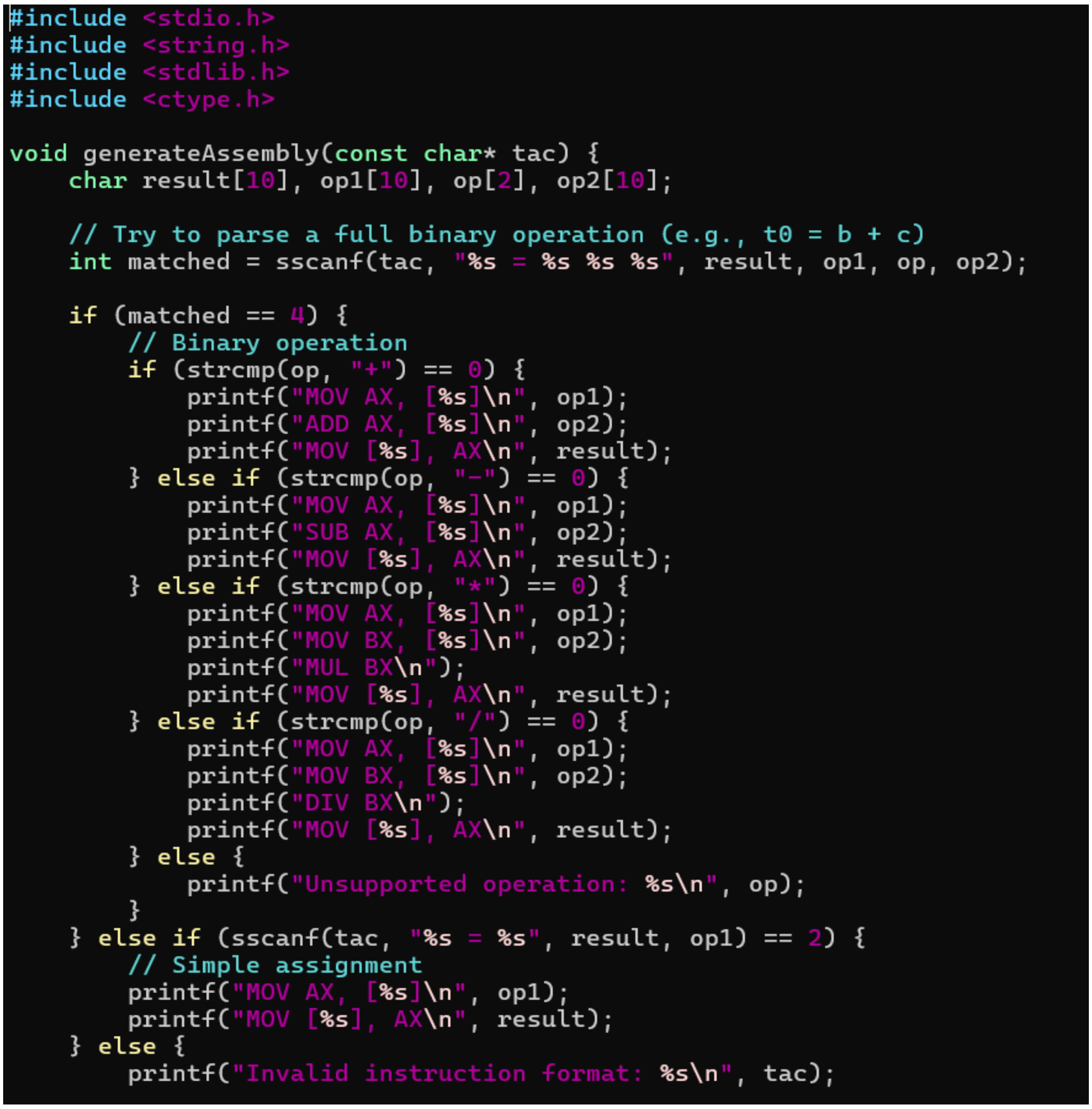
EXP NO-9

DEVELOP THE BACK-END OF A COMPILER THAT TAKES THREE-ADDRESS CODE (TAC) AS INPUT AND GENERATES CORRESPONDING 8086 ASSEMBLY LANGUAGE CODE AS OUTPUT.

AIM:

To design and implement the back-end of a compiler that takes three-address code (TAC) as input and produces 8086 assembly language code as output. The three-address code is an intermediate representation used by compilers to break down expressions and operations, while the 8086 Assembly code is a machine-level representation of the program that can be executed by a processor.

PROGRAM : TAC.C



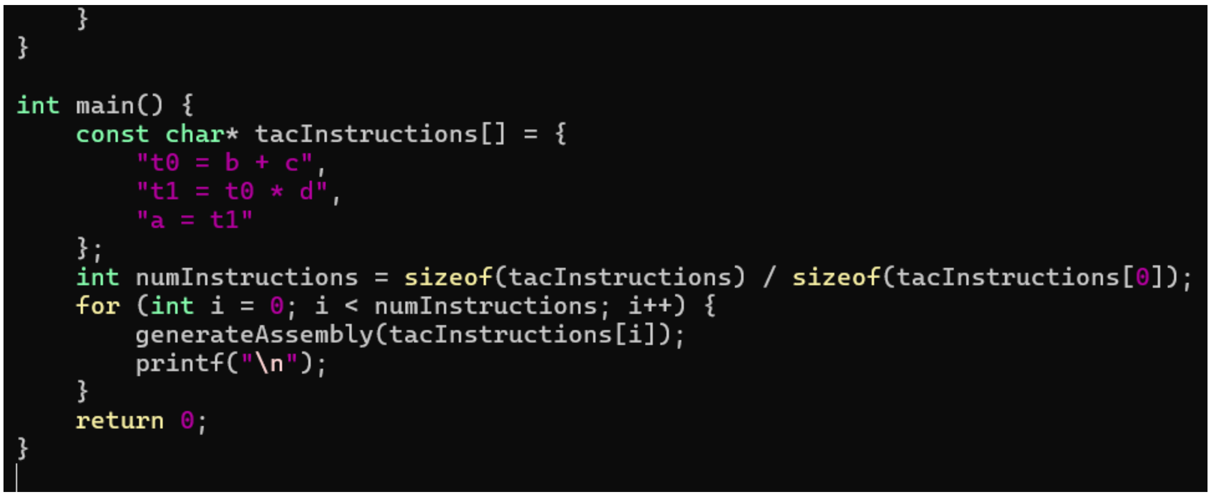
```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <ctype.h>

void generateAssembly(const char* tac) {
    char result[10], op1[10], op[2], op2[10];

    // Try to parse a full binary operation (e.g., t0 = b + c)
    int matched = sscanf(tac, "%s = %s %s %s", result, op1, op, op2);

    if (matched == 4) {
        // Binary operation
        if (strcmp(op, "+") == 0) {
            printf("MOV AX, [%s]\n", op1);
            printf("ADD AX, [%s]\n", op2);
            printf("MOV [%s], AX\n", result);
        } else if (strcmp(op, "-") == 0) {
            printf("MOV AX, [%s]\n", op1);
            printf("SUB AX, [%s]\n", op2);
            printf("MOV [%s], AX\n", result);
        } else if (strcmp(op, "*") == 0) {
            printf("MOV AX, [%s]\n", op1);
            printf("MOV BX, [%s]\n", op2);
            printf("MUL BX\n");
            printf("MOV [%s], AX\n", result);
        } else if (strcmp(op, "/") == 0) {
            printf("MOV AX, [%s]\n", op1);
            printf("MOV BX, [%s]\n", op2);
            printf("DIV BX\n");
            printf("MOV [%s], AX\n", result);
        } else {
            printf("Unsupported operation: %s\n", op);
        }
    } else if (sscanf(tac, "%s = %s", result, op1) == 2) {
        // Simple assignment
        printf("MOV AX, [%s]\n", op1);
        printf("MOV [%s], AX\n", result);
    } else {
        printf("Invalid instruction format: %s\n", tac);
```

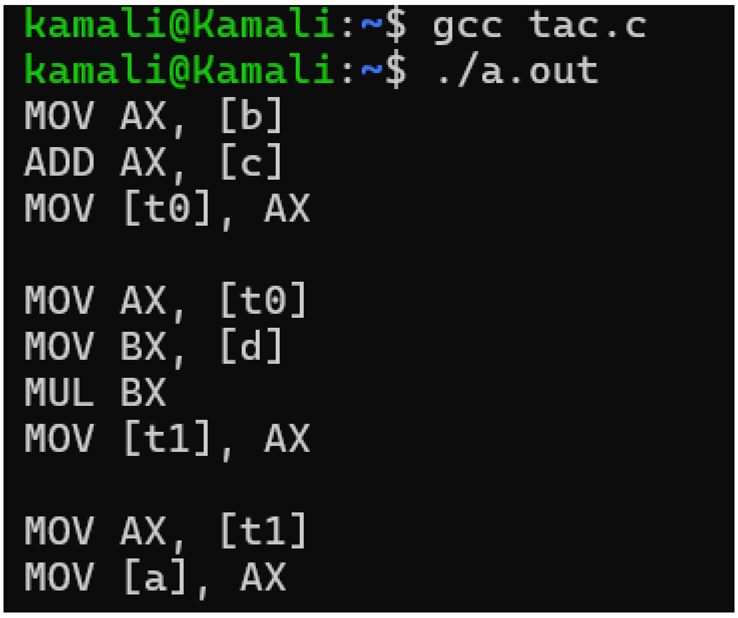
KAMALI K A - 220701118



```
    }
}

int main() {
    const char* tacInstructions[] = {
        "t0 = b + c",
        "t1 = t0 * d",
        "a = t1"
    };
    int numInstructions = sizeof(tacInstructions) / sizeof(tacInstructions[0]);
    for (int i = 0; i < numInstructions; i++) {
        generateAssembly(tacInstructions[i]);
        printf("\n");
    }
    return 0;
}
```

OUTPUT



```
kamali@Kamali:~$ gcc tac.c
kamali@Kamali:~$ ./a.out
MOV AX, [b]
ADD AX, [c]
MOV [t0], AX

MOV AX, [t0]
MOV BX, [d]
MUL BX
MOV [t1], AX

MOV AX, [t1]
MOV [a], AX
```

RESULT:

Thus the above example provides a foundational approach to converting TAC to 8086 assembly using C. For a complete compiler back-end, you would need to handle additional aspects such as register allocation, memory management, and more complex control flow constructs.

KAMALI K A - 220701118