Military Institute of Science and Technology
Department of Computer Science and Technology
CSE-304 Compiler Sessional
Assignment-5: Intermediate Code and ASM Code Generation (8086 Instruction Set)

This question is regarded for the assessment of CO3.
CO3: Analyze and adapt the new tools and technologies used for designing a compiler.

Assignment:

In the previous assignment, we have performed syntax analysis of a source code written in a very small subset of **C** language. In this assignment you have to generate intermediate code for a source program having no error. That means if your source code does not contain any error you will have to generate intermediate representation in Three Address Code as well as generate assembly code considering the instruction set of 8086 microprocessor.

Language:

You should consider the following subset of C language which has the following characteristics:

- There can be only one function named **main** without any parameter and return type.
- Statements can be either declaration of a single variable of basic data type or arithmetic expression which ends with a semicolon.
- Arithmetic expression can be any assignment using all types of operator for arithmetic expression you have seen so far. Precedence and associativity rules should be maintained as per standard.

Note:

- There will be no pre-processing directives like #include or #define.
- No need to handle initialization of variable in the declaration statement.
- No return, break statement, switch-case statement, loop and conditional statement will be used.

Tasks:

You have to generate intermediate code from the input file (source program) considering that your source program does not contain any error as the source program already successfully passed the lexical and syntax analyzer.
You have to complete the following tasks:

1. Design Grammar for the mentioned language (partial grammar is given at last, you may change as required for the language).
2. Write LEX file for the tokens required for the designed grammar.
3. Write a YACC file in bison to incorporate the grammar.
4. Generate Intermediate Code (Three Address Code) in a file named "**code.ir"** using yacc file.
5. Generate ASM Code the corresponding three address code in a file named "code.asm" using the same bison file.
6. Print the Symbol Table in a file named **"table.txt"**
7. **Bonus tasks:** The following tasks will be considered bonus.
    a. Able to handle Print in the console
    b. Handling multiple function
    c. Handling the grammar used in Assignment-4 (Syntax Analyzer)

Instruction for Three Address Code Generation:
To generate three address code, consider the following instructions.

- You need to define a functions named **newTemp()** which will generate a new temporary variable name. For example t1, t2 etc.
- You have to change the type of YYSTYPE using either union so that it can handle SymbolInfo objects or redefining the YYSTYPE to SymbolInfo class.
- For each production of the grammar directly print three address code in the output file "**code.ir**" as soon as three address code needs be generated.

Instruction for ASM Code Generation:
Using the instruction set of 8086 Microprocessor, you have to design the ASM code.

- Add a field **'code'** attribute in **SymbolInfo** Class and corresponding function required for handling the attributes of the class.
- The **"code"** attribute should concatenate all the ASM code required for each three address code according to the designed grammar.
- When parser reduce to the start symbol, you should use this code attribute to print in the output file **"code.asm".**
- To check the generated "**code.asm**" file add initialization part in assembly program like initializing the data segment register in the main procedure of the generated assembly code.

Materials Attached with the Assignment:

1. **“input.txt”** is the file for which you have to generate **code.ir** and **code.asm** file.
2. Some sample input/output is given for simple expression

Partial Grammar:

```
prog → MAIN(){ stmt }

stmt → stmt unit | unit

unit → var_decl NEWLINE
      | expr_decl NEWLINE

var_decl → type_spec decl_list SEMICOLON

type_spec→ INT

decl_list→ term

expr→ NUM
      | expr ADDOP expr
      | expr MULOP expr
      | <add other grammar for other arithmetic and logical operator>
      | <add grammar rule for expression with parentheses>
      |term

term→ ID

expr_decl-> term ASSOP expr SEMICOLON
```