

Assignment #3

Assembly Code Generation (13 marks)

Deadline : 11/10/2020, 11:55PM

1 Task

The aim of the assignment is to generate x86 assembly code for the language used in Assignment 2 with the help of generated Abstract Syntax Tree from Assignment #2.

2 For which basic constructs assembly code should be generated?

We will be using the grammar given in the Assignment #2 and we will restrict assembly generation only to few constructs. Our test-cases (.c file) will contain only the below basic constructs for which you are required to generate assembly code,

- Variable declaration statements - You need to handle only integer type declarations (excluding pointers in that).
- Assignment statements - Includes all but excluding Array and Function assignment statements.
- Function calls - You need to handle only 'printf' function call. User-defined function and others are excluded.
- If statement - Includes both 'If then' and 'If then else'. The nesting of If statement is excluded.
- While statement - Excluding nesting of 'While' statement, 'break' and 'continue' constructs. Also, no combination of If and While statements together (i.e.) no If stmt inside While and vice-versa.

Bonus (2 marks): You should handle below construct to get the bonus marks,

- Functions - You should handle user-defined functions and return statements (1 mark).
- Handling Array and Function assignment statements (1 mark).

3 Input

The input will be a program (.c file) which contains statements according to the above basic construct(s) description.

Sample execution format:

```
$ ./a.out < test_case1.c
```

Note: Input should be read from stdin.

4 Output

You should emit the assembly code for the corresponding input file (here test_case1.c), which we will capture to a .s file like,

```
$ ./a.out < test_case1.c > tc1.s
```

Note: Output should be written to stdout.

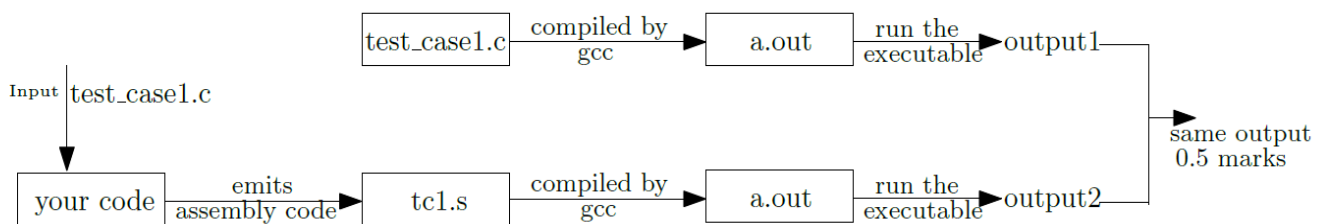
5 Submission (in moodle)

Submit a tar.gz file with filename as $\langle \text{ROLLNO} \rangle$.tar.gz (eg.CS18B043.tar.gz) containing the following structure:

- CS18B043 $\langle \text{directory} \rangle$
 - *.l
 - *.y
 - The Makefile should run lex, yacc, compile the generated code and generate an executable a.out file.

6 How we will evaluate your code?

- We will evaluate your code against 30 test-cases. 5 public test-cases, 21 hidden test-cases, and last 4 test-cases for Bonus marks.
- Each test-case carries .5 marks and total marks is 13. Bonus is 2 mark (last 4 test-cases).
- Below figure shows how your code is evaluated,



- The 30 test-cases will contain a mix-up of very simple, simple, less complex and complex test-cases. We will look only at the final printed output not your generated assembly code. So, you must generate the assembly code for ‘printf’ function call to pass all test-cases.

7 Sample Test Case

- testcase_1.c

– Input

```
int main() {
    int a;
    a = 6;
    printf("%d\n", a);
    return 0;
}
```

– Your emitted assembly code stored in tc1.s file

```
.LC0:
.string "%d\n"
.text
.globl main
.type main, @function
```

```

main:
.LFB0:
    .cfi_startproc
    .pushq   %rbp
    .cfi_def_cfa_offset 16
    .cfi_offset 6, -16
    movq    %rsp, %rbp
    .cfi_def_cfa_register 6
    subq    $16, %rsp
    movl    $6, -4(%rbp)
    movl    -4(%rbp), %eax
    movl    %eax, %esi
    movl    $.LC0, %edi
    movl    $0, %eax
    call    printf
    movl    $0, %eax
    leave
    .cfi_def_cfa 7,8
    ret
    .cfi_endproc

```

– **evaluation**

```

$ gcc testcase_1.c
$ ./a.out
6
$ gcc tc1.s
$ ./a.out
6

```

– here both output result matches and will be given 0.5 marks.

8 What will we upload in the moodle?

- This file (Assignment3.pdf), 5 + 2 public test-cases (last two are bonus test-cases) and evaluation script.
- All 30 test-cases (after the deadline).

9 Some stuff(s) to know

- You can generate assembly code using gcc. how? `$gcc -S testcase_1.c`
- A tool named MOSS, which can compare multiple codes and identifies which parts of the code are similar (copied).

10 FAQs

- You are allowed to use both C and C++ languages for coding.
- Only accepted printf stmt used in test cases as per the grammar is “printf(“%d\n”, identifier);”
- The marks uploaded in moodle will be out of 13.

- Emit your code only using `<STDIO>`, not using `<STDERR>`.
- Remove all your debugging statements.

11 Grammer of Assignment #2

```
program → decl_list
decl_list → decl_list decl | decl
decl → var_decl | func_decl | struct_decl
struct_decl → "struct" identifier "{" local_decls "}" ";"
var_decl → type_spec identifier ";"
          | type_spec identifier "," var_decl
          | type_spec identifier "[" integerLit "]" ";"
          | type_spec identifier "[" integerLit "]" "," var_decl
type_spec → extern_spec "void" | extern_spec "int" | extern_spec "float"
          | extern_spec "void" "*" | extern_spec "int" "*" | extern_spec "float" "*"
          | "struct" identifier | "struct" identifier "*"
extern_spec → "extern" | ε
fun_decl → type_spec identifier "(" params ")" compound_stmt
params → param_list | ε
param_list → param_list "," param | param
param → type_spec identifier | type_spec identifier "[" "]"
stmt_list → stmt_list stmt | stmt
stmt → assign_stmt | compound_stmt | if_stmt | while_stmt | switch_stmt
      | return_stmt | break_stmt | continue_stmt | dowhile_stmt | print_stmt
      | incr_stmt | decr_stmt
while_stmt → "while" "(" expr ")" stmt
dowhile_stmt → "do" stmt "while" "(" "expr" ")" ";"
print_stmt → "printf" "(" format_specifier "," identifier ")" ";"
format_specifier → "%d"
compound_stmt → "{" local_decls stmt_list "}"
local_decls → local_decls local_decl | ε
local_decl → type_spec identifier ";"
            | type_spec identifier "[" expr "]" ";"
if_stmt → "if" "(" expr ")" stmt
         | "if" "(" expr ")" stmt "else" stmt
return_stmt → "return" ";" | "return" expr ";"
break_stmt → "break" ";"
continue_stmt → "continue" ";"
switch_stmt → "switch" "(" expr ")" "{" compound_case default_case "}"
compound_case → single_case compound_case
              | single_case
single_case → "case" integerLit ":" stmt_list
default_case → "default" ":" stmt_list
```

$\text{assign_stmt} \rightarrow \text{identifier "=" expr ";;"} \mid \text{identifier "[" expr "]" "=" expr ";;"}$
 $\mid \text{identifier "->" identifier "=" expr ";;"}$
 $\mid \text{identifier "." identifier "=" expr ";;"}$
 $\text{incr_stmt} \rightarrow \text{identifier "++" ";;"}$
 $\text{decr_stmt} \rightarrow \text{identifier "--" ";;"}$
 $\text{expr} \rightarrow \text{Pexpr "<" Pexpr} \mid \text{Pexpr ">" Pexpr}$
 $\rightarrow \text{Pexpr "<=" Pexpr} \mid \text{Pexpr ">=" Pexpr}$
 $\rightarrow \text{Pexpr "||" Pexpr} \mid \text{"sizeof" "(" Pexpr ")"}$
 $\rightarrow \text{Pexpr "==" Pexpr} \mid \text{Pexpr "!=" Pexpr} \mid \text{Pexpr "<=>" Pexpr}$
 $\rightarrow \text{Pexpr "&\&" Pexpr} \mid \text{Pexpr "->" Pexpr}$
 $\rightarrow \text{Pexpr "+" Pexpr} \mid \text{Pexpr "-" Pexpr}$
 $\rightarrow \text{Pexpr "*" Pexpr} \mid \text{Pexpr "/" Pexpr} \mid \text{Pexpr "%" Pexpr}$
 $\rightarrow \text{"!" Pexpr} \mid \text{"-" Pexpr} \mid \text{"+" Pexpr} \mid \text{"*" Pexp} \mid \text{"\&" Pexp}$
 $\rightarrow \text{Pexpr}$
 $\rightarrow \text{identifier "(" args ")"}$
 $\rightarrow \text{identifier "[" expr "]"}$
 $\text{Pexpr} \rightarrow \text{integerLit} \mid \text{floatLit} \mid \text{identifier} \mid \text{"(" expr ")"}$
 $\text{integerLit} \rightarrow < \text{INTEGER_LITERAL} >$
 $\text{floatLit} \rightarrow < \text{FLOAT_LITERAL} >$
 $\text{identifier} \rightarrow < \text{IDENTIFIER} >$
 $\text{arg_list} \rightarrow \text{arg_list "," expr} \mid \text{expr}$
 $\text{args} \rightarrow \text{arg_list} \mid \epsilon$