```
//A sample 16-bit instruction set encoding program in C++
#include <cstdlib>
#include <iostream>
#include <string>
#include <iomanip>
#include <sstream>
using namespace std;
 int mem = 0, loaddata[0];
 bool stallflag = 0;
 bool newflag = 0;
 bool bloody = 0;
string sixteenbits(int r[])
{
     int a[16], b = 0, rev = 0, bin1[16], len, lennew;
     string s, val, k1 = "0000000000000000";
      int newreg = r[0];
      while(newreg)
           {
                a[b] = newreg % 2;
                newreg /= 2;
                ++b;
                }

           for(int n = b-1; n >=0; n--)
              {
             rev = rev * 10 + a[n];
             }
```

```
   stringstream out; //convert int to binary
   out << rev; // convert int to binary
   s = out.str(); // convert int to binary
   len = s.length();
   lennew = 16 - len;
   for(int j1 = 0; j1 < lennew; j1++)
    val += k1.at(j1);

    val += s; // final 16- character string form of binary is
available in val
    return val;
}
string twoscomplement(int r[]) // 2's complement 16-bit
{
     int a1[8] = {1,0,0,0,0}, c[8] = {0}, mon, carry = 0, p =
0;
     int a[8] = {0}, b = 0, rev = 0, bin1[16], len, lennew;
     string s, s1, val, k1 = "1111111111111111";
      int newreg = r[0];
      while(newreg)
           {
                 mon = newreg % 2;

                 if(mon == 0)
                 a[b] = 1;
                else if(mon == 1)
                 a[b] = 0;
```

```
                ++b;
                newreg /= 2;

                } //end of while

    for(int i = 0; i < b; i++)
    {

             if (a1[i] + a[i] + carry == 1) // 1 + 0 = 1
            { c[p]= 1;
             carry = 0;}
             else if(a1[i] + a[i] + carry == 0)
             {
               c[p]= 0;
               carry = 0;
             }

             else if(a1[i] + a[i] + carry == 2) // 1 + 1 =
0, carry 1
             {c[p]= 0;
             carry = 1;
             }
             else if (a1[i] + a[i] + carry == 3)//1+1+1 = 1,
carry 1
             {
             c[p]= 1;
             carry = 1;
```

```
                }
                 ++p;
        }

       stringstream out; //convert int to binary
       for(int i = 0; i < p; i++)
       {

         out << c[i]; //out << rev; // convert int to binary

        }
        s = out.str(); // convert int to binary

        for(int n = s.length()-1; n >=0; n--)
               {
             val += s.at(n);
        } // end of for

   len = val.length();
   lennew = 16 - len;
   for(int j1 = 0; j1 < lennew; j1++)
    s1 += k1.at(j1);

    s1 += val; // final 16- character string form of binary is
available in val

    return s1;
}
```

```
string thirtyTwobits(int r[])
{
     int a[32], b = 0, rev = 0, bin1[32], len, lennew;
     string s, val, k1 = "00000000000000000000000000000000";
      int newreg = r[0];
      while(newreg)
           {
                a[b] = newreg % 2;
                newreg /= 2;
                ++b;
                }  //end of while

           for(int n = b-1; n >=0; n--)
              {
             rev = rev * 10 + a[n]; // binary form of
number[y];
             } // end of for

   stringstream out; //convert int to binary
   out << rev; // convert int to binary
   s = out.str(); // convert int to binary
   len = s.length();
   lennew = 32 - len;
   for(int j1 = 0; j1 < lennew; j1++)
    val += k1.at(j1);

    val += s; // final 32- character string form of binary is
available in val
```

```
    return val;
}
string twoscomplement32bits(int r[])
{
     int a1[16] = {1,0,0,0,0}, c[16] = {0}, mon, carry = 0, p
= 0;
     int a[16] = {0}, b = 0, rev = 0, bin1[16], len, lennew;
     string s, s1, val, k1 =
"11111111111111111111111111111111";
      int newreg = r[0];
      while(newreg)
           {
                 mon = newreg % 2;

                 if(mon == 0)
                 a[b] = 1;
                else if(mon == 1)
                 a[b] = 0;

                 ++b;
                 newreg /= 2;

                 }  //end of while

      for(int i = 0; i < b; i++)
      {
           if (a1[i] + a[i] + carry == 1) // 1 + 0 = 1
              { c[p]= 1;
```

```
                carry = 0;}
                else if(a1[i] + a[i] + carry == 0)
                {
                  c[p]= 0;
                  carry = 0;
                }

                else if(a1[i] + a[i] + carry == 2) // 1 + 1 =
0, carry 1
                {c[p]= 0;
                carry = 1;
                }
                else if (a1[i] + a[i] + carry == 3)//1+1+1 = 1,
carry 1
                {
                c[p]= 1;
                carry = 1;
                }
                 ++p;
       }

      stringstream out; //convert int to binary
      for(int i = 0; i < p; i++)
      {

        out << c[i]; //out << rev; // convert int to binary

       }
```

```
        s = out.str(); // convert int to binary


        for(int n = s.length()-1; n >=0; n--)
               {
             val += s.at(n);
        } // end of for


   len = val.length();
   lennew = 32 - len;
   for(int j1 = 0; j1 < lennew; j1++)
    s1 += k1.at(j1);


    s1 += val; // final 32- character string form of binary is
available in val


    return s1;
}
void load(int r0[], int loaddata[], string operand1[], int
r1[], int mem, int j)
{
      if(r1[0] == mem)
      {
      r0[0] = loaddata[0];
      if(r0[0] >= 0)
      cout << " " << operand1[j] << " = " << r0[0] << " [" <<
sixteenbits(r0) << "] " << endl;
      else
      cout << " " << operand1[j] << " = " << r0[0] << " [" <<
twoscomplement(r0) << "] " << endl;
```

```
      }
      else
       {cout << " Error! Memory is not defined in register
r0!!\n";
       bloody = 1;
       }
}
void addload(int r0[], int loaddata[], string operand1[], int
r1[], int mem, int j)
{
      if(r1[0] == mem)
      {
      r0[0] += loaddata[0];
      if(r0[0] >= 0)
      cout << " " << operand1[j] << " = " << r0[0] << " [" <<
sixteenbits(r0) << "] " << endl;
      else
      cout << " " << operand1[j] << " = " << r0[0] << " [" <<
twoscomplement(r0) << "] " << endl;
      }
      else
       {
            cout << " Error! Memory is not defined in register
r0!!\n";
            bloody = 1;
            }
}
void subload(int r0[], int loaddata[], string operand1[], int
r1[], int mem, int j)
{
```

```
      if(r1[0] == mem)
      {
      r0[0] -= loaddata[0];
      if(r0[0] >= 0)
      cout << " " << operand1[j] << " = " << r0[0] << " [" <<
sixteenbits(r0) << "] " << endl;
      else
      cout << " " << operand1[j] << " = " << r0[0] << " [" <<
twoscomplement(r0) << "] " << endl;
      }
      else
       {
            cout << " Error! Memory is not defined in register
r0!!\n";
            bloody = 1;
            }
}
void movr0(int r0[], int r1[], string operand1[], int j)
{

      r0[0] = r1[0];
      if(r0[0] >= 0)
      cout << " " << operand1[j] << " = " << r0[0] << " [" <<
sixteenbits(r0) << "] " << endl;
      else
      cout << " " << operand1[j] << " = " << r0[0] << " [" <<
twoscomplement(r0) << "] " << endl;

}
```

```
void add(int r0[], int r1[], string operand1[], int j)

{

      r0[0] += r1[0];

      if(r0[0] >= 0)

      cout << " " << operand1[j] << " = " << r0[0] << " [" <<
sixteenbits(r0)  << "] "<< endl;

      else

      cout << " " << operand1[j] << " = " << r0[0] << " [" <<
twoscomplement(r0)  << "] "<< endl;

}

void sub(int r0[], int r1[], string operand1[], int j)

{

     r0[0] -= r1[0];

     if(r0[0] >= 0)

     cout << " " << operand1[j] << " = " << r0[0] << " [" <<
sixteenbits(r0)  << "] "<< endl;

     else

     cout << " " << operand1[j] << " = " << r0[0] << " [" <<
twoscomplement(r0)  << "] "<< endl;

}

//=============================================

void mul(int r0[], int r1[], string operand1[], int j)

{

     r0[0] *= r1[0];

     if(r0[0] >= 0)

     cout << " " << "rm:" << operand1[j] << " = " << r0[0] <<
" [" << thirtyTwobits(r0) << "]\n";

     else

     cout << " " << "rm:" << operand1[j] << " = " << r0[0] <<
" [" << twoscomplement32bits(r0) << "]\n";
```

```
}
void div(int r0[], int r1[], string operand1[], int j)
{
       int rem, remain[1]= {0};
       rem = r0[0];
       r0[0] /= r1[0];
       remain[0] = rem % r1[0];
       if(r0[0] >= 0)
       cout << " " << operand1[j] << " = " << r0[0] << " [" <<
sixteenbits(r0) << "]   Re: " << remain[0] << " [" <<
sixteenbits(remain)<<  "]\n";
       else
       cout << " " << operand1[j] << " = " << r0[0] << " [" <<
twoscomplement(r0) << "]  Re: " << remain[0] << " [" <<
sixteenbits(remain)<<  "]\n";
}
//================================================
void immMov(int r0[], int number1[], string operand1[], int j)
{
   r0[0] = number1[j];
   if (r0[0] >= 0)
   cout << " " << operand1[j] << " = " << r0[0] << " [" <<
sixteenbits(r0) << "] " << endl;
   else
   cout << " " << operand1[j] << " = " << r0[0] << " [" <<
twoscomplement(r0) << "] " << endl;
}
//================================================================
=
void immAdd(int r0[], int number1[], string operand1[], int j)
```

```
{
    r0[0] += number1[j]; //cout << operand1[j] << " = " <<
number1[j] << endl;
    if (r0[0] >= 0)
    cout << " " << operand1[j] << " = " << r0[0] << " [" <<
sixteenbits(r0) << "]" << endl;
    else
    cout << " " << operand1[j] << " = " << r0[0] << " [" <<
twoscomplement(r0) << "]" << endl;
}
void immSub(int r0[], int number1[], string operand1[], int j)
{
     r0[0] -= number1[j];
     if (r0[0] < 0)
     cout << " " << operand1[j] << " = " << r0[0] << " [" <<
twoscomplement(r0) << "]"  << endl;
     else
     cout << " " << operand1[j] << " = " << r0[0] << " [" <<
sixteenbits(r0)  << "]"  << endl;
}
void immMul(int r0[], int number1[], string operand1[], int j)
{
  r0[0] *= number1[j];
  if(r0[0] >= 0)
  cout << " " << "rm:" << operand1[j] << " = " << r0[0] << "
[" << thirtyTwobits(r0) << "]\n";
  else
  cout << " " << "rm:" << operand1[j] << " = " << r0[0] << "
[" << twoscomplement32bits(r0) << "]\n";
}
```

```
void immDiv(int r0[], int number1[], string operand1[], int j)

{

     int rem, remain[1] = {0};

     rem = r0[0];

     r0[0] /= number1[j];

     remain[0] = rem % number1[j];

     if(r0[0] >= 0)

     cout << " " << operand1[j] << " = " << r0[0] << " [" <<
sixteenbits(r0) << "]   Re: " << remain[0] << " [" <<
sixteenbits(remain)<<  "]\n";

     else

     cout << " " << operand1[j] << " = " << r0[0] << " [" <<
twoscomplement(r0) << "]  Re: " << remain[0] << " [" <<
sixteenbits(remain)<<  "]\n";

 }

int main()

{

    string s, val, k1 = "0000000000000000";

    int len, lennew;

    string opcode[50] = {" "};

    string operand1[50] = {" "};

    string operand2[50] = {" "};

    string operandval[50] = {" "};

    int number[50] = {0};

    int number1[50] = {0};

    int operanddata[50] = {0};

    int r0[1]= {0},r1[1] = {0}, r2[1] = {0}, r3[1] = {0},
r4[1] = {0}, r5[1] = {0}, r6[1] = {0}, r7[1]={0};

    int rm[1] = {0};
```

```
    int clock[50] = {0};
    int remain[1] = {0}, rem;
    bool flag[50] = {0};
    int i = 0, k[50] = {0}, x = 0;
    char cont;
    int instrcount = 0;
   // int memval[100] = {0};
   do
   {
      system("CLS");



    cout << " \n Details of the ISA:\n
====================\n";
    cout << " It is a 24-bit ISA and can handle any 16-bit or
8-bit Arithmetic Operations.\n";
    cout << " There are eight 16-bit GPRs (r0-r7) excluding
two 16-bit additional registers;\n";
    cout << " 'rm' and 're', are for storing 16-bit upper
result after a \n";
    cout << " multiplication and 16-bit remainder after a
division respectively.\n";
    cout << "\n  Select the opcode <'mov', 'add', 'sub',
'mul', 'div' or 'end' to end code>:\n";
    cout << "  Then select the first operand <r0, r1, r2, r3,
r4, r5, r6, or r7> where r0 is\n";
    cout << "  a base register. Select the second operand
<r1,.....,r7 or a decimal value>\n";
    cout << "  For example, 'mov r0 mem' or 'mov r1 23' or
type 'end 0 0' end instruction.\n\n";
```

```
    cout << " Enter the memory location for a load operation
<0 - 99> or press -1: ";


    cin >> mem;


     if (mem != -1)

     {

            cout << " Input a data to store in the memory
location " << mem << " : ";

             cin >> loaddata[0];

             cout << endl;

     }

    cout << endl;

    do{


    cout << "   ";

    cin >> opcode[i] >> operand1[i] >> operand2[i];

    if(opcode[i] == "end")

    break;


                if((operand2[i] != "r0") && (operand2[i] !=
"r1") && (operand2[i] != "r2") &&

                (operand2[i] != "r3") && (operand2[i] != "r4")
&& (operand2[i] != "r5") &&

                (operand2[i] != "r6") && (operand2[i] != "r7")
&& (operand2[i] != "[r0]"))

                {

                flag[i] = 1;

                // atoi function on operand2[i]
```

```
                number[i] =  atoi(operand2[i].c_str()); //
number[i] holds the interger oprands

                                    // if number[i] = 0, means no
integer oprands

                k[x] = i; // array k holds index values of
number[i]

                    // for keep track of number operands for
encoding purpose

                operandval[x] = operand1[i];

                //need to put a register value here for
immediate operation

                operanddata[x] = number[i];

                }


               else

              { k[x] = 51; // means non integer operand in
k[x]

                 // cout << "k[" << x << "]= " << k[x] <<
endl;

                   }

    ++i; // total number of instructions

    ++x;

       ++instrcount; // get the instruction count for pipeline

    }while(1);


    //encoded for of instruction

    // instructions: opcode + operand1 + operand2(number)


    for(int j = 0; j < i; j++)

    {
```

```
            number1[j] = number[j];  // keep copy of immediate
operands into number1[]
                        }


    string encode1[50] = {" "}; // for opcode + control bit
    string encode2[50] = {" "}; // for first operand
    string encode3[50] = {" "}; // for second operand and
integer data


   float c1 = 0, c2 = 0, c3 = 0, c4 = 0, c5 = 0;


   for(int y = 0; y < i; y++)// i is the size of instructions
    {


    if(opcode[y] == "mov")
    { if(flag[y]) // adding control bit along with opcode
      encode1[y] = "00001";
      else
      encode1[y] = "00000";
      clock[y] = 1;
    ++c1;
    }
    else if(opcode[y] == "add")
    { if(flag[y])
    encode1[y] = "00011";
    else
    encode1[y] = "00010";
    //c2 += 2;
```

```
    clock[y] = 2;
    ++c2;
    }
    else if(opcode[y] == "sub")
    { if(flag[y])
    encode1[y] = "00101";
    else
    encode1[y] = "00100";
    ++c3;
    clock[y] = 3;
    }
    else if(opcode[y] == "mul")
    { if(flag[y])
    encode1[y] = "00111";
    else
    encode1[y] = "00110";
    ++c4;
    clock[y] = 4;
    }
    else //if(opcode[y] == "div")
    { if(flag[y])
    encode1[y] = "01001";
    else
    encode1[y] = "01000";
    ++c5;
    clock[y] = 5;
     }      // it is better to add control bit along with the
opcode
```

```
 }
    //==================================
    // for encoding the operand1
     for(int y = 0; y < i; y++)// i is the size of
instructions
    {
      if(operand1[y] == "r0")
      encode2[y] = "000";
      else if(operand1[y] == "r1")
      encode2[y] = "001";
      else if(operand1[y] == "r2")
      encode2[y] = "010";
      else if(operand1[y] == "r3")
      encode2[y] = "011";
      else if(operand1[y] == "r4")
      encode2[y] = "100";
      else if(operand1[y] == "r5")
      encode2[y] = "101";
      else if(operand1[y] == "r6")
      encode2[y] = "110";
      else
      encode2[y] = "111";
    }
    //Encoding operand2; either register operand or integer
data
    //=====================================================
```

```
    int a[50], b = 0, rev = 0, bin[50] = {0}, number2[1] =
{0};
    for(int y = 0; y < i; y++)// i is the size of instructions
    {

    if(flag[y]) // if the second operand is an integer data,
then convert that into a 16-bit
    {           //binary
                    number2[0] = number[y];
            while(number[y])
            {
                  a[b] = number[y] % 2;
                  number[y] /= 2;
                  ++b;
                  }  //end of while

            for(int n = b-1; n >=0; n--)
               {
              rev = rev * 10 + a[n]; // binary form of
number[y];
              } // end of for

            bin[y] = rev; // binary values stored in bin[]
            rev = 0;
            b = 0;

            //next convert the integer binary form into 16-
character string form
            //and put it into encode3[] array
```

```
   stringstream out; //convert int to binary
   out << bin[y]; // convert int to binary
   s = out.str(); // convert int to binary
   len = s.length();
   lennew = 16 - len;
   for(int j1 = 0; j1 < lennew; j1++)
    val += k1.at(j1);


    val += s; // final 16- character string form of binary is
available in val
    encode3[y] = val;
    val = ""; // make val empty for next value


    if (number2[0] < 0)


    encode3[y] = twoscomplement(number2);  // for negative
immediate numbers
    number2[0] = 0;
    } // end of if
          //else print 13-bit zero along with second register
operand
      else
      {
          if (operand2[y] == "r0") // I need to remove this
because r0 is a base register
          encode3[y] = "0000000000000000"; // operand2, except
the integer operand in encode3[]
          else if (operand2[y] == "r1")
          encode3[y] = "0010000000000000";
```

```
            else if (operand2[y] == "r2")
            encode3[y] = "0100000000000000";
            else if (operand2[y] == "r3")
            encode3[y] = "0110000000000000";
            else if (operand2[y] == "r4")
            encode3[y] = "1000000000000000";
            else if (operand2[y] == "r5")
            encode3[y] = "1010000000000000";
            else if (operand2[y] == "r6")
            encode3[y] = "1100000000000000";
            else if (operand2[y] == "r7")
            encode3[y] = "1110000000000000";
            else if (operand2[y] == "[r0]")
            encode3[y] = sixteenbits(loaddata); // a 16-bit
memory value indicated by [r0]
            }
   }  //[r0] should hold the 16-bit binary of the value
loaddata


   //For displaying instructions.........
    cout << "\n        PC    Decoded:   Encoded
instructions(24-bit):   Clock cycles\n\n";


    for (int j = 0; j < i; j++)
    {
    cout << setw(7) << " " << "PC[" << j << "]-> " << left;
    cout << setw(2) << opcode[j] << " ";
    cout << operand1[j] << ", ";
```

```
    cout << operand2[j] << ": ";
    cout << setw(6) << encode1[j] ; //<< " " ;
    cout << setw(6) << encode2[j] ; //<< " " ;
    cout << setw(6) << encode3[j] << "  "; //<< " ";
    cout << setw(6)<< "  " << clock[j] << endl;
    }

    cout << "\n\n After the program execution contents of the
regiters are.....\n\n\n";

    //Instruction execution engine
    for(int j = 0; j < i; j++)
    {
           if(flag[j] && k[j] != 51) // if flag[j]==1, then
the operations are immediate type
           {
                if(opcode[j] == "mov" && operand1[j] == "r0")
                     {
                        immMov(r0, number1, operand1, j);

                        }
          else if(opcode[j] == "mov" && operand1[j] == "r1")
                      {
                       immMov(r1, number1, operand1, j);
                      }
          else if(opcode[j] == "mov" && operand1[j] == "r2")
                      {
                       immMov(r2, number1, operand1, j);
```

```
                    }
        else if(opcode[j] == "mov" && operand1[j] == "r3")
                    {
                     immMov(r3, number1, operand1, j);
                    }
       else if(opcode[j] == "mov" && operand1[j] == "r4")
                    {
                     immMov(r4, number1, operand1, j);
                    }
     else if(opcode[j] == "mov" && operand1[j] == "r5")
                    {
                     immMov(r5, number1, operand1, j);
                    }

else if(opcode[j] == "mov" && operand1[j] == "r6")
                    {
                     immMov(r6, number1, operand1, j);
                    }

  else if(opcode[j] == "mov" && operand1[j] == "r7")
                    {
                     immMov(r7, number1, operand1, j);
                    }
     else if(opcode[j] == "add" && operand1[j] == "r0")
                    {
                     immAdd(r0, number1, operand1, j);
                    }
```

```
        else if(opcode[j] == "add" && operand1[j] == "r1")
                    {
                       immAdd(r1, number1, operand1, j);
                    }
      else if(opcode[j] == "add" && operand1[j] == "r2")
                    {
                       immAdd(r2, number1, operand1, j);
                    }

     else if(opcode[j] == "add" && operand1[j] == "r3")
                    {
                       immAdd(r3, number1, operand1, j);
                    }
      else if(opcode[j] == "add" && operand1[j] == "r4")
                    {
                       immAdd(r4, number1, operand1, j);
                    }
else if(opcode[j] == "add" && operand1[j] == "r5")
                    {
                       immAdd(r5, number1, operand1, j);
                    }
   else if(opcode[j] == "add" && operand1[j] == "r6")
                    {
                       immAdd(r6, number1, operand1, j);
                    }
   else if(opcode[j] == "add" && operand1[j] == "r7")
                    {
```

```
                    immAdd(r7, number1, operand1, j);
                }
// sub operation should show 2's complement negative result
    else if(opcode[j] == "sub" && operand1[j] == "r0")
                {
                 immSub(r0, number1, operand1, j);
                 }

  else if(opcode[j] == "sub" && operand1[j] == "r1")
                {
                  immSub(r1, number1, operand1, j);
                }

    else if(opcode[j] == "sub" && operand1[j] == "r2")
                {
                  immSub(r2, number1, operand1, j);
                }

  else if(opcode[j] == "sub" && operand1[j] == "r3")
                {
                  immSub(r3, number1, operand1, j);
                }
  else if(opcode[j] == "sub" && operand1[j] == "r4")
                {
                  immSub(r4, number1, operand1, j);
                }
   else if(opcode[j] == "sub" && operand1[j] == "r5")
```

```
                    {
                      immSub(r5, number1, operand1, j);
                    }
          else if(opcode[j] == "sub" && operand1[j] == "r6")
                    {
                      immSub(r6, number1, operand1, j);
                    }
     else if(opcode[j] == "sub" && operand1[j] == "r7")
                    {
                      immSub(r7, number1, operand1, j);
                    }
                //multiplication doubles the results! (16-bit
x 16-bit = 32-bit result)
      else if(opcode[j] == "mul" && operand1[j] == "r0")
                    {
                      immMul(r0, number1, operand1, j);
                    }
        else if(opcode[j] == "mul" && operand1[j] == "r1")
                    {
                      immMul(r1, number1, operand1, j);
                    }

         else if(opcode[j] == "mul" && operand1[j] == "r2")
                    {
                      immMul(r2, number1, operand1, j);
                    }

           else if(opcode[j] == "mul" && operand1[j] == "r3")
```

```
                    {
                      immMul(r3, number1, operand1, j);

                    }
       else if(opcode[j] == "mul" && operand1[j] == "r4")
                    {
                      immMul(r4, number1, operand1, j);
                    }
       else if(opcode[j] == "mul" && operand1[j] == "r5")
                    {
                      immMul(r5, number1, operand1, j);
                    }
    else if(opcode[j] == "mul" && operand1[j] == "r6")
                    {
                      immMul(r6, number1, operand1, j);
                    }
    else if(opcode[j] == "mul" && operand1[j] == "r7")
                    {
                      immMul(r7, number1, operand1, j);
                    }
else if(opcode[j] == "div" && operand1[j] == "r0")
                    {
                      immDiv(r0, number1, operand1, j);
                      }
     else if(opcode[j] == "div" && operand1[j] == "r1")
                    {
                      immDiv(r1, number1, operand1, j);
```

```
                    }

    else if(opcode[j] == "div" && operand1[j] == "r2")
                    {
                      immDiv(r2, number1, operand1, j);
                      }

  else if(opcode[j] == "div" && operand1[j] == "r3")
                    {
                      immDiv(r3, number1, operand1, j);
                      }
 else if(opcode[j] == "div" && operand1[j] == "r4")
                    {
                      immDiv(r4, number1, operand1, j);
                      }
else if(opcode[j] == "div" && operand1[j] == "r5")
                    {
                      immDiv(r5, number1, operand1, j);
                      }
else if(opcode[j] == "div" && operand1[j] == "r6")
                    {
                      immDiv(r6, number1, operand1, j);
                      }

 else if(opcode[j] == "div" && operand1[j] == "r7")
                    {
                      immDiv(r7, number1, operand1, j);
```

```
                        }

         }  //end of if statement

       else  // non-immediate operand operation on registers

       {

if(opcode[j] == "mov" && operand1[j] == "r0" && operand2[j] ==
"r1")

            {  // put the immediate data into register r0

            movr0(r0, r1, operand1, j);

             }

             else if(opcode[j] == "mov" && operand1[j] == "r0"
&& operand2[j] == "r2" )

             {

              movr0(r0, r2, operand1, j);

             }

           else if(opcode[j] == "mov" && operand1[j] == "r0"
&& operand2[j] == "r3" )

             {

             movr0(r0, r3, operand1, j);

             }

           else if(opcode[j] == "mov" && operand1[j] == "r0"
&& operand2[j] == "r4" )

             {

             movr0(r0, r4, operand1, j);
```

```
            }
          else if(opcode[j] == "mov" && operand1[j] == "r0"
&& operand2[j] == "r5" )
            {
            movr0(r0, r5, operand1, j);

            }
          else if(opcode[j] == "mov" && operand1[j] == "r0"
&& operand2[j] == "r6" )
            {
            movr0(r0, r6, operand1, j);

            }
          else if(opcode[j] == "mov" && operand1[j] == "r0"
&& operand2[j] == "r7" )
            {
            movr0(r0, r7, operand1, j);
            ============================
            ----------------------------
            }
            }
          //==============LOAD ACTIONS ENDS ==============

          else if(opcode[j] == "add" && operand1[j] == "r0"
&& operand2[j] == "r1" )
            {
             add(r0, r1, operand1, j);

            }
```

```
          else if(opcode[j] == "add" && operand1[j] == "r0"
&& operand2[j] == "r2" )

            {

            add(r0, r2, operand1, j);



            }


          else if(opcode[j] == "add" && operand1[j] == "r0"
&& operand2[j] == "r3" )

            {

            add(r0, r3, operand1, j);



            }

          else if(opcode[j] == "add" && operand1[j] == "r0"
&& operand2[j] == "r4" )

            {

            add(r0, r4, operand1, j);



            }

          else if(opcode[j] == "add" && operand1[j] == "r0"
&& operand2[j] == "r5" )

            {

            add(r0, r5, operand1, j);



            }

         else if(opcode[j] == "add" && operand1[j] == "r0" &&
operand2[j] == "r6" )

            {
```

```
            add(r0, r6, operand1, j);


            }
          else if(opcode[j] == "add" && operand1[j] == "r0" &&
operand2[j] == "r7" )
            {
            add(r0, r7, operand1, j);
            }


          else if(opcode[j] == "add" && operand1[j] == "r1" &&
operand2[j] == "r0" )
            {
            add(r1, r0, operand1, j);


            }
        else if(opcode[j] == "add" && operand1[j] == "r1" &&
operand2[j] == "r2" )
            {
            add(r1, r2, operand1, j);


            }
          else if(opcode[j] == "add" && operand1[j] == "r1" &&
operand2[j] == "r3" )
            {
            add(r1, r3, operand1, j);


            }
```

```
         else if(opcode[j] == "add" && operand1[j] == "r1" &&
operand2[j] == "r4" )

             {

             add(r1, r4, operand1, j);



             }

         else if(opcode[j] == "add" && operand1[j] == "r1" &&
operand2[j] == "r5" )

             {

             add(r1, r5, operand1, j);



             }

       else if(opcode[j] == "add" && operand1[j] == "r1" &&
operand2[j] == "r6" )

             {

             add(r1, r6, operand1, j);



             }

          else if(opcode[j] == "add" && operand1[j] == "r1" &&
operand2[j] == "r7" )

             {

             add(r1, r7, operand1, j);



             }

          //=======================================

          else if(opcode[j] == "add" && operand1[j] == "r2" &&
operand2[j] == "r0" )

             {

             add(r2, r0, operand1, j);
```

```
            }


          else if(opcode[j] == "add" && operand1[j] == "r2" &&
operand2[j] == "r1" )

            else if(opcode[j] == "div" && operand1[j] == "r5"
&& operand2[j] == "r2" )

            {

            div(r5, r2, operand1, j);

            }

            else if(opcode[j] == "div" && operand1[j] == "r5"
&& operand2[j] == "r3" )

            {

            div(r5, r3, operand1, j);

            }

            else if(opcode[j] == "div" && operand1[j] == "r5"
&& operand2[j] == "r4" )

            {

            div(r5, r4, operand1, j);

            }

            else if(opcode[j] == "div" && operand1[j] == "r5"
&& operand2[j] == "r6" )

            {

            div(r5, r6, operand1, j);

            }

            else if(opcode[j] == "div" && operand1[j] == "r5"
&& operand2[j] == "r7" )

            {

            div(r5, r7, operand1, j);

            }
```

```
//============================================================
==========

            else if(opcode[j] == "div" && operand1[j] == "r6"
&& operand2[j] == "r0" )

            {

            div(r6, r0, operand1, j);

            }

             else if(opcode[j] == "div" && operand1[j] ==
"r6" && operand2[j] == "r1" )

            {

            div(r6, r1, operand1, j);

            }

             else if(opcode[j] == "div" && operand1[j] ==
"r6" && operand2[j] == "r2" )

            {

            div(r6, r2, operand1, j);

            }

             else if(opcode[j] == "div" && operand1[j] ==
"r6" && operand2[j] == "r3" )

            {

            div(r6, r3, operand1, j);

            }

             else if(opcode[j] == "div" && operand1[j] ==
"r6" && operand2[j] == "r4" )

            {

            div(r6, r4, operand1, j);

            }

             else if(opcode[j] == "div" && operand1[j] ==
"r6" && operand2[j] == "r5" )
```

```
            {
            div(r6, r5, operand1, j);
            }
             else if(opcode[j] == "div" && operand1[j] ==
"r6" && operand2[j] == "r7" )
            {
            div(r6, r7, operand1, j);
            }

//============================================================
=============

             else if(opcode[j] == "div" && operand1[j] ==
"r7" && operand2[j] == "r0" )
            {
            div(r7, r0, operand1, j);
            }

            else if(opcode[j] == "div" && operand1[j] == "r7"
&& operand2[j] == "r1" )
            {
            div(r7, r1, operand1, j);
            }
            else if(opcode[j] == "div" && operand1[j] == "r7"
&& operand2[j] == "r2" )
            {
            div(r7, r2, operand1, j);
            }
            else if(opcode[j] == "div" && operand1[j] == "r7"
&& operand2[j] == "r3" )
```

```
            {
            div(r7, r3, operand1, j);
            }
            else if(opcode[j] == "div" && operand1[j] == "r7"
&& operand2[j] == "r4" )
            {
            div(r7, r4, operand1, j);
            }
            else if(opcode[j] == "div" && operand1[j] == "r7"
&& operand2[j] == "r5" )
            {
            div(r7, r5, operand1, j);
            }
            else if(opcode[j] == "div" && operand1[j] == "r7"
&& operand2[j] == "r6" )
            {
            div(r7, r6, operand1, j);
            }
            //==========================


        } // end of else


    }// end of outter loop


    //==============CPI calculation======================
   float s1 = 0, s2 = 0, s3 = 0, s4 = 0, s5 = 0, CPI = 0;
   cout << "  \n CPI of the program..........\n\n";
```

```
    for(int f = 0; f < i; f++)
    {
            if(clock[f] == 1)
            s1 = clock[f];
            else if(clock[f] == 2)
            s2 = clock[f];
            else if(clock[f] == 3)
            s3 = clock[f];
            else if(clock[f] == 4)
             s4 = clock[f];
            else
            s5 = clock[f];
   }

    if(opcode[0] == "end")
    cout << " CPI = " << 0 << endl;
    else
    {
    CPI = (s1*c1+s2*c2+s3*c3+s4*c4+s5*c5)/(c1+c2+c3+c4+c5);
    cout << setw(8) << fixed << setprecision(2) << " CPI = "
<< CPI << endl;
    }

    k1 = "0000000000000000";
    i = 0;
    x = 0;
```

```
   cout << "\n Pipelined Execution of the Program\n
=====================================";

   cout << " \n It is assumed that the CPU has 5-pipeline
stages: IF (Instruction Fetch)";

   cout << " \n ID (Instruction decoding, EX (Execution), MEM
(Load) and WB (Write back).";

   cout << " \n And each stage completes within one clock
cycle and RAW hazard (any) will be";

   cout << " \n solved with either FORWARDING or STALL stages
(due to load instruction).\n\n ";


  // cout << " There are " << instrcount << " instructions in
the program.\n";

   // Pipeline simulation=========================

   if (instrcount != 0)

   {

   cout << setw(21) << "   ";

   for (int j2 = 1; j2 <= ik1; j2++)

   {


       cout << setw(3) << j2  << "  ";

   }


  cout << endl;

  }

  //=======================================


   for(int j1 = 0; j1 < instrcount; j1++)

   {
```

```
   oplength = operand2[j1].length();


   if(oplength >= 3)

   {

   cout << "  " <<  j1 + 1 << ". " << opcode[j1] << "  " <<
operand1[j1] << " " << operand2[j1] << " : " ;

   cout << setw(setword) << right;

   cout <<  " IF | ID | EX | MEM | WB";

   }


   else

   {

    cout << "  " << j1 + 1 << ". " << opcode[j1] << "  " <<
operand1[j1] << " " << operand2[j1] << "   : " ;

   // cout.width(setword);

   cout << setw(setword) << right;

    cout <<  " IF | ID | EX | MEM | WB";

   }


   setword = setval;

          setval += 5; //4;

   cout << endl;

   }



   bool raw = 0; // boolean for detecting a raw hazard

   int raw1[20] = {0};

   int j11 = 0;
```

```
   string rowreg[20] = {" "};


   for(int j1 = 0; j1 < instrcount; j1++)

   {

    if (operand1[j1] == operand2[j1 + 1] || (operand1[j1] ==
"r0" && operand2[j1 + 1] == "[r0]" ))

    {

       raw = 1;

       raw1[j11] = j1 + 1;

       rowreg[j11] = operand1[j1];

      ++j11;

    }




   }

   cout << "\n\n RAW hazard details:\n
=====================\n";


  if(!bloody)

  {

   if (raw == 0)

   {

           cout << " \n There is no RAW hazard detected.\n";


}


   else

   {
```

```
   for(int j1 = 0; j1 < j11; j1++)
   {
            cout << " " << rowreg[j1] << " in instructions " <<
raw1[j1] << " and " << raw1[j1] + 1 << " caused RAW
hazard.\n";

            }
   }
   }
   else
   {

   cout << " Error! Memory is not defined in register r0!!\n";
   }
      cout << endl;
  //==PIPELINE ENGINE FOR STALL DETECTION==========

   string str1 =  " IF   |";
   string stall = " STALL|";
   string str2 =  " ID   |";
   string str3 =  " EX   |";
   string str4 =  " MEM  |";
   string str5 =  " WB    ";
   string pipe;
   string pipeline[50];
   int pipecount = 5; // indicates 5-pipe stages
   int p1 = 0;
```

```
   int setword = 7;


         // this algorithm will keep all the stages into the
array "pipeline[]"


           for(int j1 = 0; j1 < instrcount; j1++) // j1 = 1
because the first inst has no stall

           {

                 if (j1 == 0) // starting of an instruction
has no hazards! // 0

                 {

                 pipe = str1 + str2 + str3 + str4 + str5;

                 pipeline[p1] = pipe;

                 ++pipecount;

                 ++p1;

                 }

               else if((operand2[j1] != operand1[j1-1]) &&
(operand2[j1-1] != "[r0]") && !stallflag)

               {

                    pipe = str1 + str2 + str3 + str4 + str5;

                    pipeline[p1] = pipe;

                    ++pipecount;

                    ++p1;

               }

                  else if((operand2[j1] == "[r0]") &&
(operand1[j1-1] == "r0") && (operand1[j1] != operand2[j1-1]))
{

                              pipe = str1 + str2 + str3 + str4
+ str5;

                                pipeline[p1] = pipe;
```

```
                                ++pipecount;
                                ++p1;
                    }
 else if (operand2[j1] == operand1[j1-1] && operand2[j1-1] ==
"[r0]")                  {
                        stallflag = 1;
                        pipe = str1 + str2 + stall + str3 +
str4 + str5;
                                pipeline[p1] = pipe;
                                pipecount += 1;
                                ++p1;
                  } else if ((operand2[j1] == operand2[j1-1]
|| operand2[j1] != operand2[j1-1]) && (operand2[j1-1] ==
operand1[j1-2]) && (stallflag)) // stall
                  {
          pipe = str1 + stall + str2 + str3 + str4 + str5;
                                pipeline[p1] = pipe;
                                //stallflag = 0;
                                newflag = 1;
                                pipecount += 1;
                                ++p1;

                  }
                else if ((operand2[j1-1] == operand2[j1-2] ||
operand2[j1-1] != operand2[j1-2]) && (operand2[j1-2] ==
operand1[j1-3]) && stallflag) // stall
                  {
    pipe = stall + str1 + str2 + str3 + str4 + str5;
                                pipeline[p1] = pipe;
                                newflag = 1;
```

```
                                ++pipecount;
                                ++p1;
                    }

            else if ((operand2[j1] == operand1[j1-1]) &&
(operand2[j1-1] == "[r0]") && (operand2[j1-2] == operand2[j1-
3]) && (operand2[j1-3] == operand1[j1-4]) && (operand2[j1-4]
== "[r0]") && newflag ) // stall
                    {
     pipe = stall + str1 + str2 + stall + str3 + str4 + str5;
                                pipeline[p1] = pipe;
                                ++pipecount;
                                ++p1;
                    }
          else if ((operand1[j1-1] == operand1[j1-2]) &&
stallflag && newflag) // stall
                    {
   pipe =   stall + str1 + stall + str2 + str3 + str4 + str5;
                                pipeline[p1] = pipe;
                                pipecount += 2;
                                ++p1;
                    }
           else if ((operand1[j1-1] == operand2[j1-2]) &&
(operand1[j1-1] == operand1[j1-3]) && stallflag && newflag)
{

                        pipe =   stall + stall + str1 + str2 +
str3 + str4 + str5;
                                pipeline[p1] = pipe;
                                pipecount += 2;
```

```
                                ++p1;
                    }

          else if ((operand2[j1] != operand1[j1-1]) &&
(operand2[j1-1] != "[r0]"))
          {
              pipe =            str1 + str2 + str3 + str4 +
str5;
                                pipeline[p1] = pipe;
                                ++pipecount;
                                ++p1;

           }
   else if ((operand2[j1] != operand1[j1-1]) &&
(operand2[j1+1] != operand1[j1])&& (operand2[j1-1] != "[r0]"))
          {
              pipe =  str1 + str2 + str3 + str4 + str5;
                                pipeline[p1] = pipe;
                                ++pipecount;
                                ++p1; }}

//============================================================
            for(int i1 = 0; i1 < p1; i1++)
           {

                    if (pipeline[i1].length() == 35)
                    {
                     cout << setw(setword + 35) << right <<
pipeline[i1] << "\n";
```

```
                    setword += 7;
                    }
                    else if (pipeline[i1].length() == 42)
                    {
                    cout << setw(setword + 42) << right <<
pipeline[i1] << "\n";
                    setword += 7;
                    }

            //cout << pipeline[i1].length() << endl;
            }
        //====================================
        //PIPELINE STAGE SEQUENCE DISPLAY UNIT
        //====================================

 cout << "\n\n";
 if(!bloody)
 {
 if (instrcount != 0)
   {
   cout << setw(19) << "   ";
   for (int j2 = 1; j2 <= pipecount; j2++)
   {
              cout << setw(5) << j2  << "  ";
   }

  cout << endl;
  }
```

```
      //------------------------

   for(int j1 = 0; j1 < instrcount; j1++)

   {


   int oplength = operand2[j1].length();


   if(oplength >= 3)

   {

   cout <<  j1 + 1 << ". " << opcode[j1] << "  " <<
operand1[j1] << " " << operand2[j1] << " : " ;




    if (pipeline[j1].length() == 35)

                   {

                    cout << setw(setword + 27) << right <<
pipeline[j1] << "\n";

                    setword += 7;

                    }

                    else if (pipeline[j1].length() == 42)

                    {

                    cout << setw(setword + 34) << right <<
pipeline[j1] << "\n";

                    setword += 7;

                    }

   }


   else

   {
```

```
    cout <<  j1 + 1 << ". " << opcode[j1] << "  " <<
operand1[j1] << " " << operand2[j1] << "   : " ;

    if (pipeline[j1].length() == 35)

                    {

                     cout << setw(setword + 27) << right <<
pipeline[j1] << "\n";

                     setword += 7;

                     }

                     else if (pipeline[j1].length() == 42)

                     {

                     cout << setw(setword + 34) << right <<
pipeline[j1] << "\n";

                     setword += 7;

                     }

   }









   }

}

    if (instrcount != 0)

    cout << "\n The CPU uses " << pipecount << " clock cycles
for its pipeline execution.\n";

   instrcount = 0;// initialize the instruction count

   bloody = 0;

   cout << "\n Press 'y' to continue or 'n' to stop the
simulation : ";

   cin >> cont;
```

```
 }while(cont != 'n' && cont != 'N'); // end first while


system("PAUSE");
return 0;
}
```