Design Laboratory (CS69202)

Spring Semester 2025

Topic : Lex + YACC

Date: January 8, 2025

Important Instructions:

1. Write Python code using PLY.
2. You must think correctly about what kind of errors can come in the process and try to handle them. Use the PLY package in python. PLY ref: https://www.dabeaz.com/ply/
3. You must NOT use any other parsing tools apart from PLY (ex: Beautiful Soup is a strict no or any other framework).
4. All errors should be handled properly.

Problem Statement

Develop a Python-based simulator using **PLY** (Python Lex-Yacc) to interpret and execute a custom assembly-like language. The simulator should support labeled instructions, basic arithmetic and logical operations, conditional jumps, output printing, and a simulated register-based memory architecture. The simulator will also manage program control flow. You are required to execute the set of lines and display the output in the terminal.

Language Specification

Instruction Set-

Memory Operations:

`STOR reg, value` -> Store the `value` in register `reg`.

`STOR A, 10 -> A = 10`

Arithmetic Operations:

- `SUM reg1,reg2 ->` Add the value of `reg2` to `reg1` and store the result in `reg1`.

- `SUB reg1,reg2` -> Subtract the value of `reg2` to `reg1` and store the result in `reg1`.

- `MUL reg1,reg2` -> Multiply the value of `reg2` to `reg1` and store the result in `reg1`.

- `DIV reg1,reg2` -> Divide the value of `reg1` by `reg2` (floating point division) and store the result in `reg1`.

- `MOD reg1,reg2` -> Calculate the remainder when the value of `reg1` is divided by the value of `reg2` and store the result in `reg1`.

Example:-

```
SUM A,B  -> A = A+B
SUB A,B  -> A = A-B
MUL A,B  -> A = A*B
DIV A,B  -> A = A/B
MOD A,B  -> A = A%B
```

Logical Operations:

- `AND reg1,reg2` -> Bitwise AND between the values of `reg2`, `reg1` and store the result in `reg1`.

- `OR reg1,reg2` -> Bitwise OR between the values of `reg2`, `reg1` and store the result in `reg1`.

- `XOR reg1,reg2` -> Bitwise XOR between the values of `reg2`, `reg1` and store the result in `reg1`.

- `NOT reg1` -> Bitwise NOT on the value of reg1 and store the result in reg1.

- `SHL reg1,rot_value` -> Perform left shift on the value of reg1 by the rot_value

- `SHR reg1,rot_value` -> Perform right shift on the value of reg1 by the rot_value

Example:-

```
AND A,B  -> A = A&B
OR A,B  -> A = A|B
XOR A,B  -> A = A^B
NOT A  -> A = ~A
SHL A,2  -> A = A<<2
SHR A,2  -> A>>2
```

Control Flow:

`IF <condition> <command> ->` If the specified condition evaluates to true, execute the command

Conditions:

- `reg == value`
- `reg != value`
- `reg > value`
- `reg < value`

`GOTO label ->` jump to the instruction at `label`

`HLT ->` Stop the program execution and print the memory content.

Example:

```
IF A==10 SUM A,2
GOTO L2
HLT
```

Output Operations:

`PRINT reg  ->` Print the value of register `reg` to the console.

Example:

```
PRINT A   -> print the content of register A
```

Supported Data Types:

- `Integer -> signed integer values, like -2,-1,0,1,2,...`
- `Float -> signed real numbers like -3.2,2.5,2.3564,....`
- `String -> texts within double quotes like “CR7”,”Hi am here”,....`

Additional String data type operations:-

- CONCAT a,b -> concat the content of register a to the content of register b and store the result in register a

- LENGTH a,b -> calculate the length of the content in register b and store it in register a

- SUBSTR a, pos1, pos2 -> Find the substring of the content in register a from pos1 to pos2 and store the result in register a. The substring of a string (0-indexed) is defined as the characters from pos1 to pos2-1. Substring(“Not Flower, wildfire”,4,10) -> “Flower”

Example:

```
CONCAT a,b   -> a = a+b
LENGTH a,b   -> a = length(b)
SUBSTR a,pos1,pos2   -> a = substring(a,pos1,pos2)
```

Instruction Format-

<label> $$$ <opcode> <operand list>

Example:-

.
.
.
.
L0 $$$ STOR A,”Study the examples”
L1 $$$ STOR a,10
L2 $$$ STOR b,2

```
L3 $$$ STOR @b,35          ->  M[b] = 35
L4 $$$ SUM a,5              ->  a = a+5
L5 $$$ PRINT a              -> a = 15 print in the terminal
L6 $$$ IF a==15 GOTO L1
L7 $$$ MUL a,15
L8 $$$ MOD a,b              -> a = a%b
L9 $$$ MOD a,@b             -> a = a%M[b]
.
.
.
.
.
.
```

Note:- Commands can be either Register Addressing Mode (L0,L1,L2,etc), Immediate addressing mode (L3) or Direct addressing mode (L9).