Bangladesh University of Engineering & Technology (BUET)
Department of Computer Science & Engineering (CSE)
CSE 310 (Compiler Sessional), January 2025 Term

Assignment 4
Intermediate Code Generation (ICG)

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, which 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 an 8086-assembly language program for an input program based on a given grammar. Your program must satisfy the following requirements.

- Use the same grammar that you used in assignment 3 on syntax and semantic analysis.
- Your program can generate the assembly code in one pass or two passes. For a two pass implementation, generate the parse tree in the first pass. In the second pass, traverse the parse tree (using appropriate traversal technique) and generate the desired assembly code.

- The generated assembly files must run successfully on the emulator EMU8086.
- You must generate the code **on the fly** and write to some file. You are allowed to use at most one temporary file to store and process the generated code.
- All the local variables of a function must be stored and accessed through the stack. You are not allowed to declare any variable in the data segment to represent a local variable.
- Use short-circuit or jumping code while generating code for boolean expressions.
- All the parameters to a function must be passed through the stack. However, for the return value you are allowed to use a register.
- Annotation of the generated assembly code by putting the line no from the source file as comments to show the mapping of a block of assembly code to its corresponding C code.
- Write a procedure for `println(ID)` function and call this procedure in your assembly code whenever you reduce a rule containing **println**.
- You are not required to handle any floating point operations in the source file.
- **Bonus tasks:** Handling recursive function..

2.2 Optimization

You have to do some **peephole optimizations** after intermediate code is generated. See Section 8.7 of your textbook for examples of **peephole optimizations.** Some examples are given below.

(i) Removing redundant **MOV** instruction from consecutive instructions as shown below.

> **MOV AX, a**
>
> **MOV a, AX ← redundant; can be removed**

(ii) Removing redundant consecutive **push and pop** of the same register/address.

      PUSH AX

      POP AX

(iii) Removing redundant operations such as -

      ADD AX, 0

      MUL AX, 1

(iv) Removing redundant labels such -

      L3:

      L4:

      L5: ← Any two of the three labels can be removed

3   Input

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

5   Some Guidelines

- Instead of writing the full code in one shot, build your code incrementally for different constructs of the language. You may follow the below order.
  - Data: constants → non-array variables → array variables
  - Expressions: add operations → multiplication operations → unary operations → relational operations → logical operations
  - Statements: assignments → if → if-else → loops

○ Functions: no arg no return value → no args but with return value → having both args and return value

5 Submission

- Plagiarism is strongly prohibited. In case of plagiarism -100% marks will be given.
- No submission after the deadline will be allowed.

6 Submission

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

i.   In your local machine, create a new folder and name it with your 7-digit student id.

ii. Put all the required files including a 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.

iii. Compress the folder in a **zip** file which should be named as your 7-digit student id.

iv. Submit the zip file within the Deadline.

7 Deadline

Submission deadline: **Friday July 18, 2025, 11:00 PM**.