Bangladesh University of Engineering & Technology
Department of Computer Science & Engineering
CSE 310: Compiler Sessional
Session January 2021

# Assignment 4
# Intermediate Code Generation

---

June 12, 2021

## 1 Introduction

In the previous assignment, we have performed syntax and semantic analysis of a source code written in a 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, that was to be detected in the previous offline, you will have to generate intermediate code for the source code. We have picked 8086 assembly language as our intermediate representation.

## 2 Tasks

You have to complete the following tasks in this assignment.

### 2.1 Intermediate Code Generation

You have to generate 8086 assembly language program from the input file after the input file successfully pass all the previous steps (lexical, syntax, semantics). You have to do write codes in the same file of your previous assignment. To generate assembly code you may find the following instructions helpful.

- Add a field 'code' in SymbolInfo Class. As each of your non-terminal has a `SymbolInfo` pointer as attribute, you can propagate code for different portion using this newly added field.
- To generate assembly code for conditional statements and loops, define two functions named **newLabel()** and **newTemp()** where newLabel will generate a new label on each call and newTemp will generate a new temporary variable.

• Write a procedure for `println(ID)` function and call this procedure in your assembly code whenever you reduce a rule containing println.

• Do not forget some initialization part in assembly program like initializing the data segment register in the main procedure of the generated assembly code. So you may need to check whether the **ID** in the production body of func_definition is main or not.

• You have to declare the variables declared in the source code in the data segment of the assembly code. As the variable name can be same in different scope, you can concatenate the scope id with the variable name to make it unique.

• Handle the function call and return using stack. For an explanation see the class recording at around the time 00:54:00.

• Annotate the generated assembly code by putting the statement as a comment at the beginning of each block of assembly code that corresponds to the statement. See the class recording (at around the time 00:42:00) for an illustration.

• **Bonus tasks:** The following tasks will be considered bonus. (i) Handling recursive function and (ii) efficient use of temporary variables.

You may also find the given sample code helpful in this case. Note that the sample file consists of only some portion of the grammar. You have to generate intermediate code for all the productions given in previous grammar.txt file.

## 2.2 Optimization

You have to do some **peephole optimization** works after intermediate code is generated. See Section 8.7 of your text book for examples of **peephole optimizations.** One such example is removing redundant **mov** instruction from consecutive instructions as shown below.

**mov ax, a**
**mov a, ax ← redundant**

In this case, the second **mov** instruction can be omitted.

# 3 Input

The input will be a C source program in .c extension. File name will be given from command line.

# 4 Output

- The output of this assignment are primarily two files, namely **code.asm** and **optimized_code.asm**. The file **code.asm** will contain the generated assembly code before performing optimization. The other file **optimized_code.asm** will contain the assembly code after optimization. **The generated assembly files must run successfully on the emulator EMU8086.**

- Additionally, like the last assignment (assignment 3), there will be an **error.txt** file that will contain error count and any lexical, syntax or semantic errors.
- Note that, if there are errors in the input file then no codes will be generated, i.e., when **error.txt** is not-empty, the files **code.asm** and **optimized_code.asm** will be blank.
- You can also output a **log.txt** file as in assignment 3. But that is optional.

# 5 Submission

• Plagiarism is strongly prohibited. In case of plagiarism -100% marks will be given.

• No submission after the deadline will be allowed.

• Deadline shall not be extended under any circumstances.

# 6 Submission

All submissions will be taken via the departmental Moodle site. Please follow the steps given below to submit your assignment.

1. In your local machine create a new folder which name is your 7 digit student id.
2. Put the lex file named as <your_student_id>.l and yacc file named as <your_student_id>.y containing your code. Also put additional c file or header file that is necessary to compile your lex file. Do not put the generated lex.yy.c file or executable file in this folder. Additionally put the required script to compile and run your code. In case, your program cannot not produce assembly codes that run successfully on the emulator for all the supplied sample input files, provide your custom input files for which it works.
3. Compress the folder in a **zip** file which should be named as your 7 digit student id.
4. Submit the zip file within the Deadline.

# 7 Deadline

Submission deadline is set at **Monday July 05, 2021, 11:55 PM**. There will be no extension of this deadline. In case you cannot access the Moodle site, send an email to any of your course teachers attaching the zip file mentioned in the earlier section (within the deadline of course).