                                        "consequent" : {

                                                                                                "conse" : { "arrow" : {
                                                                                                                                                {
                                        "RTL(move, o1, o2)":"Move_RTL_Stmt", "ASM(move, o1, o2)" : "ASM_Stmt"}},

                                                                                                "quent""move"
 : {"Reg_File[o2]" : "Reg_File[o1]"
:
                                                                                                                                                }
{
                                        }
                        }
        }
}"rtl_move"
:
{
"antecedent"
:
{
"RTL(move, o1, o2)"
:
"Move_RTL_Stmt"
,
"o1"
:
```

Division of work done

Sai Ghule	Studied and compiled a report on: - TAC - Command Line arguments for SCLP - SPIM Specifications for Load, Store and Move Instructions - RIPES Specifications for Arithmetic Instructions Studied RTL for Specs Rules for RTL to ASM Translation on paper(Paper Specifications)
Bhagyashree Rane	Studied and compiled a report on: - AST - Bug Reports - SPIM Specifications for Branch and Compare Instructions - RIPES Specifications for Branch and Compare Instructions - RTL Specifications for Load Instructions Lex Script for Parsing translation rules
Shravasti Deore	Studied and compiled a report on: - RTL - Bug Reports - SPIM Specifications for Arithmetic Instructions - RIPES Specifications for Load, Store and Move Instructions Compared RTL and ASM Translation rules

Progress


RTLeRTLSpecs
TransRules
TransGen
RTL
ASMGen
Assembly Program
ASMeASMSpecs
Completed
In Progress
Output

Technology

Languages to write the translator generators:

- C++
- Python

Lexical Analysers and Parser Generators:

- Lex and Yacc
- Flex & Bison

Simulators:

- SPIM MIPS simulator
- RIPES RISC-V simulator

Future Enhancements

Using Specifications with program state to:

- Implement generators that will construct interpreters
- Thus, providing virtual machines for AST, TAC, and RTL.

Documents and Process Followed

Documents Created so far

1. SRS (System Requirement Specification)
2. Bug Reports with 14 bugs
3. Specification Design Critique Report
4. Specification report for ASM of SPIM
5. Specification report for ASM of RIPES
6. Specification report for RTL
7. Rules for RTL to ASM Translation on paper(Paper Specifications)
8. Translation Rules Design Critique Report

Software Engineering

Agile Project:

1. Iterative Process:

- Thoroughly tested the SCLP compiler using existing test suite.
- SCLP has gone through a lot of versions each of which has been tested.

2. Flexible requirements :

- Project has grown.
- Literature survey has been from the very beginning.

Software Engineering

- The project is being developed using the Agile methodology.
- The requirements and solutions evolve through collaboration between the external mentor and the team every 10 days.
- The external mentor also checks if the requirements and the solutions are met after every iteration.
- We have weekly meetings with our external mentor Dr. Uday Khedkar sir.
- We also have 2 internal weekly meetings to discuss and distribute work and to make sure that each of us are on the same page.
- Practices such as using collaboration tools like Overleaf is used.
- We have also started with the NPTEL Course on compiler construction.

References

- A. V. Aho, Alfred V. Aho, Ravi Sethi, Jeffrey D. Ullman, J. D. Ullman, 1986. Compilers: Principles, Techniques, and Tools. Addison-Wesley.
- Tofte. M. 2012 compiler generators : what they can do , what they might do and what they will never do. Springer Science & Business Media.
- A Complete Specification of a Simple Compiler
- Sclp: A Language Processor for a Small C-like Language
- Programmed Introduction to MIPS Assembly Language
- SPIM Quick Reference
- MIPS IV Instruction Set
- SPIM Documentation
- Instruction Set Architecture
- The RISC-V Instruction Set Manual

Thank You