**Exercise – 10:** Generate the machine code from the given intermediate code.

**Aim:** To generate the machine for the Intel 8086 architecture for a simple case.

**Theory:**

Machine codes are the ones that are finally loaded into the main memory of the computer for execution. Hence the machine code is dependent on the target machine. In order to generate the machine code, the knowledge of the hardware architecture of the target machine, addressing modes, instruction, etc. is essential. It is too involed with the bit and byte of the target machine. Code generation process involves the allocation and reallocation of the registers in the processor of the target machine and mapping of the intermedite code with the machine code.

**Steps:**

1. Initialize the register descriptor as an array of characters.
2. Read the quadruple in the format `op, operand1, operand2, result.`
3. For every quadruple entry perform the following:
    a. if an operand is already available required register then useLoad the operand in one register
    b. else
    c. Obtain the free register and load the operand into the register
    d. Load the second operand into another register
    e. Call the appropriate code generation routine depending on the operator

**Algorithm:**

An outline of the code generation procedure is given section 8.5.2. The following algorithm presents the code generation process for a simple move and arithmetic operations.

```
Algorithm – genCode(Q)
// List of Quadruples
//returns the 8086 assembly language code equivalent
{
      for each quadruple Q in quadruples list do
      {
            R1= register for  operand1
            genCode(“Mov”, R1, operand1)
            R2 = register for operand2
            genCode(“Mov”, R2, operand2)
            genCode(operator,R1,R2)
            genCode(“Mov”result,R1)
      }
}
```

**Source Code Listing for Machine Code generation for a simple arithmetic expression:**

```c
#include<stdio.h>
#include<string.h>
#define noop 6

typedef struct quad
{
  char operand1[10];
  char operand2[10];
  char oprator[10];
  char result[10];
}quad;

char *oparray[]={"+","-","*","/","uminus","="};
quad readQuadruples(FILE*);
void genCode(quad);

void add(quad);
void sub(quad);
void mul(quad);
void div(quad);
void neg(quad);
void ass(quad);

void add(quad q1)
{
      printf("MOV AX, %s\n",q1.operand1);
      printf("MOV BX, %s\n",q1.operand2);
      printf("ADD AX, BX\n");
      printf("MOV %s, AX\n",q1.result);
}
void sub(quad q1)
{
      printf("MOV AX, %s\n",q1.operand1);
      printf("MOV BX, %s\n",q1.operand2);
      printf("SUB AX, BX\n");
      printf("MOV %s, AX\n",q1.result);
}

void mul(quad q1)
```

```
{
      printf("MOV AX, %s\n",q1.operand1);
      printf("MOV BX, %s\n",q1.operand2);
      printf("MUL BX\n");
      printf("MOV %s, AX\n",q1.result);
}

void div(quad q1)
{
      printf("MOV AX, %s\n",q1.operand1);
      printf("MOV BX, %s\n",q1.operand2);
      printf("DIV BX\n");
      printf("MOV %s, AX\n",q1.result);
}

void neg(quad q1)
{
      printf("MOV AX, %s\n",q1.operand1);
      printf("NEG AX\n");
      printf("MOV %s, AX\n",q1.result);
}
void assi(quad q1)
{
      printf("MOV AX, %s\n",q1.operand1);
      printf("MOV %s, AX\n",q1.result);
}

void (*opname[])(quad) ={add,sub,mul,div,neg,assi};
```

```
quad readQuadruples(FILE*fp)
{
      quad q1;
      fscanf(fp,"%s\t%s\t%s\t%s\n",q1.oprator,q1.operand1,q1.operand2,q1.re
      sult);
      printf("The     given     quadruple     is:     %s     =     %s     %s
      %s\n",q1.result,q1.operand1,q1.oprator,q1.operand2);
      return q1;
}

void genCode(quad q1)
{
      void(*ptof)(quad);
      int i;

      // Find the index of the operator in the array of operator names
      for(i=0;i<noop;i++)
        if(strcmp(oparray[i],q1.oprator)==0)
                break;

      if(i==noop)
      {
        printf("Invalid Operator");
        return;
      }
      // Call the semantic routine that has the name of the operator using
      //pointer to functions
      ptof=opname[i];
      (*ptof)(q1);
}

int main()
{
      FILE *fp;
      quad q1;

      fp=fopen("ari.txt","r");

      do
      {
            q1=readQuadruples(fp);
            genCode(q1);
      } while(!feof(fp));
      fclose(fp);
}
```

## Output

```
The given quadruple is: c = a + b
MOV AX, a
MOV BX, b
ADD AX, BX
MOV c, AX
The given quadruple is: r = p - q
MOV AX, p
MOV BX, q
SUB AX, BX
MOV r, AX
The given quadruple is: z = x * y
```

```
MOV AX, x
MOV BX, y
MUL BX
MOV z, AX
```

We have generated the assembly code for the processor. However, the machine code has to be generated for the given program. Generation of machine for 8086 processor is dealt in section 8.5.2.

**Exercise for the students:**

Experiment with other expression and other programming language constructs by suitably modifying the program.