```
HCOPY  000000001077
T0000001D17202D69202D4B10103603202629000033200074B10105D3F2FEC032010
T00001D130F20160100030F200D4B10105D3E2003454F46
T0000301DB410B400B44075101000E32019332FFAE32013A00433200857C003B850
T0010531D3B2FEA1340004F0000F1B410774000E32011332FFA53C003DF2008B850
T00107073B2FEF4F000005
M00000705
M00001405
M00002705
E000000
```

```
HCOPY  000000000CEA
T0000001D17202D69202D4B100CA703202629000033200074B100CCF3F2FEC032010
T00001D130F20160100030F200D4B100CCF3E200348484B
T000CA71DB410B400B440751003E6E32019332FFAE32013A00433200857C003B850
T000CC41E3B2FEA1340004F0000F101B410774000E32011332FFA53C003DF2008B850
T000CE273B2FEF4F0000EE5
M00000705
M00001405
M00002705
E000000
```

Assembler Pass2

Group #9

Names:

(11) Ahmed Mahmoud Sallam

(21) Hassan Khalil

(27) Shady Abdel Aziz

(41) Omar Khaled

(64) Mahmoud Abdel Latif.

Requirements Specification

1. The pass1 is to execute by entering pass1 <source-file-name>
2. The source file for the main program for this phase is to be named pass1.c
3. After getting a 2D array from Pass1. First dimension = number of input lines, second dimension = 5. First columns represent the address, second column represent the label, third column represents the Mnemonic, fourth column represents the operand, fifth column represents the comment.
4. Loop over the 2D array , if the word in column 3 is found in the byte table:
    a. If it is format 2, then get the object code of the input register, check if this register is contained in the available registers table. If there are 2 registers, check if the size of input operand = 3. Then get the object code of the 2 input registers from the registers table.
    b. If it is format 3
        I. If it is immediate address, check if it a number, if it is a number, put it in the object code directly, else check the lapel in the symbol table if it is found. In case it is found in the symbol table, try to use the program counter to get the displacement. If the displacement doesn't fit using program counter, check if the input program defined a base before or not. If there is a base, try using the base relative. If it doesn't fit also, indicate an error. Put I = 1, n = 0 as the input addressing mode is immediate. Then put pc = 1 or b = 1, e = 0.The displacement should fit in 12 bits.

II. If it is indirect address, check if it a number, if it is a number, put it in the object code directly, else check the lapel in the symbol table if it is found. In case it is found in the symbol table, try to use the program counter to get the displacement. If the displacement doesn't fit using program counter, check if the input program defined a base before or not. If there is a base, try using the base relative. If it doesn't fit also, indicate an error. Put I = 0, n = 1 as the input addressing mode is indirect. Then put pc = 1 or b = 1, e = 0.

III. If it is direct address, check if it a number, check the lapel in the symbol table if it is found. In case it is found in the symbol table, try to use the program counter to get the displacement. If the displacement doesn't fit using program counter, check if the input program defined a base before or not. If there is a base, try using the base relative. If it doesn't fit also, indicate an error. Put I = 1, n = 1 (in SIC/XE) as the input addressing mode is direct. Then put pc = 1 or b = 1, e = 0.

C. If it is format 4:

i. Put e = 1, n = 1, I = 1, b = 0, pc = 0 as no program counter nor base relative will be used as the whole address will be put instead of the displacement, but there will be a modification record to avoid the error if the loader loads the program in another position other than the specified start address in the given input program.

5. If the word in column three is directive symbol:
    i. *EQU directive*: (SYMBOL EQU VALUE)
    This directive takes the symbol on the left hand side and checks if it is present in the symbol table , and if not it inserts it in the symbol table with the value on the right hand side.(If the right hand side contains symbols it must have been defined previously)
    ii. *ORG directive* : (ORG VALUE)
    This directive sets the location counter to the value specified on the right hand side.(If the value is a symbol it must have defined previously)
    iii. *Literals* :
    Literals are made using the literal table which contains a map of literals (key is the name of literal and value is Literal data which contains the literal length, address and length) and when the '=' sign is found in the operand the operand is validated for three formats (=C'ASCII TEXT', =X'HEX TEXT',=DECIMAL) then it is searched for in the literal table and if it is not found then it will be added in the literal table.(After END statement the literals are placed and given addresses)
    iv. LTORG directive:
    This directive allows to place the literals in a specified place to be close from the area where the literals are referenced. Line class : this class has a method that processes a line that was read from the file resulting in Pass 1 and sets the values of the line (Address,Label,Mnemonic, Operand) and if a value is not present it is set to empty string.

6) After generating each object code; check if it is a text record , if yes, check if it will fit in the current text record.
If it won't, write the current text record to the file, and Create a new one.
If it is a header record, write the program's name and the Starting address.
If it is an End record, write it in the file, paired with the Length of the program.

1.Design

Our project consists of 5 classes :

1- class holds opCode map with functions :
get() excist(), getMap()

2- class holds symbol table with functions :
get() excist(), getMap()

3- class holds operation byte map with functions :
get() excist(), getMap()

4- class Parser parses the input file and fill the 2D array.

5-class Engine that performs Pass1 algorithm and works on the 2D array sent from parser then sets addresses of every line or set errors.

6) Class ObjectCode Generateor which generate the object code for each given input line.

7) Class Literal table which contains all the input literal inputs and their corresponding values.

Data Structure

We used two data structure:

- Map: Store the OPTAB and the SYMTAB
- 2d array of strings: Store the data of the output file such as address, mnemonics and comments...
- Map for the literals and their corresponding values.

Assumptions:

Comments must start with (.)..

Any word after (RSUB) is treated as comment ..

Algorithms used

If the line in format 2:

```
if(type.compare("2") == 0){

        int registersNum = getRegistersNum(menomonic);

        if(registersNum == 1){

          if(operand.size() != 1){

            printf("error in format2, expected 1 register but size != 1");
            continue;
```

```
    } else {

        //string operand0 = "";
        //operand0.push_back(operand[0]);
        string registerCode = regsisterMap[ operand/*0*/];
        if(registerCode.compare("") == 0){

            printf("wrong register name!");
            continue;
        }

        object_code[i] = menomonicCode + registerCode + "0";
              cout << object_code[i] << "\n";

        continue;
    }

} else {

    if(operand.size() != 3){

        printf("error in format2, expected 2 register but size != 3");
        continue;

    } else {

        string operand1 = "";
        string operand2 = "";
        operand1.push_back(operand[0]);
        operand2.push_back(operand[2]);
        string registerCode1 = regsisterMap[operand1];
        string registerCode2 = regsisterMap[operand2];
        //cout<< operand1;
        //cout<< operand2;

        if(registerCode1.compare("") == 0 || registerCode2.compare("") == 0){

            printf("wrong register name!");
            continue;
        }

        if(operand[1] != ','){

            printf("expected , in format 2");
```

```
            continue;
        }

        object_code[i] = menomonicCode + registerCode1 + registerCode2;
            cout << object_code[i] << "\n";

    }
}
```

If the line in Format 3:

```
else if(type.compare("3") == 0){

    if(operand[0] == '#' || operand[0] == '@'){

        bool isNumber = true;
        int len  = operand.size();
        for(int j = 1; j < len && isNumber; j++){

            if(operand[j] < '0' || operand[j] > '9')
                isNumber = false;
        }

        if(isNumber){

            int immediateNumber = strToDev(operand.substr(1));
            address = "";
            base_conv(immediateNumber , 16);
            if(address.size() > 3){

                printf("Immediate number is larger than 3 half bytes");
                continue;

            } else if (address.size() < 3){

                while(address.size() != 3) address = "0" + address;

            }

            string hexaOpernadNum = address;
            int tempNum = toDecimal(menomonicCode) + 1;
            if(operand[0] == '@'){
                tempNum++;
            }
            address = "";
```

```
                base_conv(tempNum , 16);
                while(address.size() != 2) address = "0" + address;
                menomonicCode = address;
                object_code[i] = menomonicCode + "0" + hexaOpernadNum;
                    cout << object_code[i] << "\n";

                continue;

            } else {
                string operand_label = operand.substr(1);
                string operand_address = symTable.get(operand_label);
                if(operand_address == ""){
                cout <<  "error label name " << operand_label;
                continue;
                }
                if(operand[0] == '@'){
                if(!is_relative_address(operand_address, src[i+1][0], base_used, base_address, i,
object_code, menomonicCode, 2, "2" , "4")){
                cout <<  "error no pc,base,number format 3 # / @" ;
                }
                }
                else{
                if(!is_relative_address(operand_address, src[i+1][0], base_used, base_address, i,
object_code, menomonicCode, 1, "2" , "4")){
                cout <<  "error no pc,base,number format 3 # / @" ;
                }
                }

                continue;
            }
        }
        if(is_exist(operand,',')){
            string data[2];
            split(data, operand, ',');
            if(data[1] != "X"){
                cout <<  "error no x , X" ;
                continue;
            }
            string operand_address = symTable.get(data[0]);
            if(operand_address != ""){
                if(!is_relative_address(operand_address, src[i+1][0], base_used, base_address, i,
object_code, menomonicCode, 3, "A" , "C")){
                cout <<  "error no pc,base,number format 3  (X)" ;
                }
```

```
                continue;
            }
            else if( (isHex(data[0]) == true) && (data[0].length() <= 3) ){
                while(data[0].size() != 3) data[0] = "0" + data[0];
                int tempNum = toDecimal(menomonicCode) + 3;
                address = "";
                base_conv(tempNum , 16);
                while(address.size() != 2) address = "0" + address;
                menomonicCode = address;
                object_code[i] = menomonicCode + "8" + data[0];
                    cout << object_code[i] << "\n";

                continue;
            }
            else{
                cout <<  "error no address valid format 3  (X)" ;
                continue;
            }
        }
        if(isHex(operand) && operand.length() <= 3){
            while(operand.size() != 3) operand = "0" + operand;
            int tempNum = toDecimal(menomonicCode) + 3;
            address = "";
            base_conv(tempNum , 16);
            while(address.size() != 2) address = "0" + address;
            menomonicCode = address;
            object_code[i] = menomonicCode + "0" + operand;
                cout << object_code[i] << "\n";

            continue;
        }
        else{
            string operand_address = symTable.get(operand);
            if(operand_address == ""){
                operand_address = lit_table.get(operand);
            }
            if(operand_address != ""){
                if(!is_relative_address(operand_address, src[i+1][0], base_used, base_address, i,
object_code, menomonicCode, 3, "2" , "4")){
                    cout <<  "error no pc,base,number format 3  no ay 7aga " ;
                }
                continue;
            }
            else{
```

```
            cout <<  "error label does not exist " ;
            continue;
          }
        }
      }
```

If the line in format 4:

```
string operand_address = symTable.get(operand);
        if(operand_address != ""){
          int tempNum = toDecimal(menomonicCode) + 3;
          address = "";
          base_conv(tempNum , 16);
          while(address.size() != 2) address = "0" + address;
          while(operand_address.size() != 5) operand_address = "0" + operand_address;
          menomonicCode = address;
          object_code[i] = menomonicCode + "1" + operand_address;
              cout << object_code[i] << "\n";
          int len = 5 - (src[i][0].length());
          string m_record = "M";
          for(int y = 0 ; y < len+1 ; y++){
            m_record = m_record + "0";
          }
          int val = toDecimal(src[i][0]) + 1;
          address = "";
          base_conv(val, 16);
          m_record = m_record + address;
          m_record = m_record + "05";
          cout<<m_record;
          modification_record.push_back(m_record);
          continue;
        }
        else{
          if(is_exist(operand,',')){
            string data[2];
            split(data, operand, ',');
            if(data[1] != "X"){
              cout <<  "error no x , X" ;
              continue;
            }
            string operand_address = symTable.get(data[0]);
            if(operand_address != ""){
              int tempNum = toDecimal(menomonicCode) + 3;
```

```
        address = "";
        base_conv(tempNum , 16);
        while(address.size() != 2) address = "0" + address;
        menomonicCode = address;
        object_code[i] = menomonicCode + "9" + operand_address;
              cout << object_code[i] << "\n";
        int len = 5 - (src[i][0].length());
        string m_record = "M";
        for(int y = 0 ; y < len+1 ; y++){
           m_record = m_record + "0";
        }
        int val = toDecimal(src[i][0]) + 1;
        address = "";
        base_conv(val, 16);
        m_record = m_record + address;
        m_record = m_record + "05";
        cout<<m_record;
        modification_record.push_back(m_record);
        continue;
     }
     else{
        cout <<  "error label does not exist " ;
        continue;
     }
  }
  else{
     if(operand[0] == '#' || operand[0] == '@'){
        string operand_label = operand.substr(1);
        string operand_address = symTable.get(operand_label);
        if(operand_address == ""){
           bool isNumber = true;
           int len  = operand.size();
           for(int j = 1; j < len && isNumber; j++){

              if(operand[j] < '0' || operand[j] > '9')
                 isNumber = false;
           }

           if(isNumber){

              int immediateNumber = strToDev(operand.substr(1));
              address = "";
              base_conv(immediateNumber , 16);
              if(address.size() > 5){
```

```
            printf("Immediate number is larger than 5 half bytes");
            continue;

        } else if (address.size() < 5){

            while(address.size() != 5) address = "0" + address;

        }

        string hexaOpernadNum = address;
        int tempNum = toDecimal(menomonicCode) + 1;
        if(operand[0] == '@'){
            tempNum++;
        }
        address = "";
        base_conv(tempNum , 16);
        while(address.size() != 2) address = "0" + address;
        menomonicCode = address;
        object_code[i] = menomonicCode + "1" + hexaOpernadNum;
              cout << object_code[i] << "\n";

        continue;

    }else{
        cout <<  "error label name " << operand_label;
        continue;
    }

}
if(operand[0] == '#'){
    int tempNum = toDecimal(menomonicCode) + 1;
    address = "";
    base_conv(tempNum , 16);
    while(address.size() != 2) address = "0" + address;
    menomonicCode = address;
    object_code[i] = menomonicCode + "1" + operand_address;
          cout << object_code[i] << "\n";

    continue;
}
else{
    int tempNum = toDecimal(menomonicCode) + 2;
    address = "";
```

```
                    base_conv(tempNum , 16);
                    while(address.size() != 2) address = "0" + address;
                    menomonicCode = address;
                    object_code[i] = menomonicCode + "9" + operand_address;
                        cout << object_code[i] << "\n";

                    continue;
                  }


                }
                else{
                  cout <<  "error format 4 lable does not exist ";
                  continue;

                }

              }
            }
          }
        }
```

If the input line has a directive symbol:

```
else if(src[i][2] == "BASE"){
        if(src[i][3] == "*"){
          base_address = src[i][0];
          base_used = true;
        }else{
          string operand = src[i][3];
          base_address = symTable.get(operand);
          if(base_address == ""){
            printf("label does't exist base statement");
            continue;
          }
          base_used = true;
        }
      } else if(src[i][2] == "NOBASE"){
        base_used = false;
        base_address = "";
        if(src[i][1]!= "" || src[i][3] != ""){
          printf("NOBASE statement does't have label nor operand ");
          continue;
        }
```

```
}else if(src[i][2] == "WORD"){
   regex integer ("\\d+");
   if(regex_match(src[i][3], integer)){
     address = "";
     int decimal1 = toDecimal(src[i][3]);
     base_conv(decimal1, 16);
     object_code[i] = address;
     cout<<address;
     continue;
   }else{
      printf("WORD must have a numeric operand");
      continue;
   }
}else if(src[i][2] == "BYTE"){
   regex in ("\\s*[C]['](.+)[']\\s*");
   regex in1 ("\\s*[X]['][0-9A-F]+[']\\s*");
   if(regex_match(src[i][3], in)){
      string obj_code = "";
      int len = (src[i][3].length()-1);
      for(int k = 2 ; k < len ; k++){
         int dec_val = src[i][3][k];
         address = "";
         base_conv(dec_val, 16);
         obj_code += address;
      }
      object_code[i] = obj_code;
      cout<<obj_code;
      continue;
   }else if(regex_match(src[i][3], in1)){
      string obj_code = "";
      int len = (src[i][3].length()-1);
      for(int k = 2 ; k < len ; k++){
         obj_code += src[i][3][k];
      }
      object_code[i] = obj_code;
      cout<<obj_code;
      continue;
   }else{
      printf("wrong constant definition");
   }
```

Check if the relative address could be used:

```
bool GenerateObjectCode::is_relative_address(string target_address, string pc, bool base_used,
string base_address, int i, string object_code[], string menomonicCode, int add_m, string
add_pc, string add_base)
{
   cout << "TA = " <<  target_address << "\n";
   cout << "PC = " << pc << "\n";

   int diff = toDecimal(target_address) - toDecimal(pc);
   if(diff >= -2048 && diff <= 2047){
      if(diff < 0){
         diff = diff * -1;
         diff--;
         diff = 4095 - diff;
      }
      address = "";
      base_conv(diff,16);
      while(address.size() != 3) address = "0" + address;
      string diff_hex = address;
      int tempNum = toDecimal(menomonicCode) + add_m;
      address = "";
      base_conv(tempNum , 16);
      while(address.size() != 2) address = "0" + address;
      menomonicCode = address;
      object_code[i] = menomonicCode + add_pc + diff_hex;
            cout << object_code[i] << "\n";

      return true;
   }
   else if(base_used){

      diff = toDecimal(target_address) - toDecimal(base_address);
      if(diff >= 0 && diff <= 4095){
         address = "";
         base_conv(diff,16);
         while(address.size() != 3) address = "0" + address;
         string diff_hex = address;
         int tempNum = toDecimal(menomonicCode) + add_m;
         address = "";
         base_conv(tempNum , 16);
         while(address.size() != 2) address = "0" + address;
```

```
            menomonicCode = address;
            object_code[i] = menomonicCode + add_base + diff_hex;
                  cout << object_code[i] << "\n";

            return true;
         }
      }
      return false;

}
```

Literal table:

```
bool LiteralTable::exists(string str)
{
   map<string, string>::iterator it;
   it = litMap.find(str);
   if(it != litMap.end())
   {
      return true;
   }
   return false;
}i
void LiteralTable::add(string key, string address)
{
   litMap[key] = address;
}

string LiteralTable::get(string key)
{
   if(exists(key))
   {
      return litMap.find(key)->second;
   }
   else
   {
      return "";
 }
}
map<string, string> LiteralTable::getMap()
{
   return litMap;
}
```

Check directives symbols in the engine class:

```
// EQU
if(stringcmp(p.src[i][2].c_str(), equ.c_str()) == 0){
   if((p.src[i][1] == "") || (p.src[i][3] == "")){
      printLine(i);
      myfile << "\t\terror label and operand field must not be empty!!" << endl;
      p.src[i+1][0] = p.src[i][0];
      continue;
   }
   if(isAbsolute(p.src[i][3])){
      p.src[i+1][0] = p.src[i][0];
      p.src[i][0] = address;
      symtable.add(p.src[i][1], p.src[i][0]);
      symtable.makeAbsolute(p.src[i][1]);
      printLine(i);
      continue;
   }
   else if(isRelative(p.src[i][3])){
      p.src[i+1][0] = p.src[i][0];
      p.src[i][0] = address;
      printLine(i);
   }
   else if(p.src[i][3] == "*"){
      printLine(i);
      p.src[i+1][0] = p.src[i][0];
   }else {
      printLine(i);
      myfile << "\t\tError : invalid operand!!" << p.src[i][3] << endl;
      p.src[i+1][0] = p.src[i][0];
      continue;
   }
} // ORG
else if(stringcmp(p.src[i][2].c_str(), org.c_str()) == 0){
   printLine(i);
   if(p.src[i][1] != ""){
      myfile << "\t\terror there mustn't be a symbol in label field !!" << endl;
   }
   if(isRelative(p.src[i][3])){
      p.src[i+1][0] = address;
   }
   else if(p.src[i][3] == "*"){
```

```
            //do nothing
        }
        else {
            myfile << "\t\tError : invalid operand!!" << p.src[i][3] << endl;
            p.src[i+1][0] = p.src[i][0];
        }
    }
```

Check if the operand is absolute or relative:

```
bool Engine::isRelative(string operand){

  if(regex_match(operand, regex(".+(\\+|\\-)[0-9]+"))){
    smatch m;
    regex_search (operand, m, regex("\\+|\\-"));
    if(symtable.exists(operand.substr(0,m.position(0)))){
      calculate(operand.substr(0,m.position(0)), m[0],
operand.substr(m.position(0)+1,operand.size()));
      return true;
    }
    return false;
  }
  else if(regex_match(operand, regex("[0-9]+\\+.+"))){
    smatch m;
    regex_search (operand, m, regex("\\+"));
    if(symtable.exists(operand.substr(m.position(0)+1,operand.size()))){
      calculate(operand.substr(0,m.position(0)), m[0],
operand.substr(m.position(0)+1,operand.size()));
      return true;
    }
    return false;
  }else if(symtable.exists(operand)){
    address = symtable.get(operand);
    return true;
  }
  return false;
}

bool Engine::isAbsolute(string operand){

  if(regex_match(operand, regex("[0-9]+((\\+|\\-|\\*|\\/)[0-9]+)*"))){
    if(isNum(operand)){
      address = "";
      base_conv(strToDev(operand), 16);
      return true;
```

```
        }
        smatch m;
        regex_search (operand, m, regex("\\+|\\-|\\*|\\/"));
        calculate(operand.substr(0,m.position(0)), m[0],
operand.substr(m.position(0)+1,operand.size()));
        return true;
    }
    else if(regex_match(operand, regex(".+\\-.+"))){
        smatch m;
        regex_search (operand, m, regex("\\-"));
        if(symtable.exists(operand.substr(0,m.position(0))) &&
symtable.exists(operand.substr(m.position(0)+1,operand.size()))){
            calculate(operand.substr(0,m.position(0)), m[0],
operand.substr(m.position(0)+1,operand.size()));
            return true;
        }
        return false;
    }
    else if(regex_match(operand, regex(".+(\\+|\\-|\\*|\\/).+"))){
        smatch m;
        regex_search (operand, m, regex("(\\+|\\-|\\*|\\/)"));
        if(symtable.isAbs(operand.substr(0,m.position(0))) &&
symtable.isAbs(operand.substr(m.position(0)+1,operand.size()))){
            calculate(operand.substr(0,m.position(0)), m[0],
operand.substr(m.position(0)+1,operand.size()));
            return true;
        }
        return false;
    }
    else if(regex_match(operand, regex(".+(\\+|\\-|\\*|\\/)[0-9]+"))){
        smatch m;
        regex_search (operand, m, regex("\\+|\\-|\\*|\\/"));
        if(symtable.isAbs(operand.substr(0,m.position(0)))){
            calculate(operand.substr(0,m.position(0)), m[0],
operand.substr(m.position(0)+1,operand.size()));
            return true;
        }
        return false;
    }
    else if(regex_match(operand, regex("[0-9]+(\\+|\\-|\\*|\\/).+"))){
        smatch m;
        regex_search (operand, m, regex("\\+|\\-|\\*|\\/"));
        if(symtable.isAbs(operand.substr(m.position(0)+1,operand.size()))){
```

```
            calculate(operand.substr(0,m.position(0)), m[0],
operand.substr(m.position(0)+1,operand.size()));
            return true;
        }
        return false;
    }else if(symtable.isAbs(operand)){
        address = symtable.get(operand);
        return true;
    }
    return false;
}
```

*Calculate the given input expressions:

```
void Engine::calculate(string left, string operat, string right){

    int n1, n2;
    if(isNum(left)){
        n1 = strToDev(left);
    }else{
        n1 = toDecimal(symtable.get(left));
    }

    if(isNum(right)){
        n2 = strToDev(right);
    }else {
        n2 = toDecimal(symtable.get(right));
    }

    address = "";
    if(operat=="+"){
        base_conv(n1+n2, 16);
    }
    else if(operat=="-"){
        base_conv(n1-n2, 16);
    }
    else if(operat=="*"){
        base_conv(n1*n2, 16);
    }
    else if(operat=="/"){
        base_conv(n1/n2, 16);
    }
}
```

Sample runs

source code:

```
RDSTR    START   0000
BGN      LDX     #0
         LDA     #0      comment1        comment2
RLOOP    TD      INDEV    comment
         JEQ     RLOOP
         RD      INDEV
         COMP    #0
         JEQ     EXIT
.This is a comment

         TIX     #0
         J       RLOOP
EXIT     RMO     X,A
         J       *               comment
INDEV    BYTE    X'F3'
         END     BGN
```

output:

Line	Address	Label	Mnemonic	Operand	Comment
1	0000	RDSTR	START	0000	
2	0000	BGN	LDX	#0	
3	3		LDA	#0	comment1 comment2
4	6	RLOOP	TD	INDEV	comment
5	9		JEQ	RLOOP	
6	C		RD	INDEV	
7	F		COMP	#0	
8	12		JEQ	EXIT	
9	15				.This is a comment
10	15				
11	15		TIX	#0	
12	18		J	RLOOP	
13	1B	EXIT	RMO	X,A	
14	1D		J	*	comment
15	20	INDEV	BYTE	X'F3'	
16	21		END	BGN	

****

S Y M B O L T A B L E

Label	Address
BGN	0000
EXIT	1B
INDEV	20
RLOOP	6

source code:

```
.5431266336666
prob1       start         1000
                          lde    BETA
ham     byte c'dfghj sv '
kjn     byte   xl2'
GAMMA    WORD     5
         end
```

output:

```
Line    Address Label           Mnemonic        Operand         Comment

1                                                               .5431266336666
2       1000    prob1           start           1000
3       1000                    lde             BETA
                error: invalid operation code 'lde'
4       1000    ham             byte            c'dfghj sv '
5       1009    kjn             byte                            xl2'
                error in matching operand field
6       1009    GAMMA           WORD            5
7       100C                    end


                ****************************************


                        S Y M B O L     T A B L E

                        Label           Address

                        GAMMA           1009
                        ham             1000
                        kjn             1009
```

source code:

```
SEARCH   START   0000

BGN      LDX     #0
                         LDCH    TARGET                comment1
                 RMO     A,S     comment2
   LOOP     LDCH        STRING,X
            COMPR    A,S
           JEQ              FOUND
           TIX       LENGTH
              JLT                    LOOP
              RMO         T,A
                    .     comment2
                  J       *
FOUND       LDA     #STRING
                  ADDR    X,A
                  J       *
         STRING   BYTE    C'ABC DEFG'
           LENGTH   WORD    8
TARGET    BYTE    C'D'
END      BGN
```

output:

```
2
3       0000    SEARCH          START           0000
4       0000
5       0000    BGN             LDX             #0
6       3                       LDCH            TARGET          comment1
7       6                       RMO             A,S             comment2
8       8       LOOP            LDCH            STRING,X
9       B                       COMPR           A,S
10      D                       JEQ             FOUND
11      10                      TIX             LENGTH
12      13                      JLT             LOOP
13      16                      RMO             T,A
14      18                                                      . comment2
15      18                      J               *
16      1B      FOUND           LDA             #STRING
17      1E                      ADDR            X,A
18      20                      J               *
19      23      STRING          BYTE            C'ABC DEFG'
20      2B      LENGTH          WORD            8
21      2E      TARGET          BYTE            C'D'
22      2F                      END             BGN

                ****************************************

                        S Y M B O L     T A B L E

                        Label           Address

                        BGN             0000
                        FOUND           1B
                        LENGTH          2B
                        LOOP            8
                        STRING          23
                        TARGET          2E
```

source code:

```
SEARCH   start
hassan  byte     x'jk'
        ldc  mahmoud
sallam  byte   mahmoud
        rsub    comment
```

output:

```
Line    Address Label           Mnemonic        Operand         Comment

1       0000    SEARCH          start
2       0000    hassan          byte            x'jk'
                error: operand is not hexadicimal number 'jk'
3       1                       ldc             mahmoud
                error: invalid operation code 'ldc'
4       1       sallam          byte                            mahmoud
                error in matching operand field
5       1                       rsub                            comment
                error: program must terminated with END statment


                ****************************************


                        S Y M B O L     T A B L E

                        Label           Address

                        hassan          0000
                        sallam          1
```

```
1  HCOPY  000000001077
2  T0000001D17202D69202D4B10103603202629000033200074B10105D3F2FEC032010
3  T00001D130F20160100030F200D4B10105D3E2003454F46
4  T0000301DB410B400B44075101000E32019332FFAE32013A00433200857C003B850
5  T0010531D3B2FEA1340004F0000F1B410774000E32011332FFA53C003DF2008B850
6  T00107073B2FEF4F000005
7  M00000705
8  M00001405
9  M00002705
10 E000000
11
```

```
HCOPY  000000000CEA
T0000001D17202D69202D4B100CA7032026290000332007 4B100CCF3F2FEC032010
T00001D130F20160100030F200D4B100CCF3E200348484B
T000CA71DB410B400B440751003E6E32019332FFAE32013A00433200857C003B850
T000CC41E3B2FEA1340004F0000F101B410774000E32011332FFA53C003DF2008B850
T000CE273B2FEF4F0000EE5
M00000705
M00001405
M00002705
E000000
```

```
COPY  START 0000
FIRST STL RETARD
LDB   #LENGTH
BASE LENGTH
CLOOP +JSUB REDREC
LDA LENGTH
COMP #0
JEQ ENDFIL
+JSUB WRREC
J CLOOP
ENDFIL LDA EOF
STA BUFFER
LDA #3
STA LENGTH
+JSUB WRREC
J @RETARD
EOF BYTE C'EOF'
RETARD RESW 1
LENGTH RESW 1
BUFFER RESB 4096
REDREC CLEAR X
CLEAR A
CLEAR S
+LDY #4096
TD INPUT
JEQ RLOOP
TD INPUT
COMPR A,S
JEQ EXIT
STCH BUFFER,X
TIXR T
JLT RLOOP
```

```
30  STCH    BUFFER,X
31  TIXR    T
32  JLT     RLOOP
33  EXIT    STX     LENGTH
34  RSUB
35  INPUT   BYTE    X'F101'
36  WRREC   CLEAR   X
37  LDT     LENGTH
38  WLOOP     TD    OUTPUT
39  JEQ     WLOOP
40  LDCH      BUFFER,X
41  WD      OUTPUT
42  TIXR    T
43  JLT     WLOOP
44  RSUB
45  OUTPUT    BYTE  X'EE5'
46  END     FIRST
```

```
 1  COPY    START   0000
 2  FIRST    STL    RETADR
 3  LDB     #LENGTH
 4  BASE    LENGTH
 5  CLOOP   +JSUB    RDREC
 6  LDA   LENGTH
 7  COMP   #0
 8  JEQ   ENDFIL
 9  +JSUB   WRREC
10  J    CLOOP
11  ENDFIL   LDA   EOF
12  STA    BUFFER
13  LDA   #3
14  STA   LENGTH
15  +JSUB   WRREC
16  J   @RETADR
17  EOF    BYTE   C'HHK'
18  RETADR   RESW   1
19  LENGTH   RESW   1
20  BUFFER   RESB   3185
21  RDREC    CLEAR   X
22  CLEAR   A
23  CLEAR   S
24  +LDT    #999
25  RLOOP   TD   INPUT
26  JEQ    RLOOP
27  TD    INPUT
28  COMPR   A,S
29  JEQ    EXIT
30  STCH    BUFFER,X
31  TIXR    T
32  JLT    RLOOP
```

```
32  JLT RLOOP
33  EXIT STX LENGTH
34  RSUB
35  INPUT BYTE X'F1'
36  WRREC CLEAR X
37  LDT LENGTH
38  WLOOP TD OUTPUT
39  JEQ WLOOP
40  LDCH BUFFER,X
41  WD OUTPUT
42  TIXR T
43  JLT WLOOP
44  RSUB
45  OUTPUT BYTE X'05'
46  END FIRST
47
```