PART B
EXPERIMENT NUMBER 6

Aim: Write a program to generate the target code.

(PART B: TO BE COMPLETED BY STUDENTS)

(Students must submit the soft copy as per the following segments within two hours of the practical. The soft copy must be uploaded at the end of the practical)

Roll No. 50	**Name:** AMEY THAKUR
Class: Comps TE B	**Batch:** B3
Date of Experiment: 09/04/2021	**Date of Submission:** 09/04/2021
Grade:	

B.1 Software Code written by a student:

(Paste your code completed during the 2 hours of practice in the lab here)

- **SPCC-6.C**

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
int main()
{
char A[12][12]={"T=A B","X=T"};
char B;
int OP=0;
printf("MOV %c,R1\n",A[0][2]);
printf("MOV %c,R2\n",A[0][4]);

B=A[0][3];
printf("THERE ARE TWO EXPRESSIONS: \n1.T=A OP B\n2.X=T\nCHOOSE THE OP VALUE FROM OPTION\n1.+\n2.-\n3.*\n4./\n");
scanf("%d",&OP);
switch(OP)
{
case 1:
printf("ADD R1,R2\n");
break;
case 2:
printf("SUB R1,R2\n");
```

```
break;
case 3:
printf("MUL R1,R2\n");
break;
case 4:
printf("DIV R1,R2\n");
break;
default:
break;
}
printf("MOV R1,x");
getch();
}
```

B.2 Input and Output:

```
C:\Users\ameyt\Desktop>GCC SPCC-6.C

C:\Users\ameyt\Desktop>a.exe
MOV A,R1
MOV B,R2
THERE ARE TWO EXPRESSIONS:
1.T=A OP B
2.X=T
CHOOSE THE OP VALUE FROM OPTION
1.+
2.-
3.*
4./
1
ADD R1,R2
MOV R1,x
```

```
C:\Users\ameyt\Desktop>GCC SPCC-6.C

C:\Users\ameyt\Desktop>a.exe
MOV A,R1
MOV B,R2
THERE ARE TWO EXPRESSIONS:
1.T=A OP B
2.X=T
CHOOSE THE OP VALUE FROM OPTION
1.+
2.-
3.*
4./
2
SUB R1,R2
MOV R1,x
```

```
C:\Users\ameyt\Desktop>GCC SPCC-6.C

C:\Users\ameyt\Desktop>a.exe
MOV A,R1
MOV B,R2
THERE ARE TWO EXPRESSIONS:
1.T=A OP B
2.X=T
CHOOSE THE OP VALUE FROM OPTION
1.+
2.-
3.*
4./
3
MUL R1,R2
MOV R1,x
```

```
C:\Users\ameyt\Desktop>GCC SPCC-6.C

C:\Users\ameyt\Desktop>a.exe
MOV A,R1
MOV B,R2
THERE ARE TWO EXPRESSIONS:
1.T=A OP B
2.X=T
CHOOSE THE OP VALUE FROM OPTION
1.+
2.-
3.*
4./
4
DIV R1,R2
MOV R1,x
```

B.3 Observations and learning:

(Students are expected to comment on the output obtained with clear observations and learning for each task/ subpart assigned)

Target code could be easily generated by making use of the switch case.

B.4 Conclusion:

(Students must write the conclusion as per the attainment of individual outcome listed above and learning/observation noted in section B.3)

We implemented a program to generate target code understood intermediate code generation and code generation phase of the compiler

B.5 Question of Curiosity

(To be answered by a student based on the practical performed and learning/ observations)

1. **Differentiate Between Machine Dependent and Machine Independent code optimization.**

Ans:

The main difference between both is that the machine-dependent optimization is applied to the object code and the independent code optimization is applied to the intermediate code.

➔ Machine code depends on specific features of the computer to which it was written. Independent computer or portable code includes less and, preferably, no such dependency.

➔ Machine-based optimization requires CPU registers and absolute memory references, while CPU registers or absolute memory references are not used in machine-independent code optimization.

2. Describe Peephole Optimization.

Ans:

➔ Peephole optimization is a type of Code Optimization performed on a small part of the code. It is performed on a very small set of instructions in a segment of code.

➔ *The small set of instructions or a small part of the code on which peephole optimization is performed is known as* ***peephole*** *or* ***window****.*

➔ It works on the theory of replacement in which a part of code is replaced by shorter and faster code without change in output. The peephole is machine-dependent optimization.

➔ The objective of peephole optimization is:

1. To improve performance
2. To reduce memory footprint
3. To reduce code size

Peephole Optimization Techniques:

1. Redundant load and store elimination:

In this technique, redundancy is eliminated.

Initial code:

```
y = x + 5;
i = y;
z = i;
w = z * 3;
```

Optimized code:

```
y = x + 5;
i = y;
w = y * 3;
```

2. Constant folding:

The code that can be simplified by the user itself, is simplified.

Initial code:

```
x = 2 * 3;
```

Optimized code:

```
x = 6;
```

3. Strength Reduction:

The operators that consume higher execution time are replaced by the operators consuming less execution time.

Initial code:

```
y = x * 2;
```

Optimized code:

```
y = x + x;    or    y = x << 1;
```

Initial code:

```
y = x / 2;
```

Optimized code:

```
y = x >> 1;
```

4. Null sequences:

Useless operations are deleted.

5. Combine operations:

Several operations are replaced by a single equivalent operation.