```
// Copyright 2000 Vk-Cic
// Author: Vahe Karamian
// Date:   12/19/2000
// Version 1.0
//---------------------------------------------------------------------------

#include <vcl.h>
#pragma hdrstop
USERES("jasm.res");
USEFORM("mainWin.cpp", MDIfrmMain);
USEFORM("newWin.cpp", MDICfrmNew);
USEFORM("aboutWin.cpp", AboutBox);
USEFORM("asmWin.cpp", MDICfrmAssembler);
USEFORM("instructionWin.cpp", frmInstructions);
//---------------------------------------------------------------------------
WINAPI WinMain(HINSTANCE, HINSTANCE, LPSTR, int)
{
   try
   {
       Application->Initialize();
       Application->CreateForm(__classid(TMDIfrmMain), &MDIfrmMain);
       Application->CreateForm(__classid(TMDICfrmNew), &MDICfrmNew);
       Application->CreateForm(__classid(TAboutBox), &AboutBox);
       Application->CreateForm(__classid(TMDICfrmAssembler), &MDICfrmAssembler);
       Application->CreateForm(__classid(TfrmInstructions), &frmInstructions);
       Application->Run();
   }
   catch (Exception &exception)
   {
       Application->ShowException(&exception);
   }
   return 0;
}
//---------------------------------------------------------------------------

// Vahe Karamian - CS 365 - Project 3 - Assembler for JARC
// Filename: project3.cpp

#include <iostream.h>
#include <strstrea.h>
#include <fstream.h>
#include <string.h>
#include <stdlib.h>
#include <stdio.h>

// PROGRAM INSTRUCTION SET ------------
const int add  [5] = { 0, 0, 0, 0, 0 };
const int addi [5] = { 0, 0, 0, 0, 1 };
const int addm [5] = { 0, 0, 0, 1, 0 };
const int sub  [5] = { 0, 0, 0, 1, 1 };
const int subi [5] = { 0, 0, 1, 0, 0 };
const int subm [5] = { 0, 0, 1, 0, 1 };
const int or   [5] = { 0, 0, 1, 1, 0 };
const int ori  [5] = { 0, 0, 1, 1, 1 };
const int orm  [5] = { 0, 1, 0, 0, 0 };
const int and  [5] = { 0, 1, 0, 0, 1 };
const int andi [5] = { 0, 1, 0, 1, 0 };
const int andm [5] = { 0, 1, 0, 1, 1 };
const int xor  [5] = { 0, 1, 1, 0, 0 };
const int xori [5] = { 0, 1, 1, 0, 1 };
const int xorm [5] = { 0, 1, 1, 1, 0 };
const int not  [5] = { 0, 1, 1, 1, 1 };
const int slt  [5] = { 1, 0, 0, 0, 0 };
const int slti [5] = { 1, 0, 0, 0, 1 };
const int sltm [5] = { 1, 0, 0, 1, 0 };
const int beq  [5] = { 1, 0, 0, 1, 1 };
const int bne  [5] = { 1, 0, 1, 0, 0 };
const int j    [5] = { 1, 0, 1, 0, 1 };
const int lwm  [5] = { 1, 0, 1, 1, 0 };
const int swm  [5] = { 1, 0, 1, 1, 1 };
const int lwi  [5] = { 1, 1, 0, 0, 0 };
const int lws  [5] = { 1, 1, 0, 0, 1 };
const int show [5] = { 1, 1, 0, 1, 0 };
// PROGRAM INSTRUCTION SET ------------

char* firstConversion( char *convert );
char* secondConversion( char *convert );
char* thirdConversion( char *convert );
```

```
void  decode( char *convert, int value );
void  convertDecToBin( int value, int limit, int index );
void  convertBinToHex( int start, int end );

int i=0;
int output[16];

int  count = 0;
bool result = false;

int main( )
{
    ifstream dataFile( "source.mps" );
    if( !dataFile )
    {
        cerr<<"Unable to open source.mps"<<endl;
        exit(1);
    }

    ofstream higherBits( "higherBits.txt" );
    if( !higherBits )
    {
        cerr<<"Unable to open higherBits.txt"<<endl;
        exit( 1 );
    }

    ofstream lowerBits( "lowerBits.txt" );
    if( !lowerBits )
    {
        cerr<<"Unable to open lowerBits.txt"<<endl;
        exit( 1 );
    }

    char *convert = new char[20];

    while( !dataFile.eof( ) )
    {
        for( int x=0; x<16; x++ )
            output[x] = 1;

        dataFile.getline( convert, 20 );

        char *opCode = firstConversion ( convert );

        if( strcmp( opCode, "add" ) == 0 )
            decode( convert, 1 );

        if( strcmp( opCode, "addi" ) == 0 )
            decode( convert, 2 );

        if( strcmp( opCode, "addm" ) == 0 )
            decode( convert, 3 );

        if( strcmp( opCode, "sub" ) == 0 )
            decode( convert, 4 );

        if( strcmp( opCode, "subi" ) == 0 )
            decode( convert, 5 );

        if( strcmp( opCode, "subm" ) == 0 )
            decode( convert, 6 );

        if( strcmp( opCode, "or" ) == 0 )
            decode( convert, 7 );

        if( strcmp( opCode, "ori" ) == 0 )
            decode( convert, 8 );

        if( strcmp( opCode, "orm" ) == 0 )
            decode( convert, 9 );

        if( strcmp( opCode, "and" ) == 0 )
            decode( convert, 10 );

        if( strcmp( opCode, "andi" ) == 0 )
            decode( convert, 11 );

        if( strcmp( opCode, "andm" ) == 0 )
            decode( convert, 12 );
```

```
        if( strcmp( opCode, "xor" ) == 0 )
            decode( convert, 13 );

        if( strcmp( opCode, "xori" ) == 0 )
            decode( convert, 14 );

        if( strcmp( opCode, "xorm" ) == 0 )
            decode( convert, 15 );

        if( strcmp( opCode, "not" ) == 0 )
            decode( convert, 16 );

        if( strcmp( opCode, "slt" ) == 0 )
            decode( convert, 17 );

        if( strcmp( opCode, "slti" ) == 0 )
            decode( convert, 18 );

        if( strcmp( opCode, "sltm" ) == 0 )
            decode( convert, 19 );

        if( strcmp( opCode, "beq" ) == 0 )
            decode( convert, 20 );

        if( strcmp( opCode, "bne" ) == 0 )
            decode( convert, 21 );

        if( strcmp( opCode, "j" ) == 0 )
            decode( convert, 22 );

        if( strcmp( opCode, "lwm" ) == 0 )
            decode( convert, 23 );

        if( strcmp( opCode, "swm" ) == 0 )
            decode( convert, 24 );

        if( strcmp( opCode, "lwi" ) == 0 )
            decode( convert, 25 );

        if( strcmp( opCode, "lws" ) == 0 )
            decode( convert, 26 );

        if( strcmp( opCode, "show" ) == 0 )
            decode( convert, 27 );

        // Write to file
        for( int z=0; z<8; z++ )
            lowerBits<<output[z];
        lowerBits<<endl;

        for( int y=8; y<16; y++ )
            higherBits<<output[y];
        higherBits<<endl;

        // Convert to HEXADECIMAL
        cout<<"\t";
        convertBinToHex(  0,  3 );
        convertBinToHex(  4,  7 );
        convertBinToHex(  8, 11 );
        convertBinToHex( 12, 15 );
        cout<<endl;

        i = 0;
    }

    // Close the data file
    higherBits.close( );
    lowerBits.close( );
    dataFile.close( );

    return 0;
}

char* firstConversion( char *convert )
{
    ostrstream opcode;
    while( convert[i] != ' ' )
    {
```

```
        opcode<<convert[i];
        i++;
    }
    i++;
    opcode<<ends;
    char *opcodeChar = opcode.str( );

    return opcodeChar;
}

char* secondConversion( char *convert )
{
    ostrstream number;
    while( convert[i] != ' ' )
    {
        number<<convert[i];
        i++;
    }
    i++;
    number<<ends;
    char *numChar = number.str( );

    return numChar;
}

char* thirdConversion( char *convert )
{
    ostrstream number;
    while( convert[i] != '\n' )
    {
        number<<convert[i];
        i++;
    }
    number<<ends;
    char *numChar = number.str( );

    return numChar;
}

void decode( char *convert, int value )
{
    switch( value )
    {
        case 1:
        {
            // INSTRUCTION ADD
            // ADD RD, RS, RT

            char *rd = secondConversion( convert );
            char *rs = secondConversion( convert );
            char *rt = thirdConversion ( convert );

            int rdConvert = atoi( rd );
            int rsConvert = atoi( rs );
            int rtConvert = atoi( rt );

            cout<<"add\t"<<rdConvert<<"\t"<<rsConvert<<"\t"<<rtConvert<<"\t";

            // DECODE THE INSTRUCTION INTO BINARY
            for( int x=0; x<5; x++ )
                output[x] = add[x];

            convertDecToBin( rdConvert, 2, 5 );
            convertDecToBin( rsConvert, 2, 7 );
            convertDecToBin( rtConvert, 2, 9 );

            for( int z=0; z<16; z++ )
                cout<<output[z];
            // cout<<endl;

            break;
        }

        case 2:
        {
            // INSTRUCTION ADDI
            // ADDI RD, RS, IMM

            char *rd  = secondConversion( convert );
```

```
    char *rs  = secondConversion( convert );
    char *imm = thirdConversion ( convert );

    int rdConvert  = atoi( rd );
    int rsConvert  = atoi( rs );
    int immConvert = atoi( imm );

    cout<<"addi\t"<<rdConvert<<"\t"<<rsConvert<<"\t"<<immConvert<<"\t";

    // DECODE THE INSTRUCTION INTO BINARY
    for( int x=0; x<5; x++ )
        output[x] = addi[x];

    convertDecToBin( rdConvert,  2, 5 );
    convertDecToBin( rsConvert,  2, 7 );
    convertDecToBin( immConvert, 4, 9 );

    for( int z=0; z<16; z++ )
        cout<<output[z];
    // cout<<endl;

    break;
}

case 3:
{
    // INSTRUCTION ADDM
    // ADDM RD, RS, RAM

    char *rd  = secondConversion( convert );
    char *rs  = secondConversion( convert );
    char *ram = thirdConversion ( convert );

    int rdConvert  = atoi( rd );
    int rsConvert  = atoi( rs );
    int ramConvert = atoi( ram );

    cout<<"addm\t"<<rdConvert<<"\t"<<rsConvert<<"\t"<<ramConvert<<"\t";

    // DECODE INSTRUCTION INTO BINARY
    for( int x=0; x<5; x++ )
        output[x] = addm[x];

    convertDecToBin( rdConvert,  2, 5 );
    convertDecToBin( rsConvert,  2, 7 );
    convertDecToBin( ramConvert, 7, 9 );

    for( int z=0; z<16; z++ )
        cout<<output[z];
    // cout<<endl;

    break;
}

case 4:
{
    // INSTRUCTION SUB
    // SUB RD, RS, RT

    char *rd = secondConversion( convert );
    char *rs = secondConversion( convert );
    char *rt = thirdConversion ( convert );

    int rdConvert = atoi( rd );
    int rsConvert = atoi( rs );
    int rtConvert = atoi( rt );

    cout<<"sub\t"<<rdConvert<<"\t"<<rsConvert<<"\t"<<rtConvert<<"\t";

    // DECODE INSTRUCTION INTO BINARY
    for( int x=0; x<5; x++ )
        output[x] = sub[x];

    convertDecToBin( rdConvert, 2, 5 );
    convertDecToBin( rsConvert, 2, 7 );
    convertDecToBin( rtConvert, 2, 9 );

    for( int z=0; z<16; z++ )
        cout<<output[z];
```

```
    // cout<<endl;

    break;
}

case 5:
{
    // INSTRUCTION SUBI
    // SUBI RD, RS, IMM

    char *rd  = secondConversion( convert );
    char *rs  = secondConversion( convert );
    char *imm = thirdConversion ( convert );

    int rdConvert  = atoi( rd );
    int rsConvert  = atoi( rs );
    int immConvert = atoi( imm );

    cout<<"subi\t"<<rdConvert<<"\t"<<rsConvert<<"\t"<<immConvert<<"\t";

    // DECODE INSTRUCTION INTO BINARY
    for( int x=0; x<5; x++ )
        output[x] = subi[x];

    convertDecToBin( rdConvert,  2, 5 );
    convertDecToBin( rsConvert,  2, 7 );
    convertDecToBin( immConvert, 4, 9 );

    for( int z=0; z<16; z++ )
        cout<<output[z];
    // cout<<endl;

    break;
}

case 6:
{
    // INSTRUCTION SUBM
    // SUBM RD, RS, RAM

    char *rd  = secondConversion( convert );
    char *rs  = secondConversion( convert );
    char *ram = thirdConversion ( convert );

    int rdConvert  = atoi( rd );
    int rsConvert  = atoi( rs );
    int ramConvert = atoi( ram );

    cout<<"subm\t"<<rdConvert<<"\t"<<rsConvert<<"\t"<<ramConvert<<"\t";

    // DECODE INSTRUCTION INTO BINARY
    for( int x=0; x<5; x++ )
        output[x] = subm[x];

    convertDecToBin( rdConvert,  2, 5 );
    convertDecToBin( rsConvert,  2, 7 );
    convertDecToBin( ramConvert, 7, 9 );

    for( int z=0; z<16; z++ )
        cout<<output[z];
    // cout<<endl;

    break;
}

case 7:
{
    // INSTRUCTION OR
    // OR RD, RS, RT

    char *rd = secondConversion( convert );
    char *rs = secondConversion( convert );
    char *rt = thirdConversion ( convert );

    int rdConvert = atoi( rd );
    int rsConvert = atoi( rs );
    int rtConvert = atoi( rt );

    cout<<"or\t"<<rdConvert<<"\t"<<rsConvert<<"\t"<<rtConvert<<"\t";
```

```
    // DECODE INSTRUCTION INTO BINARY
    for( int x=0; x<5; x++ )
        output[x] = or[x];

    convertDecToBin( rdConvert, 2, 5 );
    convertDecToBin( rsConvert, 2, 7 );
    convertDecToBin( rtConvert, 2, 9 );

    for( int z=0; z<16; z++ )
        cout<<output[z];
    // cout<<endl;

    break;
}

case 8:
{
    // INSTRUCTION ORI
    // ORI RD, RS, IMM

    char *rd  = secondConversion( convert );
    char *rs  = secondConversion( convert );
    char *imm = thirdConversion ( convert );

    int rdConvert  = atoi( rd );
    int rsConvert  = atoi( rs );
    int immConvert = atoi( imm );

    cout<<"ori\t"<<rdConvert<<"\t"<<rsConvert<<"\t"<<immConvert<<"\t";

    // DECODE INSTRUCTION INTO BINARY
    for( int x=0; x<5; x++ )
        output[x] = ori[x];

    convertDecToBin( rdConvert,  2, 5 );
    convertDecToBin( rsConvert,  2, 7 );
    convertDecToBin( immConvert, 4, 9 );

    for( int z=0; z<16; z++ )
        cout<<output[z];
    // cout<<endl;

    break;
}

case 9:
{
    // INSTRUCTION ORM
    // ORM RD, RS, RAM

    char *rd  = secondConversion( convert );
    char *rs  = secondConversion( convert );
    char *ram = thirdConversion ( convert );

    int rdConvert  = atoi( rd );
    int rsConvert  = atoi( rs );
    int ramConvert = atoi( ram );

    cout<<"orm\t"<<rdConvert<<"\t"<<rsConvert<<"\t"<<ramConvert<<"\t";

    // DECODE INSTRUCTION INTO BINARY
    for( int x=0; x<5; x++ )
        output[x] = orm[x];

    convertDecToBin( rdConvert,  2, 5 );
    convertDecToBin( rsConvert,  2, 7 );
    convertDecToBin( ramConvert, 7, 9 );

    for( int z=0; z<16; z++ )
        cout<<output[z];
    // cout<<endl;

    break;
}

case 10:
{
    // INSTRUCTION AND
```

```
    // AND RD, RS, RT

    char *rd = secondConversion( convert );
    char *rs = secondConversion( convert );
    char *rt = thirdConversion ( convert );

    int rdConvert = atoi( rd );
    int rsConvert = atoi( rs );
    int rtConvert = atoi( rt );

    cout<<"and\t"<<rdConvert<<"\t"<<rsConvert<<"\t"<<rtConvert<<"\t";

    // DECODE INSTRUCTION INTO BINARY
    for( int x=0; x<5; x++ )
        output[x] = and[x];

    convertDecToBin( rdConvert, 2, 5 );
    convertDecToBin( rsConvert, 2, 7 );
    convertDecToBin( rtConvert, 2, 9 );

    for( int z=0; z<16; z++ )
        cout<<output[z];
    // cout<<endl;

    break;
}

case 11:
{
    // INSTRUCTION ANDI
    // ANDI RD, RS, IMM

    char *rd  = secondConversion( convert );
    char *rs  = secondConversion( convert );
    char *imm = thirdConversion ( convert );

    int rdConvert  = atoi( rd );
    int rsConvert  = atoi( rs );
    int immConvert = atoi( imm );

    cout<<"andi\t"<<rdConvert<<"\t"<<rsConvert<<"\t"<<immConvert<<"\t";

    // DECODE INSTRUCTION INTO BINARY
    for( int x=0; x<5; x++ )
        output[x] = andi[x];

    convertDecToBin( rdConvert,  2, 5 );
    convertDecToBin( rsConvert,  2, 7 );
    convertDecToBin( immConvert, 4, 9 );

    for( int z=0; z<16; z++ )
        cout<<output[z];
    // cout<<endl;

    break;
}

case 12:
{
    // INSTRUCTION ANDM
    // ANDM RD, RS, RAM

    char *rd  = secondConversion( convert );
    char *rs  = secondConversion( convert );
    char *ram = thirdConversion ( convert );

    int rdConvert  = atoi( rd );
    int rsConvert  = atoi( rs );
    int ramConvert = atoi( ram );

    cout<<"andm\t"<<rdConvert<<"\t"<<rsConvert<<"\t"<<ramConvert<<"\t";

    // DECODE INSTRUCTION INTO BINARY
    for( int x=0; x<5; x++ )
        output[x] = andm[x];

    convertDecToBin( rdConvert,  2, 5 );
    convertDecToBin( rsConvert,  2, 7 );
    convertDecToBin( ramConvert, 7, 9 );
```

```
    for( int z=0; z<16; z++ )
        cout<<output[z];
    // cout<<endl;

    break;
}

case 13:
{
    // INSTRUCTION XOR
    // XOR RD, RS, RT

    char *rd = secondConversion( convert );
    char *rs = secondConversion( convert );
    char *rt = secondConversion( convert );

    int rdConvert = atoi( rd );
    int rsConvert = atoi( rs );
    int rtConvert = atoi( rt );

    cout<<"xor\t"<<rdConvert<<"\t"<<rsConvert<<"\t"<<rtConvert<<"\t";

    // DECODE INSTRUCTION INTO BINARY
    for( int x=0; x<5; x++ )
        output[x] = xor[x];

    convertDecToBin( rdConvert, 2, 5 );
    convertDecToBin( rsConvert, 2, 7 );
    convertDecToBin( rtConvert, 2, 9 );

    for( int z=0; z<16; z++ )
        cout<<output[z];
    // cout<<endl;

    break;
}

case 14:
{
    // INSTRUCTION XORI
    // XORI RD, RS, IMM

    char *rd  = secondConversion( convert );
    char *rs  = secondConversion( convert );
    char *imm = thirdConversion ( convert );

    int rdConvert  = atoi( rd );
    int rsConvert  = atoi( rs );
    int immConvert = atoi( imm );

    cout<<"xori\t"<<rdConvert<<"\t"<<rsConvert<<"\t"<<immConvert<<"\t";

    // DECODE INSTRUCTION INTO BINARY
    for( int x=0; x<5; x++ )
        output[x] = xori[x];

    convertDecToBin( rdConvert,  2, 5 );
    convertDecToBin( rsConvert,  2, 7 );
    convertDecToBin( immConvert, 4, 9 );

    for( int z=0; z<16; z++ )
        cout<<output[z];
    // cout<<endl;

    break;
}

case 15:
{
    // INSTRUCTION XORM
    // XORM RD, RS, RAM

    char *rd  = secondConversion( convert );
    char *rs  = secondConversion( convert );
    char *ram = thirdConversion ( convert );

    int rdConvert  = atoi( rd );
    int rsConvert  = atoi( rs );
```

```
    int ramConvert = atoi( ram );

    cout<<"xorm\t"<<rdConvert<<"\t"<<rsConvert<<"\t"<<ramConvert<<"\t";

    // DECODE INSTRUCTION INTO BINARY
    for( int x=0; x<5; x++ )
        output[x] = xorm[x];

    convertDecToBin( rdConvert,  2, 5 );
    convertDecToBin( rsConvert,  2, 7 );
    convertDecToBin( ramConvert, 7, 9 );

    for( int z=0; z<16; z++ )
        cout<<output[z];
    // cout<<endl;

    break;
}

case 16:
{
    // INSTRUCTION NOT
    // NOT RD, RS

    char *rd = secondConversion( convert );
    char *rs = thirdConversion ( convert );

    int rdConvert = atoi( rd );
    int rsConvert = atoi( rs );

    cout<<"not\t"<<rdConvert<<"\t"<<rsConvert<<"\t\t";

    // DECODE INSTRUCTION INTO BINARY
    for( int x=0; x<5; x++ )
        output[x] = not[x];

    convertDecToBin( rdConvert, 2, 5 );
    convertDecToBin( rsConvert, 2, 7 );

    for( int z=0; z<16; z++ )
        cout<<output[z];
    // cout<<endl;

    break;
}

case 17:
{
    // INSTRUCTION SLT
    // SLT RD, RS, RT

    char *rd = secondConversion( convert );
    char *rs = secondConversion( convert );
    char *rt = thirdConversion ( convert );

    int rdConvert = atoi( rd );
    int rsConvert = atoi( rs );
    int rtConvert = atoi( rt );

    cout<<"slt\t"<<rdConvert<<"\t"<<rsConvert<<"\t"<<rtConvert<<"\t";

    // DECODE INSTRUCTION INTO BINARY
    for( int x=0; x<5; x++ )
        output[x] = slt[x];

    convertDecToBin( rdConvert, 2, 5 );
    convertDecToBin( rsConvert, 2, 7 );
    convertDecToBin( rtConvert, 2, 9 );

    for( int z=0; z<16; z++ )
        cout<<output[z];
    // cout<<endl;

    break;
}

case 18:
{
    // INSTRUCTION SLTI
```

```
    // SLTI RD, RS, IMM

    char *rd  = secondConversion( convert );
    char *rs  = secondConversion( convert );
    char *imm = thirdConversion ( convert );

    int rdConvert  = atoi( rd );
    int rsConvert  = atoi( rs );
    int immConvert = atoi( imm );

    cout<<"slti\t"<<rdConvert<<"\t"<<rsConvert<<"\t"<<immConvert<<"\t";

    // DECODE INSTRUCTION INTO BINARY
    for( int x=0; x<5; x++ )
        output[x] = slti[x];

    convertDecToBin( rdConvert,  2, 5 );
    convertDecToBin( rsConvert,  2, 7 );
    convertDecToBin( immConvert, 4, 9 );

    for( int z=0; z<16; z++ )
        cout<<output[z];
    // cout<<endl;

    break;
}

case 19:
{
    // INSTRUCTION SLTM
    // SLTM RD, RS, RAM

    char *rd  = secondConversion( convert );
    char *rs  = secondConversion( convert );
    char *ram = thirdConversion ( convert );

    int rdConvert  = atoi( rd );
    int rsConvert  = atoi( rs );
    int ramConvert = atoi( ram );

    cout<<"sltm\t"<<rdConvert<<"\t"<<rsConvert<<"\t"<<ramConvert<<"\t";

    // DECODE INSTRUCTION INTO BINARY
    for( int x=0; x<5; x++ )
        output[x] = sltm[x];

    convertDecToBin( rdConvert,  2, 5 );
    convertDecToBin( rsConvert,  2, 7 );
    convertDecToBin( ramConvert, 7, 9 );

    for( int z=0; z<16; z++ )
        cout<<output[z];
    // cout<<endl;

    break;
}

case 20:
{
    // INSTRUCTION BEQ
    // BEQ RS, RT, ROM

    char *rs  = secondConversion( convert );
    char *rt  = secondConversion( convert );
    char *rom = thirdConversion ( convert );

    int rsConvert  = atoi( rs );
    int rtConvert  = atoi( rt );
    int romConvert = atoi( rom );

    cout<<"beq\t"<<rsConvert<<"\t"<<rtConvert<<"\t"<<romConvert<<"\t";

    // DECODE INSTRUCTION INTO BINARY
    for( int x=0; x<5; x++ )
        output[x] = beq[x];

    convertDecToBin( rsConvert,  2, 7 );
    convertDecToBin( rtConvert,  2, 9 );
```

```
    result = true;
    convertDecToBin( romConvert, 7, 5 );
    result = false;

    for( int z=0; z<16; z++ )
        cout<<output[z];
    // cout<<endl;

    break;
}

case 21:
{
    // INSTRUCTION BNE
    // BNE RS, RT, ROM

    char *rs  = secondConversion( convert );
    char *rt  = secondConversion( convert );
    char *rom = thirdConversion ( convert );

    int rsConvert  = atoi( rs );
    int rtConvert  = atoi( rt );
    int romConvert = atoi( rom );

    cout<<"bne\t"<<rsConvert<<"\t"<<rtConvert<<"\t"<<romConvert<<"\t";

    // DECODE INSTRUCTION INTO BINARY
    for( int x=0; x<5; x++ )
        output[x] = bne[x];

    convertDecToBin( rsConvert,  2, 7 );
    convertDecToBin( rtConvert,  2, 9 );

    result = true;
    convertDecToBin( romConvert, 7, 5 );
    result = false;

    for( int z=0; z<16; z++ )
        cout<<output[z];
    // cout<<endl;

    break;
}

case 22:
{
    // INSTRUCTION J
    // J ROM

    char *rom = thirdConversion( convert );

    int romConvert = atoi( rom );

    cout<<"j  \t"<<romConvert<<"\t\t\t";

    // DECODE INSTRUCTION INTO BINARY
    for( int x=0; x<5; x++ )
        output[x] = j[x];

    result = true;
    convertDecToBin( romConvert, 7, 5 );
    result = false;

    for( int z=0; z<16; z++ )
        cout<<output[z];
    // cout<<endl;

    break;
}

case 23:
{
    // INSTRUCTION LWM
    // LWM RD, RAM

    char *rd  = secondConversion( convert );
    char *ram = thirdConversion ( convert );

    int rdConvert  = atoi( rd );
```

```
    int ramConvert = atoi( ram );

    cout<<"lwm\t"<<rdConvert<<"\t"<<ramConvert<<"\t\t";

    // DECODE INSTRUCTION INTO BINARY
    for( int x=0; x<5; x++ )
        output[x] = lwm[x];

    convertDecToBin( rdConvert,  2, 5 );
    convertDecToBin( ramConvert, 7, 9 );

    for( int z=0; z<16; z++ )
        cout<<output[z];
    // cout<<endl;

    break;
}

case 24:
{
    // INSTRUCTION SWM
    // SWM RS, RAM

    char *rs  = secondConversion( convert );
    char *ram = thirdConversion ( convert );

    int rsConvert  = atoi( rs );
    int ramConvert = atoi( ram );

    cout<<"swm\t"<<rsConvert<<"\t"<<ramConvert<<"\t\t";

    // DECODE INSTRUCTION INTO BINARY
    for( int x=0; x<5; x++ )
        output[x] = swm[x];

    convertDecToBin( rsConvert,  2, 7 );
    convertDecToBin( ramConvert, 7, 9 );

    for( int z=0; z<16; z++ )
        cout<<output[z];
    // cout<<endl;

    break;
}

case 25:
{
    // INSTRUCTION LWI
    // LWI RD, IMM

    char *rd  = secondConversion( convert );
    char *imm = thirdConversion ( convert );

    int rdConvert  = atoi( rd );
    int immConvert = atoi( imm );

    cout<<"lwi\t"<<rdConvert<<"\t"<<immConvert<<"\t\t";

    // DECODE INSTRUCTION INTO BINARY
    for( int x=0; x<5; x++ )
        output[x] = lwi[x];

    convertDecToBin( rdConvert,  2, 5 );
    convertDecToBin( immConvert, 4, 9 );

    for( int z=0; z<16; z++ )
        cout<<output[z];
    // cout<<endl;

    break;
}

case 26:
{
    // INSTRUCTION LWS
    // LWS RD

    char *rd = thirdConversion( convert );
```

```
            int rdConvert = atoi( rd );

            cout<<"lws\t"<<rdConvert<<"\t\t\t";

            // DECODE INSTRUCTION INTO BINARY
            for( int x=0; x<5; x++ )
                output[x] = lws[x];

            convertDecToBin( rdConvert, 2, 5 );

            for( int z=0; z<16; z++ )
                cout<<output[z];
            // cout<<endl;

            break;
        }

        case 27:
        {
            // INSTRUCTION SHOW
            // SHOW RS

            char *rs = thirdConversion( convert );

            int rsConvert = atoi( rs );

            cout<<"show\t"<<rsConvert<<"\t\t\t";

            // DECODE INSTRUCTION INTO BINARY
            for( int x=0; x<5; x++ )
                output[x] = show[x];

            convertDecToBin( rsConvert, 2, 7 );

            for( int z=0; z<16; z++ )
                cout<<output[z];
            // cout<<endl;

            break;
        }
    }
}

void convertDecToBin( int value, int limit, int index )
{
    int binaryMask;
    binaryMask = ( 1 << (limit-1) );

    count = 0;

    for( int c=1; c<=limit; c++ )
    {
        if( result )
        {
            if( count < 2 )
            {
                if( ( value & binaryMask ? '1' : '0' ) == 48 )
                    output[index] = 0;
                else
                    output[index] = 1;
                count++;
            }
            else
            {
                if( ( value & binaryMask ? '1' : '0' ) == 48 )
                    output[index+4] = 0;
                else
                    output[index+4] = 1;
            }
        }
        else
        {
            if( ( value & binaryMask ? '1' : '0' ) == 48 )
                output[index] = 0;
            else
                output[index] = 1;

        }
        index++;
```

```
        value <<= 1;
    }
}

// Convert Binary code to Hexadecimal value
void convertBinToHex( int start, int end )
{
    int v = 0;

    for( int i=start; i<=end; i++ )
    {
        if( output[i] == 1 )
        {
            if( i == start )
                v = v + 8;
            if( i == ( start + 1 ) )
                v = v + 4;
            if( i == ( start + 2 ) )
                v = v + 2;
            if( i == ( start + 3 ) )
                v = v + 1;
        }
    }

    if( v > 9 )
    {
        if( v == 10 )
            cout<<"A";
        if( v == 11 )
            cout<<"B";
        if( v == 12 )
            cout<<"C";
        if( v == 13 )
            cout<<"D";
        if( v == 14 )
            cout<<"E";
        if( v == 15 )
            cout<<"F";
    }
    else
    {
        cout<<v;
    }
}

//---------------------------------------------------------------------------

#ifndef mainWinH
#define mainWinH
//---------------------------------------------------------------------------
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include <Menus.hpp>
//---------------------------------------------------------------------------
#include "newWin.h"
#include <ComCtrls.hpp>
#include <Dialogs.hpp>
#include <ToolWin.hpp>
//---------------------------------------------------------------------------
class TMDIfrmMain : public TForm
{
__published:    // IDE-managed Components
   TMainMenu *mainMenu;
   TMenuItem *File1;
   TMenuItem *New1;
   TMenuItem *Open1;
   TMenuItem *Save1;
   TMenuItem *Close1;
   TMenuItem *N1;
   TMenuItem *Exit1;
   TMenuItem *N2;
   TMenuItem *Print1;
   TMenuItem *Help1;
   TMenuItem *About1;
   TStatusBar *statusBar;
   TOpenDialog *OpenDialog;
   TSaveDialog *SaveDialog;
```

```
   TMenuItem *View1;
   TMenuItem *InstructionSet1;
   TMenuItem *InstructionSetShow1;
   void __fastcall Exit1Click(TObject *Sender);
   void __fastcall New1Click(TObject *Sender);
   void __fastcall Open1Click(TObject *Sender);
   void __fastcall Save1Click(TObject *Sender);
   void __fastcall About1Click(TObject *Sender);
   void __fastcall Print1Click(TObject *Sender);
   void __fastcall InstructionSet1Click(TObject *Sender);
   void __fastcall InstructionSetShow1Click(TObject *Sender);
private:       // User declarations

   TMDICfrmNew *newSourceFile;
   TMDICfrmNew *openSourceFile;

public:     // User declarations
   __fastcall TMDIfrmMain(TComponent* Owner);
};
//---------------------------------------------------------------------------
extern PACKAGE TMDIfrmMain *MDIfrmMain;
//---------------------------------------------------------------------------
#endif

//---------------------------------------------------------------------------

#include <vcl.h>
#pragma hdrstop

#include "mainWin.h"
#include "aboutWin.h"
#include "instructionWin.h"

//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TMDIfrmMain *MDIfrmMain;
//---------------------------------------------------------------------------
__fastcall TMDIfrmMain::TMDIfrmMain(TComponent* Owner)
   : TForm(Owner)
{
   newSourceFile = NULL;
   openSourceFile = NULL;
}
//---------------------------------------------------------------------------
void __fastcall TMDIfrmMain::Exit1Click(TObject *Sender)
{
   // Cleanup if necessary
   if( newSourceFile )
      delete newSourceFile;
   if( openSourceFile )
      delete openSourceFile;

   Close( );   // Terminate Program
}
//---------------------------------------------------------------------------
void __fastcall TMDIfrmMain::New1Click(TObject *Sender)
{
   newSourceFile = new TMDICfrmNew(Application, true);
   newSourceFile->Caption = "New Source " + IntToStr(MDIChildCount + 1);
}
//---------------------------------------------------------------------------
void __fastcall TMDIfrmMain::Open1Click(TObject *Sender)
{
   if(OpenDialog->Execute())
   {
      openSourceFile = new TMDICfrmNew(Application, true);
      if(FileExists(OpenDialog->FileName))
      {
         openSourceFile->Caption = OpenDialog->FileName;
         openSourceFile->redtNewSource->Lines->LoadFromFile(OpenDialog->FileName);
      }
   }
}
//---------------------------------------------------------------------------
void __fastcall TMDIfrmMain::Save1Click(TObject *Sender)
{
   if(newSourceFile)
   {
```

```
      if(SaveDialog->Execute())
      {
         newSourceFile->redtNewSource->Lines->SaveToFile(SaveDialog->FileName);
         newSourceFile->Caption = SaveDialog->FileName;
      }
   }
}
//---------------------------------------------------------------------------
void __fastcall TMDIfrmMain::About1Click(TObject *Sender)
{
   AboutBox->ShowModal( );
}
//---------------------------------------------------------------------------

void __fastcall TMDIfrmMain::Print1Click(TObject *Sender)
{
   if( this->ActiveMDIChild == newSourceFile )
   {
      newSourceFile->redtNewSource->Print( newSourceFile->Caption );
   }
   if( this->ActiveMDIChild == openSourceFile )
   {
      openSourceFile->redtNewSource->Print( openSourceFile->Caption );
   }
}
//---------------------------------------------------------------------------



void __fastcall TMDIfrmMain::InstructionSet1Click(TObject *Sender)
{
   frmInstructions->Visible = false;
}
//---------------------------------------------------------------------------

void __fastcall TMDIfrmMain::InstructionSetShow1Click(TObject *Sender)
{
   frmInstructions->Visible = true ;
}
//---------------------------------------------------------------------------

//---------------------------------------------------------------------------

#ifndef newWinH
#define newWinH
//---------------------------------------------------------------------------
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include <ComCtrls.hpp>
#include <Menus.hpp>
//---------------------------------------------------------------------------
#include "asmWin.h"
#include <ActnList.hpp>
#include <StdActns.hpp>
//---------------------------------------------------------------------------
class TMDICfrmNew : public TForm
{
__published:    // IDE-managed Components
   TRichEdit *redtNewSource;
   TPopupMenu *popupMenu;
   TMenuItem *Cut1;
   TMenuItem *Copy1;
   TMenuItem *Paste1;
   TMenuItem *N1;
   TMenuItem *Assemble1;
   TActionList *ActionList1;
   TEditCopy *EditCopy1;
   TEditCut *EditCut1;
   TEditPaste *EditPaste1;
   TMenuItem *N2;
   TMenuItem *Print1;
   void __fastcall FormCreate(TObject *Sender);
   void __fastcall FormClose( TObject *Sender, TCloseAction &Action );
   void __fastcall redtNewSourceChange(TObject *Sender);
   void __fastcall Assemble1Click(TObject *Sender);
   void __fastcall Print1Click(TObject *Sender);
private:          // User declarations
```

```
   bool ACTIVATE;
   bool SOURCE_CHANGED;
   void __fastcall changeCaption( );

   TComponent *Master;

   TMDICfrmAssembler *assemblerView;

public:        // User declarations
   __fastcall TMDICfrmNew(TComponent* Owner);
   __fastcall TMDICfrmNew(TComponent* Owner, bool activate);
};
//---------------------------------------------------------------------------
extern PACKAGE TMDICfrmNew *MDICfrmNew;
//---------------------------------------------------------------------------
#endif

//---------------------------------------------------------------------------

#include <vcl.h>
#pragma hdrstop

//---------------------------------------------------------------------------
#include <iostream.h>
#include <strstrea.h>
#include <stdlib.h>
#include <stdio.h>
//---------------------------------------------------------------------------

#include "newWin.h"

//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TMDICfrmNew *MDICfrmNew;
//---------------------------------------------------------------------------
__fastcall TMDICfrmNew::TMDICfrmNew(TComponent* Owner)
   : TForm(Owner)
{
   Master = Owner;
   ACTIVATE = false;
   SOURCE_CHANGED = false;

   assemblerView = NULL;
}
//---------------------------------------------------------------------------
__fastcall TMDICfrmNew::TMDICfrmNew(TComponent* Owner, bool activate)
   : TForm(Owner)
{
   Master = Owner;
   ACTIVATE = activate;
   SOURCE_CHANGED = false;

   assemblerView = NULL;
}
//---------------------------------------------------------------------------
void __fastcall TMDICfrmNew::FormCreate(TObject *Sender)
{
   if( !ACTIVATE )
   {
      Close( );
   }
   else
   {
      this->redtNewSource->Width = 640;
      this->redtNewSource->Height = 300;
      this->AutoSize = true;
   }
}
//---------------------------------------------------------------------------
void __fastcall TMDICfrmNew::FormClose(TObject *Sender, TCloseAction &Action)
{
   if(assemblerView)
      delete assemblerView;

   Action = caFree;
}
void __fastcall TMDICfrmNew::redtNewSourceChange(TObject *Sender)
```

```
{
   if( !SOURCE_CHANGED )
   {
      changeCaption( );
   }
}
//---------------------------------------------------------------------------

void __fastcall TMDICfrmNew::changeCaption( )
{
   this->Caption = "[" + this->Caption + "]";
   SOURCE_CHANGED = true;
}

//---------------------------------------------------------------------------
//This is the function where the program will actually take the code and
//convert it to the binary and hex format.
//---------------------------------------------------------------------------
void __fastcall TMDICfrmNew::Assemble1Click(TObject *Sender)
{
   int NumberOfLines = 0;
   // int MaxLength = 0;

   if( !assemblerView )
   {
      assemblerView = new TMDICfrmAssembler( Master, true );
      assemblerView->Caption = "Assembler View {" + this->Caption + "}";
      NumberOfLines = redtNewSource->Lines->Count;
      if( NumberOfLines < 25 )
      {
         assemblerView->sgAssembler->RowCount = NumberOfLines + 2;
         assemblerView->resizeWindow( );
      }
      else
      {
         assemblerView->sgAssembler->RowCount = NumberOfLines + 1;
      }
   }

   TStrings *test = new TStringList;
   test = redtNewSource->Lines;

   for( int i=0; i<NumberOfLines; i++ )
   {
      // This is where we copy the instruction to the first column, we don't
      // neet it anymore, we will do it another way.
      assemblerView->sgAssembler->Cells[0][i+1] = i+1; // test->Strings[i];

      // Try to get it into a character
      char *convertString = new char[ (test->Strings[i].Length( )+10) ];
      strcpy( convertString, test->Strings[i].c_str( ) );

      // create a stream object
      ostrstream CODE;
      CODE<<convertString<<"\n"<<ends;
      char *codeConvert = CODE.str( );

      /* we do not need this anymore, why use it!
      if( MaxLength < strlen(codeConvert)*5 )
      {
         MaxLength = strlen(codeConvert)*5;
         assemblerView->sgAssembler->ColWidths[0] = MaxLength;
      }
      */

      // Send the string to the Assembler
      assemblerView->startAssembler( codeConvert, i );

      // Free memory
      delete [] convertString;
   }
}
//---------------------------------------------------------------------------



void __fastcall TMDICfrmNew::Print1Click(TObject *Sender)
{
   redtNewSource->Print( "" );
```

```
}
//---------------------------------------------------------------------------

//---------------------------------------------------------------------------
#ifndef aboutWinH
#define aboutWinH
//---------------------------------------------------------------------------
#include <vcl\System.hpp>
#include <vcl\Windows.hpp>
#include <vcl\SysUtils.hpp>
#include <vcl\Classes.hpp>
#include <vcl\Graphics.hpp>
#include <vcl\Forms.hpp>
#include <vcl\Controls.hpp>
#include <vcl\StdCtrls.hpp>
#include <vcl\Buttons.hpp>
#include <vcl\ExtCtrls.hpp>
//---------------------------------------------------------------------------
class TAboutBox : public TForm
{
__published:
    TPanel *Panel1;
    TImage *ProgramIcon;
    TLabel *ProductName;
    TLabel *Version;
    TLabel *Copyright;
    TLabel *Comments;
    TButton *OKButton;
private:
public:
    virtual __fastcall TAboutBox(TComponent* AOwner);
};
//---------------------------------------------------------------------------
extern PACKAGE TAboutBox *AboutBox;
//---------------------------------------------------------------------------
#endif

//---------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop

#include "aboutWin.h"
//---------------------------------------------------------------------
#pragma resource "*.dfm"
TAboutBox *AboutBox;
//---------------------------------------------------------------------
__fastcall TAboutBox::TAboutBox(TComponent* AOwner)
    : TForm(AOwner)
{
}
//---------------------------------------------------------------------

//---------------------------------------------------------------------------

#ifndef asmWinH
#define asmWinH
//---------------------------------------------------------------------------
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include <Grids.hpp>
//---------------------------------------------------------------------------
class TMDICfrmAssembler : public TForm
{
__published:    // IDE-managed Components
   TStringGrid *sgAssembler;
   void __fastcall FormCreate(TObject *Sender);
   void __fastcall FormClose( TObject *Sender, TCloseAction &Action );
private:          // User declarations

   bool ACTIVATE;
   void __fastcall createLabels( );

   // PROGRAM INSTRUCTION SET ------------
   static int add  [5];  // = { 0, 0, 0, 0, 0 };
   static int addi [5];  // = { 0, 0, 0, 0, 1 };
   static int addm [5];  // = { 0, 0, 0, 1, 0 };
   static int sub  [5];  // = { 0, 0, 0, 1, 1 };
```

```
   static int subi [5];  // = { 0, 0, 1, 0, 0 };
   static int subm [5];  // = { 0, 0, 1, 0, 1 };
   static int or   [5];  // = { 0, 0, 1, 1, 0 };
   static int ori  [5];  // = { 0, 0, 1, 1, 1 };
   static int orm  [5];  // = { 0, 1, 0, 0, 0 };
   static int and  [5];  // = { 0, 1, 0, 0, 1 };
   static int andi [5];  // = { 0, 1, 0, 1, 0 };
   static int andm [5];  // = { 0, 1, 0, 1, 1 };
   static int xor  [5];  // = { 0, 1, 1, 0, 0 };
   static int xori [5];  // = { 0, 1, 1, 0, 1 };
   static int xorm [5];  // = { 0, 1, 1, 1, 0 };
   static int not  [5];  // = { 0, 1, 1, 1, 1 };
   static int slt  [5];  // = { 1, 0, 0, 0, 0 };
   static int slti [5];  // = { 1, 0, 0, 0, 1 };
   static int sltm [5];  // = { 1, 0, 0, 1, 0 };
   static int beq  [5];  // = { 1, 0, 0, 1, 1 };
   static int bne  [5];  // = { 1, 0, 1, 0, 0 };
   static int j    [5];  // = { 1, 0, 1, 0, 1 };
   static int lwm  [5];  // = { 1, 0, 1, 1, 0 };
   static int swm  [5];  // = { 1, 0, 1, 1, 1 };
   static int lwi  [5];  // = { 1, 1, 0, 0, 0 };
   static int lws  [5];  // = { 1, 1, 0, 0, 1 };
   static int show [5];  // = { 1, 1, 0, 1, 0 };
   // PROGRAM INSTRUCTION SET ------------

   // ---------------------------------------------------------------------
   char* __fastcall firstConversion( char *convert );
   char* __fastcall secondConversion( char *convert );
   char* __fastcall thirdConversion( char *convert );
   // ---------------------------------------------------------------------
   void __fastcall decode( char *convert, int value );
   void __fastcall convertDecToBin( int value, int limit, int index );
   void __fastcall convertBinToHex( int start, int end );
   // ---------------------------------------------------------------------
   int  i;
   int  output[16];
   int  count;
   bool result;

public:        // User declarations
   __fastcall TMDICfrmAssembler(TComponent* Owner);
   __fastcall TMDICfrmAssembler(TComponent* Owner, bool activate);

   void __fastcall resizeWindow( );
   void __fastcall startAssembler( char* code, int r );
};
//---------------------------------------------------------------------------
extern PACKAGE TMDICfrmAssembler *MDICfrmAssembler;
//---------------------------------------------------------------------------
#endif

//---------------------------------------------------------------------------

#include <vcl.h>
#pragma hdrstop

//---------------------------------------------------------------------------
#include <iostream.h>
#include <strstrea.h>
//#include <string.h>
#include <stdlib.h>
#include <stdio.h>
//---------------------------------------------------------------------------

#include "asmWin.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TMDICfrmAssembler *MDICfrmAssembler;
//---------------------------------------------------------------------------
__fastcall TMDICfrmAssembler::TMDICfrmAssembler(TComponent* Owner)
   : TForm(Owner)
{
   ACTIVATE = false;
   i = 0;
   count = 0;
   result = false;
}
```

```
__fastcall TMDICfrmAssembler::TMDICfrmAssembler(TComponent* Owner, bool activate)
   : TForm(Owner)
{
   ACTIVATE = true;
   i = 0;
   count = 0;
   result = false;
}
//---------------------------------------------------------------------------
void __fastcall TMDICfrmAssembler::FormCreate(TObject *Sender)
{
   if( !ACTIVATE )
   {
      Close( );
   }
   else
   {
      sgAssembler->Width = sgAssembler->ColCount * sgAssembler->DefaultColWidth + 20;
      sgAssembler->Height = sgAssembler->RowCount * sgAssembler->DefaultRowHeight + 20;

      this->AutoSize = true;

      createLabels( );
   }
}
//---------------------------------------------------------------------------
void __fastcall TMDICfrmAssembler::resizeWindow( )
{
   // sgAssembler->Width = sgAssembler->ColCount * sgAssembler->DefaultColWidth + 20;
   // sgAssembler->Height = sgAssembler->RowCount * sgAssembler->DefaultRowHeight + 20;
   // this->AutoSize = true;

   sgAssembler->Width  = this->Width;
   sgAssembler->Height = this->Height;
   //this->AutoSize = true;
}
//---------------------------------------------------------------------------
void __fastcall TMDICfrmAssembler::createLabels( )
{
   sgAssembler->Cells[1][0] = "OP4";
   sgAssembler->Cells[2][0] = "OP3";
   sgAssembler->Cells[3][0] = "OP2";
   sgAssembler->Cells[4][0] = "OP1";
   sgAssembler->Cells[5][0] = "OP0";

   sgAssembler->Cells[6][0] = "RD1";
   sgAssembler->Cells[7][0] = "RD0";

   sgAssembler->Cells[8][0] = "RS1";
   sgAssembler->Cells[9][0] = "RS0";

   sgAssembler->Cells[10][0] = "RT1";
   sgAssembler->Cells[11][0] = "RT0";

   sgAssembler->Cells[12][0] = "IM1";
   sgAssembler->Cells[13][0] = "IM0";

   sgAssembler->Cells[14][0] = "AD2";
   sgAssembler->Cells[15][0] = "AD1";
   sgAssembler->Cells[16][0] = "AD0";
}

void __fastcall TMDICfrmAssembler::FormClose( TObject *Sender, TCloseAction &Action )
{
   Action = caFree;
}

//---------------------------------------------------------------------------
void __fastcall TMDICfrmAssembler::startAssembler( char* code, int r )
{

   if( code[0] == '\n' )
   {
      return; // nothing to do
   }
   else
   {
      for( int z=0; z<16; z++ )
         output[z] = 1;
```

```
char *opCode = firstConversion( code );

  if( strcmp( opCode, "add" ) == 0 )
      decode( code, 1 );

  if( strcmp( opCode, "addi" ) == 0 )
      decode( code, 2 );

  if( strcmp( opCode, "addm" ) == 0 )
      decode( code, 3 );

  if( strcmp( opCode, "sub" ) == 0 )
      decode( code, 4 );

  if( strcmp( opCode, "subi" ) == 0 )
      decode( code, 5 );

  if( strcmp( opCode, "subm" ) == 0 )
      decode( code, 6 );

  if( strcmp( opCode, "or" ) == 0 )
      decode( code, 7 );

  if( strcmp( opCode, "ori" ) == 0 )
      decode( code, 8 );

  if( strcmp( opCode, "orm" ) == 0 )
      decode( code, 9 );

  if( strcmp( opCode, "and" ) == 0 )
      decode( code, 10 );

  if( strcmp( opCode, "andi" ) == 0 )
      decode( code, 11 );

  if( strcmp( opCode, "andm" ) == 0 )
      decode( code, 12 );

  if( strcmp( opCode, "xor" ) == 0 )
      decode( code, 13 );

  if( strcmp( opCode, "xori" ) == 0 )
      decode( code, 14 );

  if( strcmp( opCode, "xorm" ) == 0 )
      decode( code, 15 );

  if( strcmp( opCode, "not" ) == 0 )
      decode( code, 16 );

  if( strcmp( opCode, "slt" ) == 0 )
      decode( code, 17 );

  if( strcmp( opCode, "slti" ) == 0 )
      decode( code, 18 );

  if( strcmp( opCode, "sltm" ) == 0 )
      decode( code, 19 );

  if( strcmp( opCode, "beq" ) == 0 )
      decode( code, 20 );

  if( strcmp( opCode, "bne" ) == 0 )
      decode( code, 21 );

  if( strcmp( opCode, "j" ) == 0 )
      decode( code, 22 );

  if( strcmp( opCode, "lwm" ) == 0 )
      decode( code, 23 );

  if( strcmp( opCode, "swm" ) == 0 )
      decode( code, 24 );

  if( strcmp( opCode, "lwi" ) == 0 )
      decode( code, 25 );

  if( strcmp( opCode, "lws" ) == 0 )
```

```
            decode( code, 26 );

        if( strcmp( opCode, "show" ) == 0 )
            decode( code, 27 );

      // Display the binary code!
      for( int x=0; x<16; x++ )
      {
         sgAssembler->Cells[x+1][r+1] = output[x];
      }
      i = 0;
   }
}
//---------------------------------------------------------------------------
char* __fastcall TMDICfrmAssembler::firstConversion( char *convert )
{
    ostrstream opcode;
    while( convert[i] != ' ' )
    {
        opcode<<convert[i];
        i++;
    }
    i++;
    opcode<<ends;
    char *opcodeChar = opcode.str( );

    return opcodeChar;
}
//---------------------------------------------------------------------------
char* __fastcall TMDICfrmAssembler::secondConversion( char *convert )
{
    ostrstream number;
    while( convert[i] != ' ' )
    {
        number<<convert[i];
        i++;
    }
    i++;
    number<<ends;
    char *numChar = number.str( );

    return numChar;
}
//---------------------------------------------------------------------------
char* __fastcall TMDICfrmAssembler::thirdConversion( char *convert )
{
    ostrstream number;
    while( convert[i] != '\n' )
    {
        number<<convert[i];
        i++;
    }
    number<<ends;
    char *numChar = number.str( );

    return numChar;
}
//---------------------------------------------------------------------------
void __fastcall TMDICfrmAssembler::decode( char *convert, int value )
{
    switch( value )
    {
        case 1:
        {
            // INSTRUCTION ADD
            // ADD RD, RS, RT

            char *rd = secondConversion( convert );
            char *rs = secondConversion( convert );
            char *rt = thirdConversion ( convert );

            int rdConvert = atoi( rd );
            int rsConvert = atoi( rs );
            int rtConvert = atoi( rt );

            cout<<"add\t"<<rdConvert<<"\t"<<rsConvert<<"\t"<<rtConvert<<"\t";

            // DECODE THE INSTRUCTION INTO BINARY
            for( int x=0; x<5; x++ )
```

```
        output[x] = add[x];

    convertDecToBin( rdConvert, 2, 5 );
    convertDecToBin( rsConvert, 2, 7 );
    convertDecToBin( rtConvert, 2, 9 );

    break;
}

case 2:
{
    // INSTRUCTION ADDI
    // ADDI RD, RS, IMM

    char *rd  = secondConversion( convert );
    char *rs  = secondConversion( convert );
    char *imm = thirdConversion ( convert );

    int rdConvert  = atoi( rd );
    int rsConvert  = atoi( rs );
    int immConvert = atoi( imm );

    cout<<"addi\t"<<rdConvert<<"\t"<<rsConvert<<"\t"<<immConvert<<"\t";

    // DECODE THE INSTRUCTION INTO BINARY
    for( int x=0; x<5; x++ )
        output[x] = addi[x];

    convertDecToBin( rdConvert,  2, 5 );
    convertDecToBin( rsConvert,  2, 7 );
    convertDecToBin( immConvert, 4, 9 );

    break;
}

case 3:
{
    // INSTRUCTION ADDM
    // ADDM RD, RS, RAM

    char *rd  = secondConversion( convert );
    char *rs  = secondConversion( convert );
    char *ram = thirdConversion ( convert );

    int rdConvert  = atoi( rd );
    int rsConvert  = atoi( rs );
    int ramConvert = atoi( ram );

    cout<<"addm\t"<<rdConvert<<"\t"<<rsConvert<<"\t"<<ramConvert<<"\t";

    // DECODE INSTRUCTION INTO BINARY
    for( int x=0; x<5; x++ )
        output[x] = addm[x];

    convertDecToBin( rdConvert,  2, 5 );
    convertDecToBin( rsConvert,  2, 7 );
    convertDecToBin( ramConvert, 7, 9 );

    break;
}

case 4:
{
    // INSTRUCTION SUB
    // SUB RD, RS, RT

    char *rd = secondConversion( convert );
    char *rs = secondConversion( convert );
    char *rt = thirdConversion ( convert );

    int rdConvert = atoi( rd );
    int rsConvert = atoi( rs );
    int rtConvert = atoi( rt );

    cout<<"sub\t"<<rdConvert<<"\t"<<rsConvert<<"\t"<<rtConvert<<"\t";

    // DECODE INSTRUCTION INTO BINARY
    for( int x=0; x<5; x++ )
        output[x] = sub[x];
```

```
    convertDecToBin( rdConvert, 2, 5 );
    convertDecToBin( rsConvert, 2, 7 );
    convertDecToBin( rtConvert, 2, 9 );

    break;
}

case 5:
{
    // INSTRUCTION SUBI
    // SUBI RD, RS, IMM

    char *rd  = secondConversion( convert );
    char *rs  = secondConversion( convert );
    char *imm = thirdConversion ( convert );

    int rdConvert  = atoi( rd );
    int rsConvert  = atoi( rs );
    int immConvert = atoi( imm );

    cout<<"subi\t"<<rdConvert<<"\t"<<rsConvert<<"\t"<<immConvert<<"\t";

    // DECODE INSTRUCTION INTO BINARY
    for( int x=0; x<5; x++ )
        output[x] = subi[x];

    convertDecToBin( rdConvert,  2, 5 );
    convertDecToBin( rsConvert,  2, 7 );
    convertDecToBin( immConvert, 4, 9 );

    break;
}

case 6:
{
    // INSTRUCTION SUBM
    // SUBM RD, RS, RAM

    char *rd  = secondConversion( convert );
    char *rs  = secondConversion( convert );
    char *ram = thirdConversion ( convert );

    int rdConvert  = atoi( rd );
    int rsConvert  = atoi( rs );
    int ramConvert = atoi( ram );

    cout<<"subm\t"<<rdConvert<<"\t"<<rsConvert<<"\t"<<ramConvert<<"\t";

    // DECODE INSTRUCTION INTO BINARY
    for( int x=0; x<5; x++ )
        output[x] = subm[x];

    convertDecToBin( rdConvert,  2, 5 );
    convertDecToBin( rsConvert,  2, 7 );
    convertDecToBin( ramConvert, 7, 9 );

    break;
}

case 7:
{
    // INSTRUCTION OR
    // OR RD, RS, RT

    char *rd = secondConversion( convert );
    char *rs = secondConversion( convert );
    char *rt = thirdConversion ( convert );

    int rdConvert = atoi( rd );
    int rsConvert = atoi( rs );
    int rtConvert = atoi( rt );

    cout<<"or\t"<<rdConvert<<"\t"<<rsConvert<<"\t"<<rtConvert<<"\t";

    // DECODE INSTRUCTION INTO BINARY
    for( int x=0; x<5; x++ )
        output[x] = or[x];
```

```
    convertDecToBin( rdConvert, 2, 5 );
    convertDecToBin( rsConvert, 2, 7 );
    convertDecToBin( rtConvert, 2, 9 );

    break;
}

case 8:
{
    // INSTRUCTION ORI
    // ORI RD, RS, IMM

    char *rd  = secondConversion( convert );
    char *rs  = secondConversion( convert );
    char *imm = thirdConversion ( convert );

    int rdConvert  = atoi( rd );
    int rsConvert  = atoi( rs );
    int immConvert = atoi( imm );

    cout<<"ori\t"<<rdConvert<<"\t"<<rsConvert<<"\t"<<immConvert<<"\t";

    // DECODE INSTRUCTION INTO BINARY
    for( int x=0; x<5; x++ )
        output[x] = ori[x];

    convertDecToBin( rdConvert,  2, 5 );
    convertDecToBin( rsConvert,  2, 7 );
    convertDecToBin( immConvert, 4, 9 );

    break;
}

case 9:
{
    // INSTRUCTION ORM
    // ORM RD, RS, RAM

    char *rd  = secondConversion( convert );
    char *rs  = secondConversion( convert );
    char *ram = thirdConversion ( convert );

    int rdConvert  = atoi( rd );
    int rsConvert  = atoi( rs );
    int ramConvert = atoi( ram );

    cout<<"orm\t"<<rdConvert<<"\t"<<rsConvert<<"\t"<<ramConvert<<"\t";

    // DECODE INSTRUCTION INTO BINARY
    for( int x=0; x<5; x++ )
        output[x] = orm[x];

    convertDecToBin( rdConvert,  2, 5 );
    convertDecToBin( rsConvert,  2, 7 );
    convertDecToBin( ramConvert, 7, 9 );

    break;
}

case 10:
{
    // INSTRUCTION AND
    // AND RD, RS, RT

    char *rd = secondConversion( convert );
    char *rs = secondConversion( convert );
    char *rt = thirdConversion ( convert );

    int rdConvert = atoi( rd );
    int rsConvert = atoi( rs );
    int rtConvert = atoi( rt );

    cout<<"and\t"<<rdConvert<<"\t"<<rsConvert<<"\t"<<rtConvert<<"\t";

    // DECODE INSTRUCTION INTO BINARY
    for( int x=0; x<5; x++ )
        output[x] = and[x];

    convertDecToBin( rdConvert, 2, 5 );
```

```
    convertDecToBin( rsConvert, 2, 7 );
    convertDecToBin( rtConvert, 2, 9 );

    break;
}

case 11:
{
    // INSTRUCTION ANDI
    // ANDI RD, RS, IMM

    char *rd  = secondConversion( convert );
    char *rs  = secondConversion( convert );
    char *imm = thirdConversion ( convert );

    int rdConvert  = atoi( rd );
    int rsConvert  = atoi( rs );
    int immConvert = atoi( imm );

    cout<<"andi\t"<<rdConvert<<"\t"<<rsConvert<<"\t"<<immConvert<<"\t";

    // DECODE INSTRUCTION INTO BINARY
    for( int x=0; x<5; x++ )
        output[x] = andi[x];

    convertDecToBin( rdConvert,  2, 5 );
    convertDecToBin( rsConvert,  2, 7 );
    convertDecToBin( immConvert, 4, 9 );

    break;
}

case 12:
{
    // INSTRUCTION ANDM
    // ANDM RD, RS, RAM

    char *rd  = secondConversion( convert );
    char *rs  = secondConversion( convert );
    char *ram = thirdConversion ( convert );

    int rdConvert  = atoi( rd );
    int rsConvert  = atoi( rs );
    int ramConvert = atoi( ram );

    cout<<"andm\t"<<rdConvert<<"\t"<<rsConvert<<"\t"<<ramConvert<<"\t";

    // DECODE INSTRUCTION INTO BINARY
    for( int x=0; x<5; x++ )
        output[x] = andm[x];

    convertDecToBin( rdConvert,  2, 5 );
    convertDecToBin( rsConvert,  2, 7 );
    convertDecToBin( ramConvert, 7, 9 );

    break;
}

case 13:
{
    // INSTRUCTION XOR
    // XOR RD, RS, RT

    char *rd = secondConversion( convert );
    char *rs = secondConversion( convert );
    char *rt = thirdConversion( convert );

    int rdConvert = atoi( rd );
    int rsConvert = atoi( rs );
    int rtConvert = atoi( rt );

    cout<<"xor\t"<<rdConvert<<"\t"<<rsConvert<<"\t"<<rtConvert<<"\t";

    // DECODE INSTRUCTION INTO BINARY
    for( int x=0; x<5; x++ )
        output[x] = xor[x];

    convertDecToBin( rdConvert, 2, 5 );
    convertDecToBin( rsConvert, 2, 7 );
```

```
    convertDecToBin( rtConvert, 2, 9 );

    break;
}

case 14:
{
    // INSTRUCTION XORI
    // XORI RD, RS, IMM

    char *rd  = secondConversion( convert );
    char *rs  = secondConversion( convert );
    char *imm = thirdConversion ( convert );

    int rdConvert  = atoi( rd );
    int rsConvert  = atoi( rs );
    int immConvert = atoi( imm );

    cout<<"xori\t"<<rdConvert<<"\t"<<rsConvert<<"\t"<<immConvert<<"\t";

    // DECODE INSTRUCTION INTO BINARY
    for( int x=0; x<5; x++ )
        output[x] = xori[x];

    convertDecToBin( rdConvert,  2, 5 );
    convertDecToBin( rsConvert,  2, 7 );
    convertDecToBin( immConvert, 4, 9 );

    break;
}

case 15:
{
    // INSTRUCTION XORM
    // XORM RD, RS, RAM

    char *rd  = secondConversion( convert );
    char *rs  = secondConversion( convert );
    char *ram = thirdConversion ( convert );

    int rdConvert  = atoi( rd );
    int rsConvert  = atoi( rs );
    int ramConvert = atoi( ram );

    cout<<"xorm\t"<<rdConvert<<"\t"<<rsConvert<<"\t"<<ramConvert<<"\t";

    // DECODE INSTRUCTION INTO BINARY
    for( int x=0; x<5; x++ )
        output[x] = xorm[x];

    convertDecToBin( rdConvert,  2, 5 );
    convertDecToBin( rsConvert,  2, 7 );
    convertDecToBin( ramConvert, 7, 9 );

    break;
}

case 16:
{
    // INSTRUCTION NOT
    // NOT RD, RS

    char *rd = secondConversion( convert );
    char *rs = thirdConversion ( convert );

    int rdConvert = atoi( rd );
    int rsConvert = atoi( rs );

    cout<<"not\t"<<rdConvert<<"\t"<<rsConvert<<"\t\t";

    // DECODE INSTRUCTION INTO BINARY
    for( int x=0; x<5; x++ )
        output[x] = not[x];

    convertDecToBin( rdConvert, 2, 5 );
    convertDecToBin( rsConvert, 2, 7 );

    break;
}
```

```
case 17:
{
    // INSTRUCTION SLT
    // SLT RD, RS, RT

    char *rd = secondConversion( convert );
    char *rs = secondConversion( convert );
    char *rt = thirdConversion ( convert );

    int rdConvert = atoi( rd );
    int rsConvert = atoi( rs );
    int rtConvert = atoi( rt );

    cout<<"slt\t"<<rdConvert<<"\t"<<rsConvert<<"\t"<<rtConvert<<"\t";

    // DECODE INSTRUCTION INTO BINARY
    for( int x=0; x<5; x++ )
        output[x] = slt[x];

    convertDecToBin( rdConvert, 2, 5 );
    convertDecToBin( rsConvert, 2, 7 );
    convertDecToBin( rtConvert, 2, 9 );

    break;
}

case 18:
{
    // INSTRUCTION SLTI
    // SLTI RD, RS, IMM

    char *rd  = secondConversion( convert );
    char *rs  = secondConversion( convert );
    char *imm = thirdConversion ( convert );

    int rdConvert  = atoi( rd );
    int rsConvert  = atoi( rs );
    int immConvert = atoi( imm );

    cout<<"slti\t"<<rdConvert<<"\t"<<rsConvert<<"\t"<<immConvert<<"\t";

    // DECODE INSTRUCTION INTO BINARY
    for( int x=0; x<5; x++ )
        output[x] = slti[x];

    convertDecToBin( rdConvert,  2, 5 );
    convertDecToBin( rsConvert,  2, 7 );
    convertDecToBin( immConvert, 4, 9 );

    break;
}

case 19:
{
    // INSTRUCTION SLTM
    // SLTM RD, RS, RAM

    char *rd  = secondConversion( convert );
    char *rs  = secondConversion( convert );
    char *ram = thirdConversion ( convert );

    int rdConvert  = atoi( rd );
    int rsConvert  = atoi( rs );
    int ramConvert = atoi( ram );

    cout<<"sltm\t"<<rdConvert<<"\t"<<rsConvert<<"\t"<<ramConvert<<"\t";

    // DECODE INSTRUCTION INTO BINARY
    for( int x=0; x<5; x++ )
        output[x] = sltm[x];

    convertDecToBin( rdConvert,  2, 5 );
    convertDecToBin( rsConvert,  2, 7 );
    convertDecToBin( ramConvert, 7, 9 );

    break;
}
```

```
case 20:
{
    // INSTRUCTION BEQ
    // BEQ RS, RT, ROM

    char *rs  = secondConversion( convert );
    char *rt  = secondConversion( convert );
    char *rom = thirdConversion ( convert );

    int rsConvert  = atoi( rs );
    int rtConvert  = atoi( rt );
    int romConvert = atoi( rom );

    cout<<"beq\t"<<rsConvert<<"\t"<<rtConvert<<"\t"<<romConvert<<"\t";

    // DECODE INSTRUCTION INTO BINARY
    for( int x=0; x<5; x++ )
        output[x] = beq[x];

    convertDecToBin( rsConvert,  2, 7 );
    convertDecToBin( rtConvert,  2, 9 );

    result = true;
    convertDecToBin( romConvert, 7, 5 );
    result = false;

    break;
}

case 21:
{
    // INSTRUCTION BNE
    // BNE RS, RT, ROM

    char *rs  = secondConversion( convert );
    char *rt  = secondConversion( convert );
    char *rom = thirdConversion ( convert );

    int rsConvert  = atoi( rs );
    int rtConvert  = atoi( rt );
    int romConvert = atoi( rom );

    cout<<"bne\t"<<rsConvert<<"\t"<<rtConvert<<"\t"<<romConvert<<"\t";

    // DECODE INSTRUCTION INTO BINARY
    for( int x=0; x<5; x++ )
        output[x] = bne[x];

    convertDecToBin( rsConvert,  2, 7 );
    convertDecToBin( rtConvert,  2, 9 );

    result = true;
    convertDecToBin( romConvert, 7, 5 );
    result = false;

    break;
}

case 22:
{
    // INSTRUCTION J
    // J ROM

    char *rom = thirdConversion( convert );

    int romConvert = atoi( rom );

    cout<<"j  \t"<<romConvert<<"\t\t\t";

    // DECODE INSTRUCTION INTO BINARY
    for( int x=0; x<5; x++ )
        output[x] = j[x];

    result = true;
    convertDecToBin( romConvert, 7, 5 );
    result = false;

    break;
}
```

```
case 23:
{
    // INSTRUCTION LWM
    // LWM RD, RAM

    char *rd  = secondConversion( convert );
    char *ram = thirdConversion ( convert );

    int rdConvert  = atoi( rd );
    int ramConvert = atoi( ram );

    cout<<"lwm\t"<<rdConvert<<"\t"<<ramConvert<<"\t\t";

    // DECODE INSTRUCTION INTO BINARY
    for( int x=0; x<5; x++ )
        output[x] = lwm[x];

    convertDecToBin( rdConvert,  2, 5 );
    convertDecToBin( ramConvert, 7, 9 );

    break;
}

case 24:
{
    // INSTRUCTION SWM
    // SWM RS, RAM

    char *rs  = secondConversion( convert );
    char *ram = thirdConversion ( convert );

    int rsConvert  = atoi( rs );
    int ramConvert = atoi( ram );

    cout<<"swm\t"<<rsConvert<<"\t"<<ramConvert<<"\t\t";

    // DECODE INSTRUCTION INTO BINARY
    for( int x=0; x<5; x++ )
        output[x] = swm[x];

    convertDecToBin( rsConvert,  2, 7 );
    convertDecToBin( ramConvert, 7, 9 );

    break;
}

case 25:
{
    // INSTRUCTION LWI
    // LWI RD, IMM

    char *rd  = secondConversion( convert );
    char *imm = thirdConversion ( convert );

    int rdConvert  = atoi( rd );
    int immConvert = atoi( imm );

    cout<<"lwi\t"<<rdConvert<<"\t"<<immConvert<<"\t\t";

    // DECODE INSTRUCTION INTO BINARY
    for( int x=0; x<5; x++ )
        output[x] = lwi[x];

    convertDecToBin( rdConvert,  2, 5 );
    convertDecToBin( immConvert, 4, 9 );

    break;
}

case 26:
{
    // INSTRUCTION LWS
    // LWS RD

    char *rd = thirdConversion( convert );

    int rdConvert = atoi( rd );
```

```
            cout<<"lws\t"<<rdConvert<<"\t\t\t";

            // DECODE INSTRUCTION INTO BINARY
            for( int x=0; x<5; x++ )
                output[x] = lws[x];

            convertDecToBin( rdConvert, 2, 5 );

            break;
        }

        case 27:
        {
            // INSTRUCTION SHOW
            // SHOW RS

            char *rs = thirdConversion( convert );

            int rsConvert = atoi( rs );

            cout<<"show\t"<<rsConvert<<"\t\t\t";

            // DECODE INSTRUCTION INTO BINARY
            for( int x=0; x<5; x++ )
                output[x] = show[x];

            convertDecToBin( rsConvert, 2, 7 );

            break;
        }
    }
}

void __fastcall TMDICfrmAssembler::convertDecToBin( int value, int limit, int index )
{
    int binaryMask;
    binaryMask = ( 1 << (limit-1) );

    count = 0;

    for( int c=1; c<=limit; c++ )
    {
        if( result )
        {
            if( count < 2 )
            {
                if( ( value & binaryMask ? '1' : '0' ) == 48 )
                    output[index] = 0;
                else
                    output[index] = 1;
                count++;
            }
            else
            {
                if( ( value & binaryMask ? '1' : '0' ) == 48 )
                    output[index+4] = 0;
                else
                    output[index+4] = 1;
            }
        }
        else
        {
            if( ( value & binaryMask ? '1' : '0' ) == 48 )
                output[index] = 0;
            else
                output[index] = 1;

        }
        index++;
        value <<= 1;
    }
}

// Convert Binary code to Hexadecimal value
void __fastcall TMDICfrmAssembler::convertBinToHex( int start, int end )
{
    int v = 0;

    for( int i=start; i<=end; i++ )
```

```
    {
        if( output[i] == 1 )
        {
            if( i == start )
                v = v + 8;
            if( i == ( start + 1 ) )
                v = v + 4;
            if( i == ( start + 2 ) )
                v = v + 2;
            if( i == ( start + 3 ) )
                v = v + 1;
        }
    }

    if( v > 9 )
    {
        if( v == 10 )
            cout<<"A";
        if( v == 11 )
            cout<<"B";
        if( v == 12 )
            cout<<"C";
        if( v == 13 )
            cout<<"D";
        if( v == 14 )
            cout<<"E";
        if( v == 15 )
            cout<<"F";
    }
    else
    {
        cout<<v;
    }
}


// PROGRAM INSTRUCTION SET ------------
int TMDICfrmAssembler::add  [5] = { 0, 0, 0, 0, 0 };
int TMDICfrmAssembler::addi [5] = { 0, 0, 0, 0, 1 };
int TMDICfrmAssembler::addm [5] = { 0, 0, 0, 1, 0 };
int TMDICfrmAssembler::sub  [5] = { 0, 0, 0, 1, 1 };
int TMDICfrmAssembler::subi [5] = { 0, 0, 1, 0, 0 };
int TMDICfrmAssembler::subm [5] = { 0, 0, 1, 0, 1 };
int TMDICfrmAssembler::or   [5] = { 0, 0, 1, 1, 0 };
int TMDICfrmAssembler::ori  [5] = { 0, 0, 1, 1, 1 };
int TMDICfrmAssembler::orm  [5] = { 0, 1, 0, 0, 0 };
int TMDICfrmAssembler::and  [5] = { 0, 1, 0, 0, 1 };
int TMDICfrmAssembler::andi [5] = { 0, 1, 0, 1, 0 };
int TMDICfrmAssembler::andm [5] = { 0, 1, 0, 1, 1 };
int TMDICfrmAssembler::xor  [5] = { 0, 1, 1, 0, 0 };
int TMDICfrmAssembler::xori [5] = { 0, 1, 1, 0, 1 };
int TMDICfrmAssembler::xorm [5] = { 0, 1, 1, 1, 0 };
int TMDICfrmAssembler::not  [5] = { 0, 1, 1, 1, 1 };
int TMDICfrmAssembler::slt  [5] = { 1, 0, 0, 0, 0 };
int TMDICfrmAssembler::slti [5] = { 1, 0, 0, 0, 1 };
int TMDICfrmAssembler::sltm [5] = { 1, 0, 0, 1, 0 };
int TMDICfrmAssembler::beq  [5] = { 1, 0, 0, 1, 1 };
int TMDICfrmAssembler::bne  [5] = { 1, 0, 1, 0, 0 };
int TMDICfrmAssembler::j    [5] = { 1, 0, 1, 0, 1 };
int TMDICfrmAssembler::lwm  [5] = { 1, 0, 1, 1, 0 };
int TMDICfrmAssembler::swm  [5] = { 1, 0, 1, 1, 1 };
int TMDICfrmAssembler::lwi  [5] = { 1, 1, 0, 0, 0 };
int TMDICfrmAssembler::lws  [5] = { 1, 1, 0, 0, 1 };
int TMDICfrmAssembler::show [5] = { 1, 1, 0, 1, 0 };
// PROGRAM INSTRUCTION SET ------------

//---------------------------------------------------------------------------

#ifndef instructionWinH
#define instructionWinH
//---------------------------------------------------------------------------
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include <ComCtrls.hpp>
#include <Menus.hpp>
#include <ImgList.hpp>
#include <ActnList.hpp>
```

```
#include <StdActns.hpp>
//---------------------------------------------------------------------------
class TfrmInstructions : public TForm
{
__published:    // IDE-managed Components
   TTreeView *TreeView;
   TRichEdit *redtInstructions;
   TPopupMenu *PopupMenu1;
   TMenuItem *Whatisthis1;
   TImageList *ImageList1;
   TMenuItem *ShowButton1;
   TMenuItem *ShowLines1;
   TMenuItem *N1;
   TMenuItem *HideButton1;
   TMenuItem *HideLines1;
   TMenuItem *ExpandTree1;
   TMenuItem *CollapseTree1;
   TPopupMenu *pmnuDisplay;
   TMenuItem *Copy1;
   TMenuItem *print1;
   TActionList *ActionList1;
   TEditCopy *EditCopy1;
   void __fastcall ShowButton1Click(TObject *Sender);
   void __fastcall ShowLines1Click(TObject *Sender);
   void __fastcall HideButton1Click(TObject *Sender);
   void __fastcall HideLines1Click(TObject *Sender);

   void __fastcall TreeViewMouseDown(TObject *Sender, TMouseButton Button,
                        TShiftState Shift, int X, int Y);
   void __fastcall ExpandTree1Click(TObject *Sender);
   void __fastcall CollapseTree1Click(TObject *Sender);
   void __fastcall print1Click(TObject *Sender);

private:    // User declarations


public:     // User declarations
   __fastcall TfrmInstructions(TComponent* Owner);
};
//---------------------------------------------------------------------------
extern PACKAGE TfrmInstructions *frmInstructions;
//---------------------------------------------------------------------------
#endif

//---------------------------------------------------------------------------

#include <vcl.h>
#pragma hdrstop

#include "instructionWin.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TfrmInstructions *frmInstructions;
//---------------------------------------------------------------------------
__fastcall TfrmInstructions::TfrmInstructions(TComponent* Owner)
   : TForm(Owner)
{

}
//---------------------------------------------------------------------------

void __fastcall TfrmInstructions::ShowButton1Click(TObject *Sender)
{
   TreeView->ShowButtons = true;
}
//---------------------------------------------------------------------------

void __fastcall TfrmInstructions::ShowLines1Click(TObject *Sender)
{
   TreeView->ShowLines = true;
}
//---------------------------------------------------------------------------

void __fastcall TfrmInstructions::HideButton1Click(TObject *Sender)
{
   TreeView->ShowButtons = false;
}
//---------------------------------------------------------------------------
```

```
void __fastcall TfrmInstructions::HideLines1Click(TObject *Sender)
{
   TreeView->ShowLines = false;
}
//---------------------------------------------------------------------------

void __fastcall TfrmInstructions::TreeViewMouseDown(TObject *Sender,
      TMouseButton Button, TShiftState Shift, int X, int Y)
{
   THitTests HT;

   if (Sender->ClassNameIs("TTreeView"))
   {
      TTreeView *pTV = (TTreeView *)Sender;
      HT = pTV->GetHitTestInfoAt(X,Y);
      if (HT.Contains(htOnItem))

      // pTV->Items->Delete(pTV->GetNodeAt(X,Y));

      /*
      if ( TreeView->Selected->ImageIndex == 0 )
         TreeView->Selected->ImageIndex = 1 ;
      else
         TreeView->Selected->ImageIndex = 0;
      */

      // Add Instructions
      if( TreeView->Selected->Text == "add" )
      {
         redtInstructions->Clear( );
         redtInstructions->Lines->LoadFromFile( "add.rtf" );
      }
      if( TreeView->Selected->Text == "addi" )
      {
         redtInstructions->Clear( );
         redtInstructions->Lines->LoadFromFile( "addi.rtf" );
      }
      if( TreeView->Selected->Text == "addm" )
      {
         redtInstructions->Clear( );
         redtInstructions->Lines->LoadFromFile( "addm.rtf" );
      }

      // Sub Instruction
      if( TreeView->Selected->Text == "sub" )
      {
         redtInstructions->Clear( );
         redtInstructions->Lines->LoadFromFile( "sub.rtf" );
      }
      if( TreeView->Selected->Text == "subi" )
      {
         redtInstructions->Clear( );
         redtInstructions->Lines->LoadFromFile( "subi.rtf" );
      }
      if( TreeView->Selected->Text == "subm" )
      {
         redtInstructions->Clear( );
         redtInstructions->Lines->LoadFromFile( "subm.rtf" );
      }

      // Or Instruction
      if( TreeView->Selected->Text == "or" )
      {
         redtInstructions->Clear( );
         redtInstructions->Lines->LoadFromFile( "or.rtf" );
      }
      if( TreeView->Selected->Text == "ori" )
      {
         redtInstructions->Clear( );
         redtInstructions->Lines->LoadFromFile( "ori.rtf" );
      }
      if( TreeView->Selected->Text == "orm" )
      {
         redtInstructions->Clear( );
         redtInstructions->Lines->LoadFromFile( "orm.rtf" );
      }

      // And Instruction
```

```
if( TreeView->Selected->Text == "and" )
{
   redtInstructions->Clear( );
   redtInstructions->Lines->LoadFromFile( "and.rtf" );
}
if( TreeView->Selected->Text == "andi" )
{
   redtInstructions->Clear( );
   redtInstructions->Lines->LoadFromFile( "andi.rtf" );
}
if( TreeView->Selected->Text == "andm" )
{
   redtInstructions->Clear( );
   redtInstructions->Lines->LoadFromFile( "andm.rtf" );
}

// Xor Instruction
if( TreeView->Selected->Text == "xor" )
{
   redtInstructions->Clear( );
   redtInstructions->Lines->LoadFromFile( "xor.rtf" );
}
if( TreeView->Selected->Text == "xori" )
{
   redtInstructions->Clear( );
   redtInstructions->Lines->LoadFromFile( "xori.rtf" );
}
if( TreeView->Selected->Text == "xorm" )
{
   redtInstructions->Clear( );
   redtInstructions->Lines->LoadFromFile( "xorm.rtf" );
}

// Not Instruction
if( TreeView->Selected->Text == "not" )
{
   redtInstructions->Clear( );
   redtInstructions->Lines->LoadFromFile( "not.rtf" );
}

// Slt Instruction
      if( TreeView->Selected->Text == "slt" )
{
   redtInstructions->Clear( );
   redtInstructions->Lines->LoadFromFile( "slt.rtf" );
}
if( TreeView->Selected->Text == "slti" )
{
   redtInstructions->Clear( );
   redtInstructions->Lines->LoadFromFile( "slti.rtf" );
}
if( TreeView->Selected->Text == "sltm" )
{
   redtInstructions->Clear( );
   redtInstructions->Lines->LoadFromFile( "sltm.rtf" );
}

// Beq Instruction & Bne Instruction
if( TreeView->Selected->Text == "beq" )
{
   redtInstructions->Clear( );
   redtInstructions->Lines->LoadFromFile( "beq.rtf" );
}
if( TreeView->Selected->Text == "bne" )
{
   redtInstructions->Clear( );
   redtInstructions->Lines->LoadFromFile( "bne.rtf" );
}

// Jump Instruction
if( TreeView->Selected->Text == "j" )
{
   redtInstructions->Clear( );
   redtInstructions->Lines->LoadFromFile( "j.rtf" );
}

// Show Instruction
if( TreeView->Selected->Text == "show" )
{
```

```
         redtInstructions->Clear( );
         redtInstructions->Lines->LoadFromFile( "show.rtf" );
      }

      // Lwm, Swm, Lwi, and Lws Instruction
      if( TreeView->Selected->Text == "lwm" )
      {
         redtInstructions->Clear( );
         redtInstructions->Lines->LoadFromFile( "lwm.rtf" );
      }
      if( TreeView->Selected->Text == "swm" )
      {
         redtInstructions->Clear( );
         redtInstructions->Lines->LoadFromFile( "swm.rtf" );
      }
      if( TreeView->Selected->Text == "lwi" )
      {
         redtInstructions->Clear( );
         redtInstructions->Lines->LoadFromFile( "lwi.rtf" );
      }
      if( TreeView->Selected->Text == "lws" )
      {
         redtInstructions->Clear( );
         redtInstructions->Lines->LoadFromFile( "lws.rtf" );
      }

   }
}

void __fastcall TfrmInstructions::ExpandTree1Click(TObject *Sender)
{
   TreeView->FullExpand( );
}
//---------------------------------------------------------------------------

void __fastcall TfrmInstructions::CollapseTree1Click(TObject *Sender)
{
   TreeView->FullCollapse( );
}
//---------------------------------------------------------------------------

void __fastcall TfrmInstructions::print1Click(TObject *Sender)
{
   redtInstructions->Print( "JARC Instructions Help Document" );
}
//---------------------------------------------------------------------------
```