Term Project - SIC/XE Assembler Phase (1&2)

Team Members (Ordered Alphabatically)

-Bassam Tarek Khalifa

-Khaled Ahmed Elaish

-Omar el Farouk Ahmed

-Sherif Mohi Mahmoud

-Tarek Eltalawi

pass 1

Requirements specifications.

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. You should build a parser that is capable of handling source lines that are instructions, storage declaration, comments, and assembler directives (a directive that is not implemented should be ignored possibly with a warning)
1. For instructions, the parser is to minimally be capable of decoding 2, 3 and 4-byte instructions as follows:
a. 2-byte with 1 or 2 symbolic register reference (e.g., TIXR A, ADDR S,A)
b. RSUB (ignoring any operand or perhaps issuing a warning)
c. 3-byte PC-relative with symbolic operand to include immediate, indirect, and indexed addressing
d. 3-byte absolute with non-symbolic operand to include immediate, indirect, and indexed addressing
e. 4-byte absolute with symbolic or non-symbolic operand to include immediate, indirect, and indexed addressing
2. The parser is to handle all storage directives (BYTE, WORD, RESW, and RESB).
4. The output of this phase should contain (at least):
1. The symbol table.
2. The source program in a format similar to the listing file described in your text book except that the object code is not generated as shown below. A meaningful error message is printed below the line in which the error occurred.

pass 2 :

Specifications :

a) The assembler is to execute by entering

assemble <source-file-name>

b) The source file for the main program for this phase is to be named assemble.cpp

c) The output of the assembler should include (at least):

1. Object-code file whose format is the same as the one described in the text book in section 2.1.1 and 2.3.5.

2. A report at the end of pass2. Pass1 and Pass2 errors should be included as

part of the assembler report, exhibiting both the offending line of source code

and the error.

d) The assembler should support:

1. EQU and ORG statements.

2. Simple expression evaluation. A simple expression includes simple (A

<op> B) operand arithmetic, where <op> is one of +,-,*,/ and no spaces

surround the operation, eg. A+B.

Bonus

1. General expression evaluation.

2. Literals (Including LTORG)

=C'<ASCII-TEXT>' , =X'HEX-TEXT' , =<DECIMAL-TEXT> forms.

3. Control sections

◦ Design.

Very Basic Design.

Two Secondary Object classes

First Class is the ID class which encloses information about SIC/XE operations. The op-code and the format whether it's 1, 2,or 3/4.

another class symOb, which is the object pushed in the hash map it contains the added address and a Boolean variable to know if this address is relocatable or absolute.

another LitOb class that stores objects of the litTab as an integer adresses.

◦ Main data structures :

- HashMaps : C++ built in hash maps of String and objects of written classes which holds information (example : OpTab < String operation , ID (int Opcode, int Format) >)
- String arrays that represent the possible field in a SIC/XE program.

◦ Algorithms Description for pass 1:

Main Process Algorithm:

Pseudo code

```
-  // Read first line & check opcode
-  If (opcode = START) THEN
-     Locctr = #operand
-  End IF
-  // loop over the code till END
-  While (opcode != END) do
-     If there's a symbol in LABEL field , THEN
-        Search for it in the Symbol Table
-        If (found)THEN
-           Duplicate error => true
-        Else
-           Insert (Label, Locctr) into the Symbl Table
-        END IF
-  // increment the LOCCTR
-     Search for the opcode in the OPTABLE
-     If (found)
-        Add number of bytes of opcode(2,3,4) to the LOCCTR
-     Else if(opcode = WORD)
-        Add 3 to LOCCTR
-     Else if (opcode = RESW)
-        Add 3 * #operand to LOCCTR
-     ELSE if (opcode = RESW)
-        Add  #operand to LOCCTR
-     ELSE if(opcode = BYTE)
```

- Add length of the constant to the LOCCTR
- ELSE
- Invalid operation code => true
- Read Next Line
- END WHILE
- Get Length of the program = LOCCTR – Starting Address

How it works:

Basic idea

First, the program reads the first line of the assembly code and checks that its opcode is called "START", and assign the LOCCTR to the starting address which is found in the operand field of the first line.

Then the program loops as long as the opcode isn't equal to "END", and checks if there's a symbol in the LABEL field and add it in the symbol table iff it doesn't already exists in it.

After that, the program increments the LOCCTR in one of three ways, either the opcode is found in optable therefore add number of bytes of opcode, or the opcode equals WORD therefore add 3, or opcode equals Byte therefore add length of the constant, or the opcode equals RESW therefore add 3* number in the operand, or the opcode equals RESB therefore add number in the operands, or the opcode doesn't satisfy any of these conditions and therefore there will be an error.

Finally get the size of the program.

General Algorithms -Utilities-

Read line by line from input file

-reads a line from a txt file till the end of file

```
string fgetLine(FILE* file)
{
    char str[77];
    if ( fgets (str , 77 , infile) == NULL )
    {
        eFile = -1; // END OF FILE( eFile : GLOBAL )
    }
    string d(str);
    unsigned  pos = d.find("\n");
    d =d.substr (0,pos);// IGNORE THE RUBISH
    return uppercaseString(d);
}
```

Convert letters into UpperCase

-loops on every character and converts them using toupper(char)function.

```
string uppercaseString(string s)
{
    char c;
    int i=0;
    while (s[i])
    {
        c=s[i];
        s[i] = (char) toupper(c);
        i++;
    }
    return s;
}
```

Print in output file

```
void print(FILE* pFile,string s)
{
    fputs((uppercaseString(s)+"\n").c_str(),pFile);
}
```

Print algorithm with specific format in output file

- takes a string array and lin number and location and formats a specific string to a specific format then calls the output function above.

```
string format( int line , int loc , string s [] )
{
    string out=""; // the output string
    string lineStr = toString(line);// line number in string
    string locStr = inc(loc);// loc counter in string
    out += lineStr+" "; // add line value to first for out string
// for loop to reserve the space in various length for string
    for ( int i = 0 ; i < (6 - locStr.size()) ; i++)
    {
        out+='0';
    }
    out += locStr+" ";
    out += s[0]; // Label value
```

```
    for ( int i = 0 ; i < (8- s[0].size()); i++ )
    {
        out +=' ';
    }
    out+=' ';
    out += s[1];// Mnemonic value
    for ( int i = 0 ; i < (6- s[1].size()); i++ )
    {
        out +=' ';
    }
    out+="  ";
    out += s[2];// Operand value
    for ( int i = 0 ; i < (30 - s[2].size()) ; i++)
    {
        out +=' ';
    }
    out+=s[3]; // comment value
    out+="\n";
    return out;
}
```

Open file with dynamic or static path

```
//Create Folder if the default folder not exist

void CreateFolder(const char * path)
{
    if(!CreateDirectory(path ,NULL))
    {
        return;
    }
}

// initFiles to handle dynamic and static path for open file form any folder
void initFiles(int argc, char *argv[]){
CreateFolder("C:\\AssemblerFiles\\");// create default folder
    string filename(argv[1]);
    if ( argc > 2 )//if argument have path
    {
        string Input = argv[2]; // path string ( dynamic path )
        infile =  fopen(argv[2],"r"); // open the file
        Input = Input.substr(0,Input.size()-4)+"OUT.txt";// out file path
        outfile = fopen(Input.c_str(),"w"); // open output file
    }
    else
    {
        // static path ( if files exist in default folder )
        infile = fopen(("C:\\AssemblerFiles\\"+filename+".txt").c_str(),"r");
        outfile = fopen (("C:\\AssemblerFiles\\"+filename+"OUT.txt").c_str(),"w");
    }
// check file opened or not
    if(infile == NULL)
    {
        cout << " File Can't Open !! ";
    }
    if(outfile == NULL)
    {
        cout << "\n   OUT FILE ERROR !! ";
```

```
        outfile = fopen (("C:\\AssemblerFiles\\"+filename+"OUT.txt").c_str(),"w");
        cout << "\n   BUT THE OUT FILE IN ( AssemblerFiles ) FOLDER "<< endl;
    }
    cout <<"\n\n   successfully loaded ( File : "<< filename<<" )\n\n"<<endl;
}
```

Print SYMTAB in OUTPUT FILE

- a method that loops on the symTab hashmap that is created at run-time to print at the end of pass one in the output file.

```
void printSymTab()

{
    map<string,symOb*>::iterator it;
    for (map<string,symOb*>::iterator it=symTab.begin();it!=symTab.end();++it)
    {
        symOb* symTemp = it->second;
        string name =it->first;
        int temp1 = symTemp->getAddress();
        int temp2 = symTemp->getRelocatable();
        print(outfile,rowTableFormat(name,temp1,temp2));
    }
}


string rowTableFormat(string name,int address , int r)
{
// Get format of row's to print
    string out="";
    for ( int i = 0 ; i < 5 ; i++)
    {
        out+=' ';
    }
    string addstr = hexaChange(address);
    string reAbs = (r == 1)? "Relocatable" : "Absolute";
    out += name;
    for ( int i = 0 ; i < (12 - name.length()) ; i++)
    {
        out+=' ';
    }
    out+=addstr;
    for ( int i = 0 ; i < (12 - addstr.size()) ; i++)
    {
        out+=' ';
    }
    out += reAbs;
    out+="\n";
    return out;
}
```

CheckWord():

this method is called when a certain algorithm wants to check if this operand for the word directive is valid or not.

```
bool checkWord(string word)
{
    int i = 0;

    if(word[i] == '-')
    {
        i++;
    }

    for( ; i < word.size() ; i++)
    {

        if(word[i] > '9' || word[i] < '0')
        {
            return false;
        }

    }
    return true;
}
```

ChecByte():

this method is called when a certain algorithm wants to check if this operand for the Byte directive is valid or not. returns -1 if not, returns the actual number in the operand field if yes.

```
int checkByte(string byte)
{

    if(byte[0] != 'C' && byte[0] != 'c' && byte[0] != 'X' && byte[0] != 'x')
    {
        cout << "NO X OR C" << endl;
        return -1;
    }

    if(byte[1] != '\'' )
    {
        cout << "first comma" << endl;
        return -1;
    }

    if(byte[byte.size() - 1] != '\'' )
    {
        cout << "last comma" << endl;
        return -1;
    }

// a character input
    if(byte[0] == 'C' || byte[0] == 'c')
    {
        // check
        cout << "number of chars" << endl;
        return byte.size()-3;
```

```
    }


// a hex word input
    int numberOfHalfBytes = 0 ;
    if(byte[0] == 'X' || byte[0] == 'x')
    {

        for(int i = 2 ; i < byte.size()-1 ; i++)
        {
            if(((byte[i] >= '0')&&(byte[i] <= '9'))||((byte[i] >= 'a')&&(byte[i] <=
'f'))||((byte[i] >= 'A')&&(byte[i] <= 'F')))
            {

                numberOfHalfBytes++;
            }
            else
            {
                cout << "hex out of range " << byte[i] << endl;
                return -1;
            }
        }
        if(numberOfHalfBytes % 2 == 1)
        {
            cout << "odd number of hex half bytes" << endl;
            return -1;
        }
        return numberOfHalfBytes/2;
    }
    cout << "main return" << endl;
    return -1;
}
```

CheckResrve():

this method is called when a certain algorithm wants to check if this operand for the RESW/RESB directive is valid or not. returns -1 if not, returns the actual number in the operand field if yes.

```
int checkReserve(string res)
{

    int i = 0;
    if(res[i] == '-')
    {
        i++;
        return -1;
    }


    for ( ; i < res.size() ; i++)
    {
        if(res[i] < '0' || res[i] > '9' )
        {
            return -1;
        }
    }
```

```
    int temp = atoi (res.c_str());
    return temp;

}
```

Parsing Algorithms:

-There are two main parsing Methods

PrimaryParsing()

it parses the line into chunks according to empty white spaces or tabs.

it ignores any space after 3 words and understands them as a comment, which is later used for the second parsing method which handles the main parsing and is like the brain that understands where each word should be in an array of length four [LABEL, OPERATION, OPERAND, COMMENT] that the main processing algorithm uses.

```
int primaryParsing(string inputLine, string * line){

    bool spaceFound = false;
    bool commentField = false;
    int arrayPointer = 0;
    int actualArrayPointer = -1;
    bool actualWrite = false;
    int i=0;

    if(inputLine.empty())
    {
        return -1;
    }


    if(inputLine.at(0) == ' ')
    {
        spaceFound = true;
        actualWrite = false;
        i++;
    }

    for(; i<inputLine.size() ; i++)
    {

        if(inputLine.at(i) == ' ' && !spaceFound && !commentField)
        {
            spaceFound = true;
            arrayPointer++;
            actualWrite = false;

        }
        else if(inputLine.at(i) == ' ' && spaceFound && !commentField)
        {
            actualWrite = false;
            continue;
        }
        else if(inputLine.at(i) == '\t')
        {
```

```
            continue;
        }
        else
        {
//    concatinate on current array pointer;

            if(inputLine.at(i) == '.')
            {
//        put rest of the string in line[3]
                line[3] = inputLine.substr (i,inputLine.size()-i);
                return 0;
            }


            if(arrayPointer == 3)
            {
                commentField = true;
            }
            if(!actualWrite)
            {
                actualWrite = true;
                actualArrayPointer++;
            }
            line[arrayPointer] += inputLine.at(i);
            spaceFound = false;
        }

    }
return actualArrayPointer;
}
```

---

the second method in the parsing algorithms

Secondary Parsing()

This method is responsible for performing checks that every string in each of the array slots is as stated. The Algorithm is as follow :

if the array is full and the first slot has "operation code" then we should ask if the operation is of format 1 then the rest of the slots are comments and must be moved to slot 4, else then concatenate slot 3 and 4 where both of them are comment and set them in slot 4 , then shift the operation and the operand to the right slots.

this method uses a returned value from primary parsing, which is the number of slots that actually have words written in them. referred to as SplitSize a variable of which the main algorithm depends on.

The Map Method :

An initializer method which initializes the hash map with the opcodes and formats to each instruction.

this method was built with a clever trick. we obtained the appendix of the SIC/XE in a written format and then wrote a java parser program that reads that file line by line and then outputs in the java console a code written in C++ which is designed to be copy pasted directly without filling the hashmap by hand one by one.

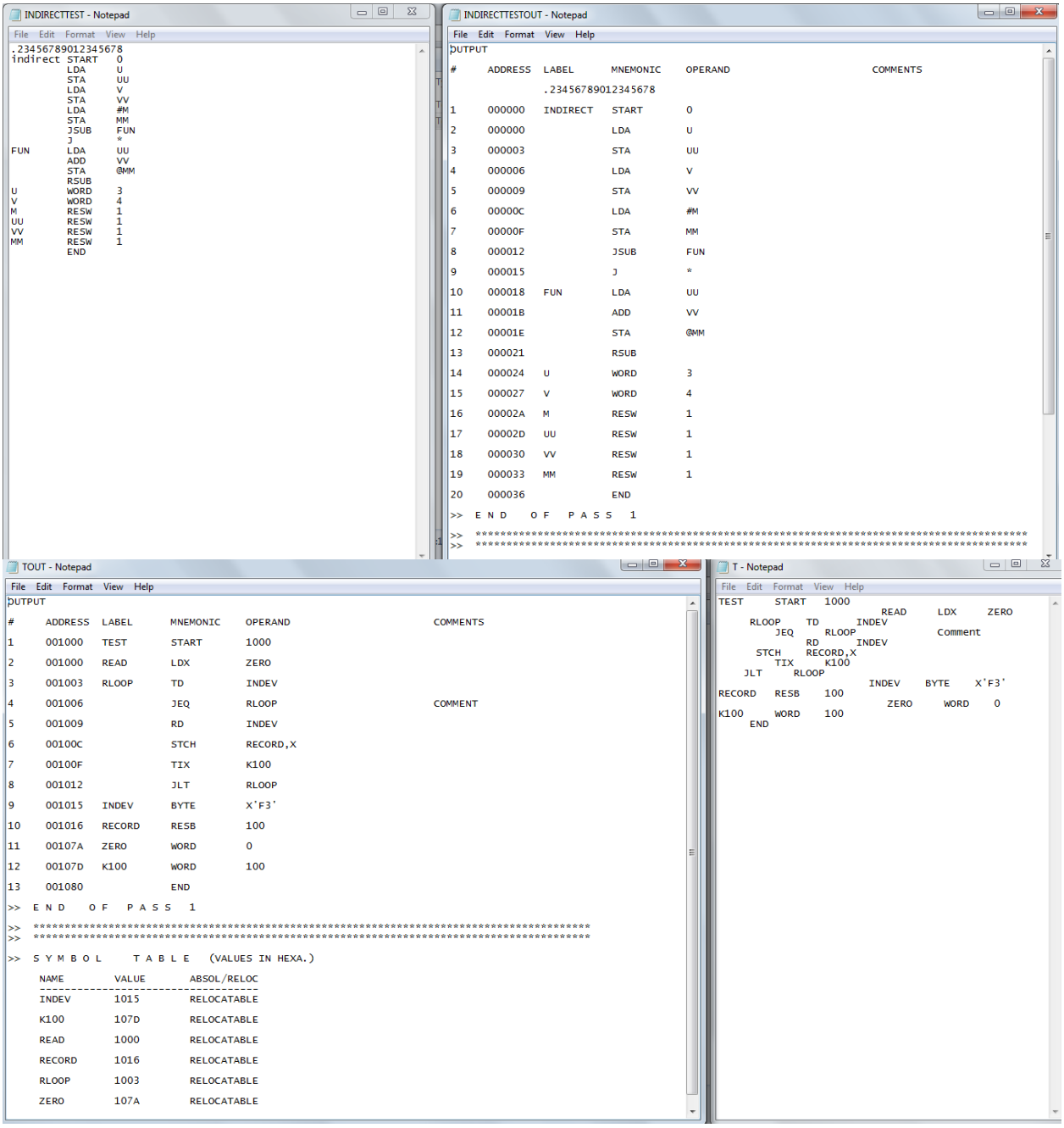
Assumptions pass 1 :

-It is assumed that the user cannot use a label name which is the same as the name of an instruction or a directive. Further information about this part in the bonus section.

Bonus :

- Format #4 instructions (instructions preceded by “+” operator), it checks first if the instruction contains “+” operator then remove from the string and sets a Boolean named formatFour to true, after that the Locctr is incremented by 4 instead of 3.
- “EQU” directive, the program handles Equate directive by directly printing the line containing “EQU” using the address in its operand field, and nothing happens to the Locctr so that it can be used normally for instructions after “EQU”.
- “ORG” directive, when the program reads line containing “ORG” it simply sets Locctr to number in operand field.
- Free format is implemented the used is free to write the four Fields in any place in the file, comments has been handled also, thus there can be two types of comments in the program : a line comment which is the normal case of a line starting with a (.) which means that this line is a comment. or a line which has it's fourth field as a comment. the parser checks to see of the instruction is of which format and thus knows if the word preceding is an operand or a comment with a certain algorithm specifically designed to handle this issue.
- The End directive is handled to be used as a label which breaks the assumption already stated, but we made little trials to test whether it can be done or not, and it worked. we didn't do it for the rest of the code for the sake of time and simplicity. also the end directive is harder than any other directive or instruction because it's the only one which can have an operand or not.
- A silent installer that adds the path of the .exe file of the program in the environment variables of the computer so that the program can run directly from the cmd.
- Two input methods for running the pass1 Program. a Drag and drop of the file to the cmd, which pass the argument to the program, this argument is the path of the file, the output file generated in the same folder from which the original file was dragged.
- expressions are handled
- Literals are also handled

Sample runs :



INDIRECTTEST - Notepad

```
.23456789012345678
indirect START   0
         LDA     U
         STA     UU
         LDA     V
         STA     VV
         LDA     #M
         STA     MM
         JSUB    FUN
         J       *
FUN      LDA     UU
         ADD     VV
         STA     @MM
         RSUB
U        WORD    3
V        WORD    4
M        RESW    1
UU       RESW    1
VV       RESW    1
MM       RESW    1
         END
```

INDIRECTTESTOUT - Notepad

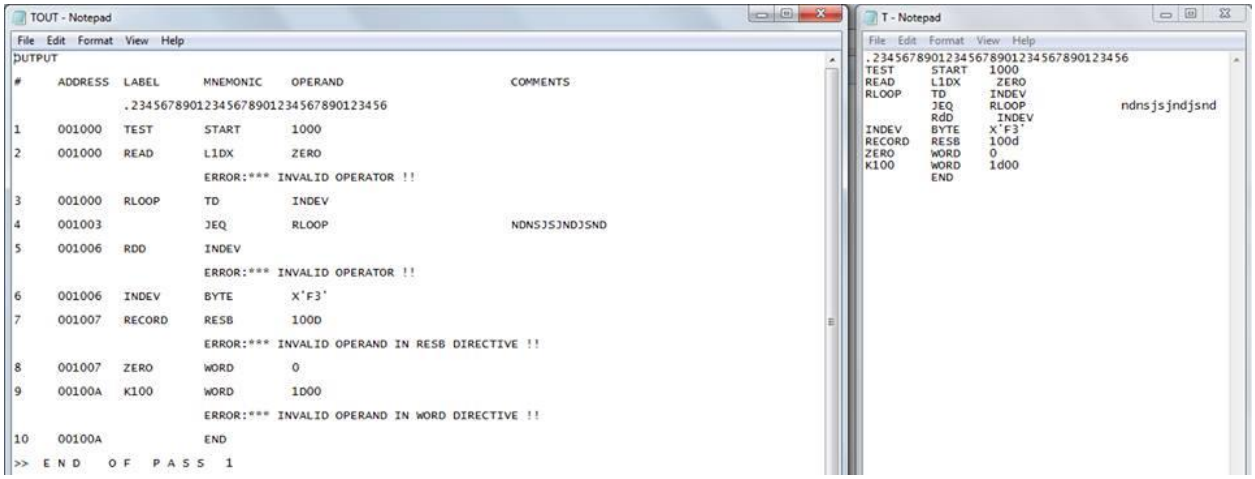
```
OUTPUT
#     ADDRESS  LABEL     MNEMONIC   OPERAND                  COMMENTS
                         .23456789012345678
1     000000   INDIRECT  START      0
2     000000             LDA        U
3     000003             STA        UU
4     000006             LDA        V
5     000009             STA        VV
6     00000C             LDA        #M
7     00000F             STA        MM
8     000012             JSUB       FUN
9     000015             J          *
10    000018   FUN       LDA        UU
11    00001B             ADD        VV
12    00001E             STA        @MM
13    000021             RSUB
14    000024   U         WORD       3
15    000027   V         WORD       4
16    00002A   M         RESW       1
17    00002D   UU        RESW       1
18    000030   VV        RESW       1
19    000033   MM        RESW       1
20    000036             END
>>  E N D    O F    P A S S    1
>>  **********************************************************************************************
>>  **********************************************************************************************
```

TOUT - Notepad

```
OUTPUT
#     ADDRESS  LABEL     MNEMONIC   OPERAND                  COMMENTS
1     001000   TEST      START      1000
2     001000   READ      LDX        ZERO
3     001003   RLOOP     TD         INDEV
4     001006             JEQ        RLOOP                    COMMENT
5     001009             RD         INDEV
6     00100C             STCH       RECORD,X
7     00100F             TIX        K100
8     001012             JLT        RLOOP
9     001015   INDEV     BYTE       X'F3'
10    001016   RECORD    RESB       100
11    00107A   ZERO      WORD       0
12    00107D   K100      WORD       100
13    001080             END
>>  E N D    O F    P A S S    1
>>  **********************************************************************************************
>>  **********************************************************************************************
>>  S Y M B O L      T A B L E    (VALUES IN HEXA.)
     NAME        VALUE       ABSOL/RELOC
     ------------------------------------
     INDEV       1015        RELOCATABLE
     K100        107D        RELOCATABLE
     READ        1000        RELOCATABLE
     RECORD      1016        RELOCATABLE
     RLOOP       1003        RELOCATABLE
     ZERO        107A        RELOCATABLE
```

T - Notepad

```
TEST      START   1000
                  READ     LDX      ZERO
     RLOOP    TD       INDEV
          JEQ      RLOOP              Comment
                   RD       INDEV
       STCH     RECORD,X
          TIX      K100
    JLT       RLOOP
                  INDEV    BYTE     X'F3'
RECORD    RESB     100          ZERO      WORD     0
K100      WORD     100
     END
```



TOUT - Notepad

File Edit Format View Help

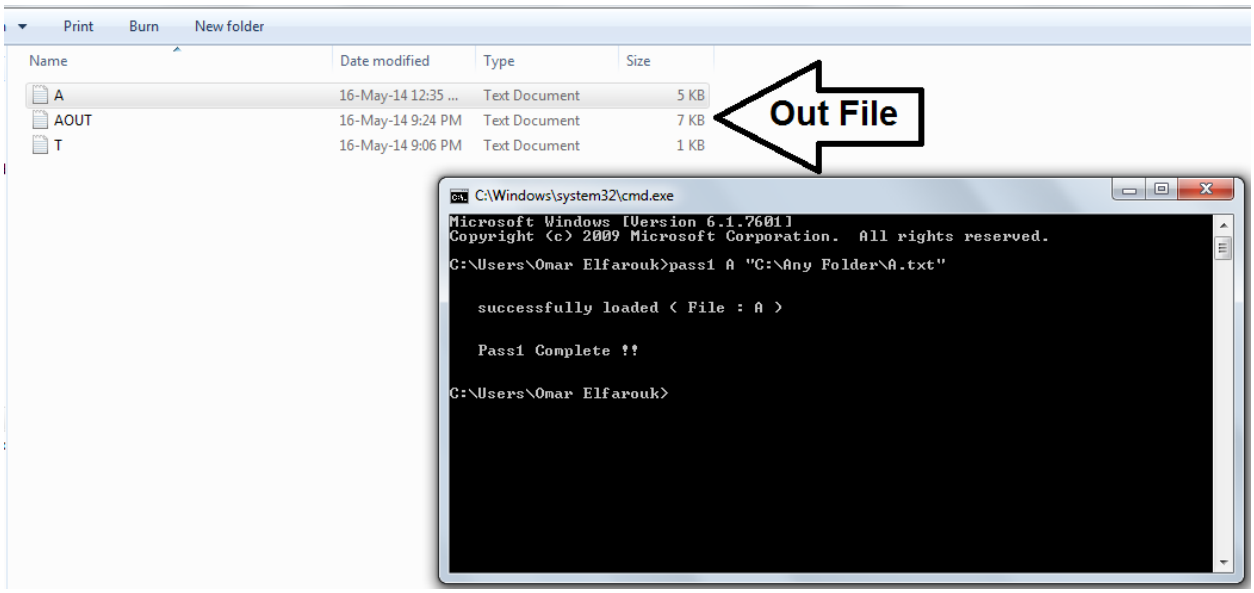
```
OUTPUT
#     ADDRESS  LABEL     MNEMONIC    OPERAND                    COMMENTS
                .23456789012345678901234567890123456
1     001000   TEST      START       1000
2     001000   READ      L1DX        ZERO
                         ERROR:*** INVALID OPERATOR !!
3     001000   RLOOP     TD          INDEV
4     001003             JEQ         RLOOP                      NDNSJSJNDJSND
5     001006   RDD       INDEV
                         ERROR:*** INVALID OPERATOR !!
6     001006   INDEV     BYTE        X'F3'
7     001007   RECORD    RESB        100D
                         ERROR:*** INVALID OPERAND IN RESB DIRECTIVE !!
8     001007   ZERO      WORD        0
9     00100A   K100      WORD        1D00
                         ERROR:*** INVALID OPERAND IN WORD DIRECTIVE !!
10    00100A             END
>> E N D   O F   P A S S   1
```

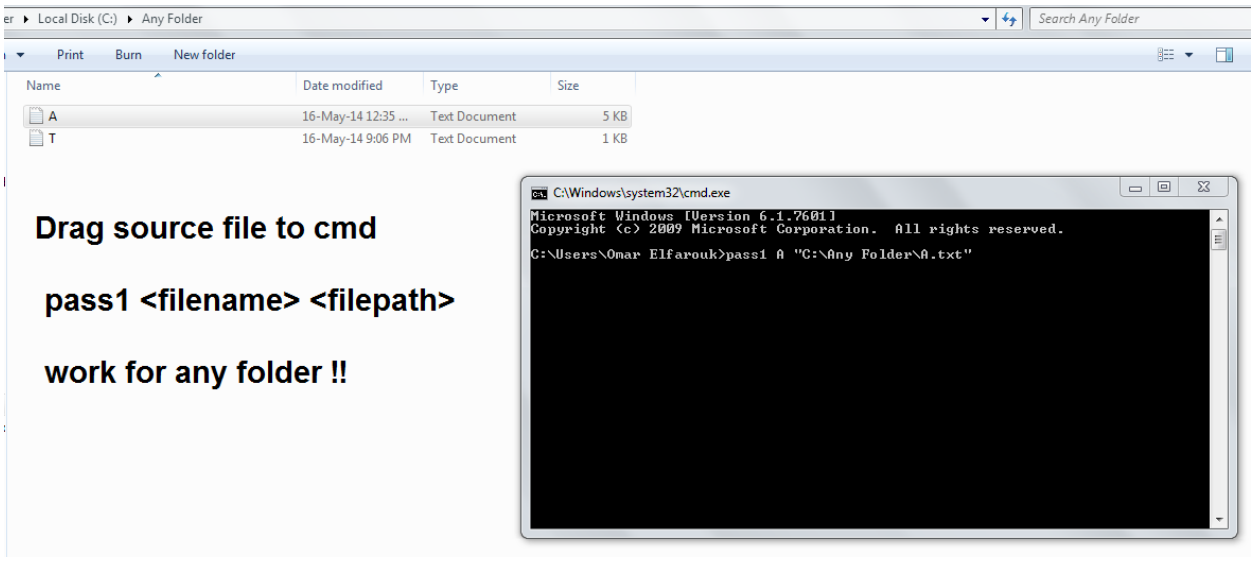
T - Notepad

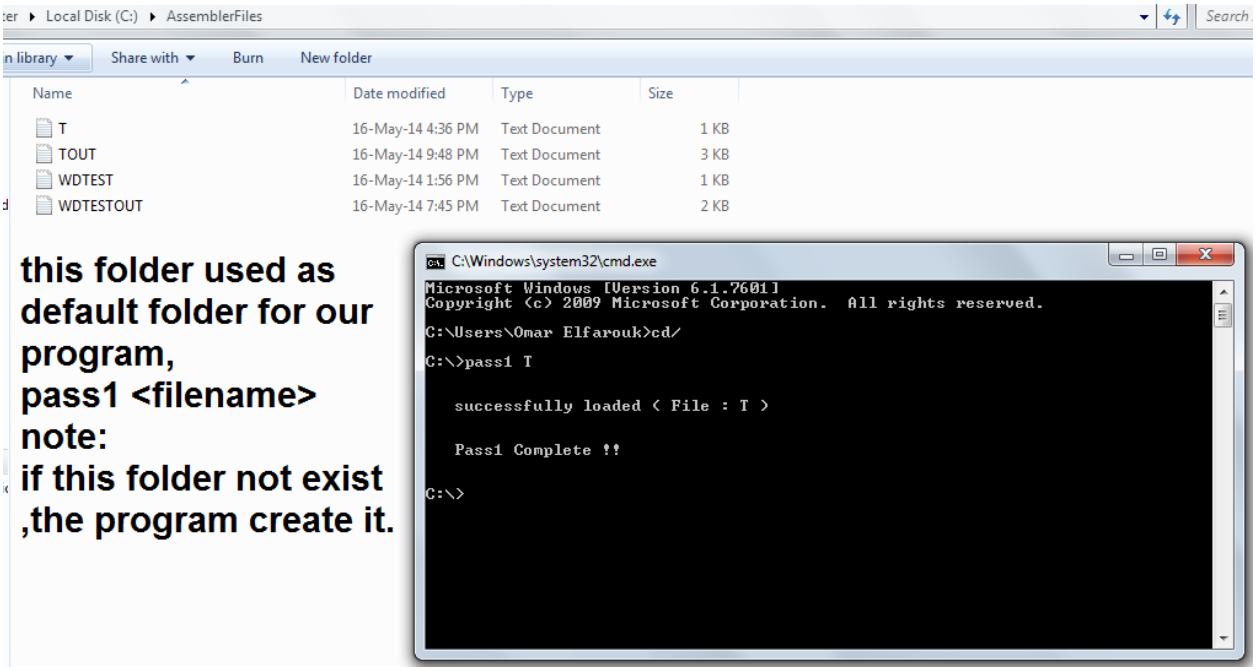
File Edit Format View Help

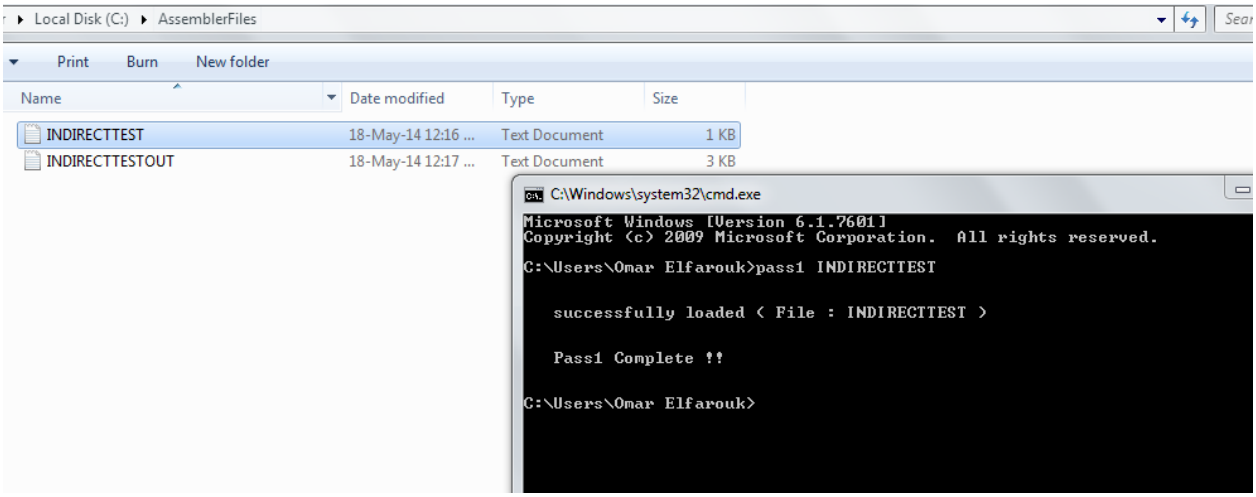
```
.23456789012345678901234567890123456
TEST      START   1000
READ      L1DX     ZERO
RLOOP     TD       INDEV
          JEQ      RLOOP                    ndnsjsjndjsnd
          RDD       INDEV
INDEV     BYTE     X'F3'
RECORD    RESB     100d
ZERO      WORD     0
K100      WORD     1d00
          END
```

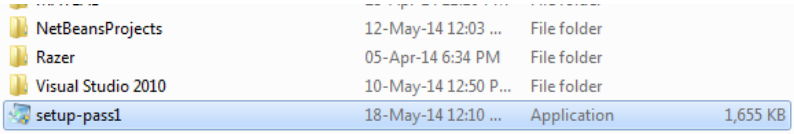
Bonus Screen Shots:




Local Disk (C:) ▸ Any Folder
Search Any Folder
Print
Burn
New folder
Name
Date modified
Type
Size
A
16-May-14 12:35 ...
Text Document
5 KB
T
16-May-14 9:06 PM
Text Document
1 KB
Drag source file to cmd
pass1 <filename> <filepath>
work for any folder !!
C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.
C:\Users\Omar Elfarouk>pass1 A "C:\Any Folder\A.txt"


Local Disk (C:) ▸ AssemblerFiles
Search
Share with
Burn
New folder
Name
Date modified
Type
Size
T
16-May-14 4:36 PM
Text Document
1 KB
TOUT
16-May-14 9:48 PM
Text Document
3 KB
WDTEST
16-May-14 1:56 PM
Text Document
1 KB
WDTESTOUT
16-May-14 7:45 PM
Text Document
2 KB
this folder used as default folder for our program,
pass1 <filename>
note:
if this folder not exist ,the program create it.
C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.
C:\Users\Omar Elfarouk>cd/
C:\>pass1 T
successfully loaded (File : T)
Pass1 Complete !!
C:\>





To initialize the program by add executable file path to environment variable

Parsing pass 2 :

Modify

How it works:

This method takes field 3 and field 4, which are (opcode and operand fields), apply operations on them and sets bits array accordingly, the bits array is the array which contains the "n-i-x-b-p-e" bits, these bits are responsible for telling whether it is format 4 or not, pc relative or base relative, immediate or indirect or neither nor and finally whether there is indexing or not.

Also this method calls method infixToPostfix, sending the operand as a parameter such that if the operand was expression then the string returned is the expression in postfix notation, however if the operand wasn't expression then the string returned will be the same string sent.

After that the modify method calls evaluatePostfix method sending the string returned from infixToPostfix method as a parameter, the evaluatePostfix method returns integer representing the address of the operand in both case whether the operand was expression or not.

Finally the modify method returns that address.

Algorithm:

```
Modify (Locctr, Field3, Field4)
    if (field3 is preceded by"+") Then
        eBit = 1
        remove "+" from field3
```

```
    end if
    else if (field4 is preceded by “@”) Then
        nBit = 1
        remove”@” from field4
    end if
    else If (field4 is preceded by “#”) Then
        iBit = 1
        remove”#” from field4
    end if
    else
        nBit = 1
        iBit = 1
    end if
    if(field4 contains “,X”at the last two indexes)Then
        xBit = 1
        remove”,X” from field4
    end if
    String postfix = infixToPostfix(field 4)
    // get field 4 in postfix notation
    int address = evaluatePostfix(postfix)
    // get address of operand
return address
```

Evaluate postfix :

This method takes a string converted from infix to postfix from another method

it iterates on the string concatinating each character if it was a Label or a number and adds it to the value stack

this algorithm was made to handle all kinds of expressions and know at the end of the expression if that expression was an Absolute expression, a Relative Expression or a "Neither-Nor" Expression.

it does so by using certain rules of the written algorithm

Data Structures for this algorithm : Two stacks of Integers : Value, Relativity

the value stack is the normal stack of the values of labels or numbers being pushed in the stack, like a normal postfix evaluation algorithm

the relativity stack is used whenever something is inserted in the value stack, the corresponding value is inserted in it

assume that anything absolute - i.e 1000 , 500 , Buffend-buffer - has a relative value of zero, so when anything absoulte is pushed in the value stack a zero is pushed in the relativity stack

but if a label was to be inserted we get it's value form the symtab and insert it into values, and get if it was relocatable or not , if it was we insert 1 in the relativity stack, meaning it has a relative rank or 1, if it was absoulte then we put 0, meaning it has relativity rank of 0

-if there were a '+', the normal protocol is done on the value stack , pop the last two add them and insert them back

but for the relativity stack we pop the last two and add them back as well

if the last two popped were 0 (corresponding to an absolute value) and 1 (a relative value) then their addition will make also a relative term (push 1 back in the relativity stack)

the same is done with '-' but subtracting the values and the relative terms

multiplication is done differently, if we found two relative terms multiplied by each other we send back an error to the calling line

if an absolute term is multiplied to a abosolute term, the values are popped multipled and pushed back, and relativity is still zero

the main case is when a relative term is multipplied by an absolute term , the value of the absolute term is multiplied by the relativity number of the relative term and pushed into the relativity term, the values is treated as basic operation of multiplication

the same is done with "/" division

at the end of the method basic checks are done like if more than element remains in one of the two stacks then this is an invalid postfix expression

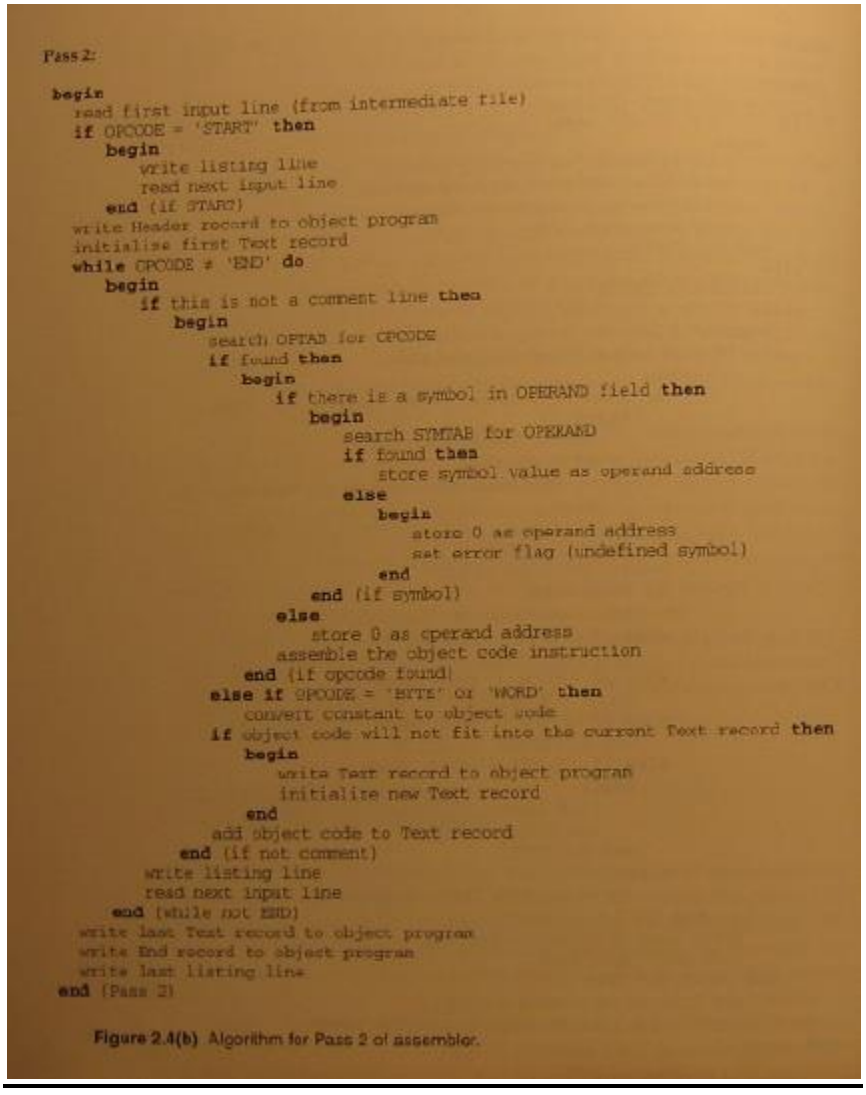
the last element in the value stack is returned to the calling line

the last element in the relativity stack, if it was one, then this expression has a relativity order of 1, meaning it's a relative valid term

if it was zero, then this expression has a relativity order of 0, meaning it's an absolute term

it it was any other value then it's an invalid expression when talking in terms of relativity, meaning it's a neither nor expression an error is returned to the calling line

Processing pass 2 algorithms:



Pass 2:

```
begin
  read first input line (from intermediate file)
  if OPCODE = 'START' then
     begin
        write listing line
        read next input line
     end (if START)
  write Header record to object program
  initialize first Text record
  while OPCODE ≠ 'END' do
     begin
        if this is not a comment line then
           begin
              search OPTAB for OPCODE
              if found then
                 begin
                    if there is a symbol in OPERAND field then
                       begin
                          search SYMTAB for OPERAND
                          if found then
                             store symbol value as operand address
                          else
                             begin
                                store 0 as operand address
                                set error flag (undefined symbol)
                             end
                       end (if symbol)
                    else
                       store 0 as operand address
                    assemble the object code instruction
                 end (if opcode found)
              else if OPCODE = 'BYTE' or 'WORD' then
                 convert constant to object code
              if object code will not fit into the current Text record then
                 begin
                    write Text record to object program
                    initialize new Text record
                 end
              add object code to Text record
           end (if not comment)
        write listing line
        read next input line
     end (while not END)
  write last Text record to object program
  write End record to object program
  write last listing line
end (Pass 2)
```

Figure 2.4(b) Algorithm for Pass 2 of assembler.

pass2Processing :

this method is responsible for looping on line by line coming from the parser which has parsed the intermediate file, and starts making checks on it so if the opcode is (start) to start writing the header record to the object code file by calling method (formatObjectCode),and then starts writing the first text record by while looping over the lines coming from the parser till operator(End) is found , checks if the line is comment then it call method (printOneLine) then checks if the opcode is found in the optab then starts to check the operand if it is in the symtab or if they are registers in case the opcode is format 2 or if it is immediate operand because all of this changes in the value of displacement which will be sent to method (formatObjectCode) which will combine all the bits into hexacharacter in string and send it to method print . if the operator was a directive like (byte) or (word) or .... , then it converts the value of the constant to object code, if the length of the line is finished a new line is started and the object code is written in this new line.

convertBitstoint :

this method takes string of bits , changes it to decimal number by checking 2 characters from the string .

findRegisterNumber :

this method takes a character which is a register and find its number ranging from 0 to 6

CheckOneRegister :

this method takes an operand and boolean called svc , if this boolean is true then the operator was svc and this means that the operand was a number . the target of this method is to help in the check done by method (checkRegisterOperand), so that when the operator is format 2 then the operand must be a register , so this method is specified for the operators of format 2 which takes only one character as an operand and returns an array of integer of size 3 where the first slot is a boolean (o -> false statement , 1 -> true statement) the second and the third slot are the values of the register or (n) in case of operator (svc).

CheckTwoRegister :

this method exactly perform the same functionality as the last described method but for the operators with operands of 2 registers and it returns array of 3 slots where the first slot is a boolean (o -> false statement , 1 -> true statement) the second and the third slot are the values of the register or (n) in case of operator (shiftr / shiftl).

CheckTwoRegister :

this method is the father method of the last 2 methods , it takes 2 strings of the operation and the operand . checks if the operation is of operation type that takes 2 registers or only one register and according to this call on of the two previous methods. it returns array of 3 slots as described

formatObjectCode :

this method is responsible for binding the bits to form the object code , this method returns string which is the object code as hexadecimal characters. at the beginning we start to check that the number of each input is correct, where the inputs are : int opcode which is the opcode as decimal and string bits which is nixbpe bits and string displacement and integer format of the opcode and int register 1 and inint register 2. this method. if the format is 1 or 2 then the object code is easy, just the object code or the object code + the number of the registers. if the operation is format 3 or format 4,first we add the decimal value of the object code + the decimal value of the 2 bits (ni) and set the value of the first 2 hexabytes , after this i add the hexadecimal character of the 4 bits (xbpe) and add the displacement to the output string and returns it .

Initialize Files :

This method to initialize files to make to objects file to read and write from same file. And to reach the first line in code.

```
void initPassTwo()
{
    readFile = fopen ("LISAFILE.txt","r");
    writeFile = fopen("LISAFILE.txt","a");
    objFile = fopen("OBJFILE.txt","w");
    char f[100];
    fgets(f,100,readFile);
    string g(f);
    while( g.at(0) != '#' )  // REACH FIRST LINE FOR THE CODE
    {
        g = fgetLine(readFile);
    }
    print(writeFile,">>
**************************************************************************
************");
    print(writeFile,">>  S t a r t   o f   P a s s   I I");
    print(writeFile,">>  A s s e m b l e d   p r o g r a m   l i s t i n
g");
    current = fgetLine(readFile);
    while ( current.at(0) == ' ' ){
        print(writeFile,current);
        current = fgetLine(readFile);
    }

}
```

Parse line (from lisa File):
Return string array with three parameter (next location counter , operator and operand)
the comment line print direct to file and get next line.

```
void getParsedLine(string out[])
{
    out[3] = current;
    if ( out[3].at(0) ==' ')
    {
        current = fgetLine(readFile);
        return;
    }
    if(out[3].at(0) =='>')
    {
        eFile = -1;
```

```
        return ;
    }
    int length=0;
    for ( int i = 26 ; i <current.size() ; i++)
    {
        if ( current.at(i) ==' ' )
        {
            break;
        }
        length++;
    }
    out[1] = current.substr(26,length);
    length=0;
    for (int i = 38 ; i < current.size(); i++)
    {
        if ( current.at(i) ==' ' )
        {
            break;
        }
        length++;
    }
    out[2] = current.substr(38,length);
    currentLoc = current.substr(6,6);;
    current = fgetLine(readFile);
    while ( current.at(0)==' ' ){
        print(writeFile,current);
        current = fgetLine(readFile);
    }
    out[0] = current.substr(6,6);
}
```

Split by specific string (for every char for string):
Return vector of string.

```
vector<string> split(string str1,string sf)
{
    vector<string> f;
    char * pch;
    for ( int i = 0 ; i < str1.length() ; i++ )
    {
        if ( str1.at(i) > 41 && str1.at(i) < 48 || str1.at(i)=='(' ||
str1.at(i)==')' )
        {
            str1 = str1.substr(0,i)+' '+str1.at(i)+'
'+str1.substr(i+1,str1.length());
            i++;
        }
    }
    char str[100];
    strcpy(str,str1.c_str());
    pch = strtok (str,sf.c_str());
    while (pch != NULL)
    {
        f.push_back(pch);
        pch = strtok (NULL, sf.c_str());
    }
```

```
    return f;
}
```

Convert infix expression to postfix:

```
string infixToPostfix(vector<string> str)
{
    stack<string> stack;
    string postFix="";
    int comingPr=0;
    int topPr=0;
    for ( int i = 0 ; i < str.size() ; i++ )
    {
        if( str.at(i).at(0)=='(' )
        {
            stack.push(str.at(i));
        }
        else if(str.at(i).at(0) ==')')
        {
            while( stack.top().at(0) !='(' )
            {
                postFix += stack.top()+" ";
                stack.pop();
            }
            stack.pop();
        }
        else if ( str.at(i).at(0) > 41 && str.at(i).at(0) < 48  )
        {
            if( str.at(i).at(0)=='(' )
            {
                stack.push(str.at(i));
            }
            else if(str.at(i).at(0) ==')')
            {
                while( stack.top().at(0) !='(' )
                {
                    postFix += stack.top()+" ";
                    stack.pop();
                }
                stack.pop();
            }
            else if ( str.at(i).at(0) == '+' || str.at(i).at(0) == '-' )
            {
                comingPr=1;
            }
            else
            {
                comingPr=2;
            }
            if ( !stack.empty() && (stack.top().at(0) == '+' ||
stack.top().at(0) == '-') )
            {
                postFix += stack.top()+" ";
                stack.pop();
                topPr =1;
            }
```

```
            else if(!stack.empty() && (stack.top().at(0) == '*' ||
stack.top().at(0) == '/') )
            {
                topPr =2;
            }
            if ( topPr < comingPr || stack.top().at(0) == '(' )
            {
                stack.push(str.at(i));
            }
            else
            {
                while( !stack.empty() && topPr >= comingPr )
                {
                    postFix += stack.top()+" ";
                    stack.pop();
                    if ( !stack.empty() && ( stack.top().at(0) == '+' ||
stack.top().at(0) == '-' ) )
                    {
                        topPr =1;
                    }
                    else if ( !stack.empty() && (stack.top().at(0) == '*' ||
stack.top().at(0) == '/') )
                    {
                        topPr =2;
                    }
                }
                stack.push(str.at(i));
            }
        }
        else
        {
            postFix += str.at(i)+" ";
        }
    }
    while( !stack.empty() )
    {
        postFix += stack.top()+" ";
        stack.pop();
    }
    return postFix;
}
```

Get char in hex base from binary string:

```
char binaryToHexChar(string str)
{
    char * ptr;
    long parsed = strtol(str.c_str(), & ptr, 2);
    return hexaChange(parsed).at(0);
}
```

Return object code for WORD Directive:

```
string wordObj(string str)
{
    string temp = hexaChange(atoi( str.c_str()));
```

```
    string zero ="";
    for ( int i =0 ; i < 6-temp.size(); i++ )
    {
        zero += '0';
    }
    return zero+temp;
}
```

Return object code for BYTE Directive:

```
string byteObj( string str )
{
    if( str.at(0) == 'X' )
    {
        vector<string> s = split(str,"'X");
        return s[0];
    }
    else if( str.at(0) == 'C' )
    {
        vector<string> s = split(str,"'C");
        string out = "";
        for ( int i = 0 ; i < s[0].size() ; i++ )
        {
            int c = (int) (s[0].at(i));
            cout << c << endl;
            out += hexaChange(c);
        }
        return out;
    }
}
```

assumptions :

1. ***the user is not allowed to enter the expression with spaces***
2. ***the user is free to enter whatever expression S/he would like***
3. ***the user is free to enter comments where ever S/he would like***
4. ***&&& cannot be used as a label.***

Screen shots :

```
.bubble sort
.temp = t
.23456789012345678901234
bubble   START   1000
         .LTORG
         base    temp
         LDA     temp
         LDA     #0
         LDT     #0
         LDX     #0
         LDS     #0
one      RESW    10
TWO      RESB    100
.
iLOOP    LDX     #0
         STX     =c'asda'
.
jLOOP    LDX     3**+iloop+*
         LDCH    STR+1,X
         .LTORG
         RMO     A,S
THREE    RESW    10
         LDCH    STR,X
         COMPR   A,S
tar      resb    4000
.she      equ     tar
das      equ   tar-jloop
.asda   equ *
d        equ     tar-tar
yy         equ     3*tar-3*tar+100-40
         JGT     JSWAP
         J       BREAK
JSWAP    JSUB    SWAP
.
BREAK    LDX     LNGTH
         LDT     =X'1234'
         SUBR    X,T
         LDX     jjj
         TIXR    T
         STX     jjj
         JLT     jLOOP
.
         LDX     iii
         TIX     LNGTH
         STX     iii
         JLT     iLOOP
         J       HALT
.
SWAP     LDX     jjj
         LDCH    STR,X
         STCH    TEMP
         LDCH    STR+1,X
```

```
            STCH    STR,X
            LDCH    TEMP
            STCH    STR+1,X
            RSUB
HALT        J       *
STR         BYTE    C'3142'
iii         WORD    0
jjj         WORD    0
LNGTH       WORD    4
LNGTH2      WORD    3
TEMP        RESB    1
            END
```

```
Output
#     Address  Label      Mnemonic    Operand                        Comments
               .BUBBLE SORT
               .TEMP = T
               .23456789012345678901 23
1     001000   BUBBLE     START       1000
               .LTORG
2     001000             BASE        TEMP
3     001000             LDA         TEMP
4     001003             LDA         #0
5     001006             LDT         #0
6     001009             LDX         #0
7     00100c             LDS         #0
8     00100f   ONE        RESW        10
9     00102d   TWO        RESB        100
               .
10    001091   ILOOP      LDX         #0
11    001094             STX         =c'asda'
               .
12    001097   JLOOP      LDX         3**+ILOOP+*
13    00109a             LDCH        STR+1,X
               .LTORG
14    00109d             RMO         A,S
15    00109f   THREE      RESW        10
16    0010bd             LDCH        STR,X
17    0010c0             COMPR       A,S
18    0010c2   TAR        RESB        4000
               .SHE      EQU     TAR
19    002062   DAS        EQU         TAR-JLOOP
               .ASDA   EQU *
20    002062   D          EQU         TAR-TAR
21    002062   YY         EQU         3*TAR-3*TAR+100-40
22    002062             JGT         JSWAP
23    002065             J           BREAK
24    002068   JSWAP      JSUB        SWAP
               .
25    00206b   BREAK      LDX         LNGTH
26    00206e             LDT         =X'1234'
27    002071             SUBR        X,T
28    002073             LDX         JJJ
29    002076             TIXR        T
30    002078             STX         JJJ
31    00207b             JLT         JLOOP
               .
32    00207e             LDX         III
33    002081             TIX         LNGTH
34    002084             STX         III
35    002087             JLT         ILOOP
36    00208a             J           HALT
               .
37    00208d   SWAP       LDX         JJJ
38    002090             LDCH        STR,X
```

```
39    002093              STCH        TEMP
40    002096              LDCH        STR+1,X
41    002099              STCH        STR,X
42    00209c              LDCH        TEMP
43    00209f              STCH        STR+1,X
44    0020a2              RSUB
45    0020a5   HALT       J           *
46    0020a8   STR        BYTE        C'3142'
47    0020ac   III        WORD        0
48    0020af   JJJ        WORD        0
49    0020b2   LNGTH      WORD        4
50    0020b5   LNGTH2     WORD        3
51    0020b8   TEMP       RESB        1
52    0020b9              END
      0020B9              =C'ASDA'
      0020BD              =X'1234'
>>  e n d    o f   p a s s   1
>>  ********************************************************************************************
>>  ********************************************************************************************
>>  s y m b o l     t a b l e   (values in Hexa.)
     Name         value       Absol/Reloc
     ---------------------------------
     BREAK        206b        Relocatable
     D            0           Absolute
     DAS          2b          Absolute
     HALT         20a5        Relocatable
     III          20ac        Relocatable
     ILOOP        1091        Relocatable
     JJJ          20af        Relocatable
     JLOOP        1097        Relocatable
     JSWAP        2068        Relocatable
     LNGTH        20b2        Relocatable
     LNGTH2       20b5        Relocatable
     ONE          100f        Relocatable
     STR          20a8        Relocatable
     SWAP         208d        Relocatable
     TAR          10c2        Relocatable
     TEMP         20b8        Relocatable
     THREE        109f        Relocatable
     TWO          102d        Relocatable
     YY           3c          Absolute
>>  ********************************************************************************************
>>  S t a r t   o f    P a s s   I I
>>  A s s e m b l e d    p r o g r a m     l i s t i n g
              .BUBBLE SORT
              .TEMP = T
              .23456789012345678901234567890123
              .LTORG
1     001000  BUBBLE     START       1000
                                     RELC N = 1 I = 1 X = 0 B = 0 P = 0 E = 0
2     001000              BASE        TEMP
```

```
                                    RELC N = 1 I = 1 X = 0 B = 1 P = 0 E = 0
3       001000          LDA         TEMP                        ..... 034000
                                    ABS N = 0 I = 1 X = 0 B = 0 P = 0 E = 0
4       001003          LDA         #0                          ..... 010000
                                    ABS N = 0 I = 1 X = 0 B = 0 P = 0 E = 0
5       001006          LDT         #0                          ..... 750000
                                    ABS N = 0 I = 1 X = 0 B = 0 P = 0 E = 0
6       001009          LDX         #0                          ..... 050000
                                    ABS N = 0 I = 1 X = 0 B = 0 P = 0 E = 0
7       00100c          LDS         #0                          ..... 6D0000

8       00100f  ONE     RESW        10
                .
9       00102d  TWO     RESB        100
                                    ABS N = 0 I = 1 X = 0 B = 0 P = 0 E = 0
10      001091  ILOOP   LDX         #0                          ..... 050000
                .
                                    RELC N = 1 I = 1 X = 0 B = 1 P = 0 E = 0
11      001094          STX         =c'asda'                    ..... 134001
                                    RELC N = 1 I = 1 X = 0 B = 0 P = 0 E = 0
12      001097  JLOOP   LDX         3**+ILOOP+*
                        Error:*** undefined symbol in operand!!
                .LTORG
                                    RELC N = 1 I = 1 X = 1 B = 0 P = 1 E = 0
13      00109a          LDCH        STR+1,X
                        Error:*** Out of boundry!!
                                    RELC N = 1 I = 1 X = 0 B = 0 P = 1 E = 0
14      00109d          RMO         A,S                         ..... AC04

15      00109f  THREE   RESW        10
                                    RELC N = 1 I = 1 X = 1 B = 0 P = 1 E = 0
16      0010bd          LDCH        STR,X
                        Error:*** Out of boundry!!
                                    RELC N = 1 I = 1 X = 0 B = 0 P = 1 E = 0
17      0010c0          COMPR       A,S                         ..... A004
                .SHE    EQU     TAR

18      0010c2  TAR     RESB        4000
                .ASDA   EQU *
                                    ABS N = 1 I = 1 X = 0 B = 0 P = 0 E = 0
19      002062  DAS     EQU         TAR-JLOOP
                                    ABS N = 1 I = 1 X = 0 B = 0 P = 0 E = 0
20      002062  D       EQU         TAR-TAR
                                    ABS N = 1 I = 1 X = 0 B = 0 P = 0 E = 0
21      002062  YY      EQU         3*TAR-3*TAR+100-40
                                    RELC N = 1 I = 1 X = 0 B = 0 P = 1 E = 0
22      002062          JGT         JSWAP                       ..... 372003
                                    RELC N = 1 I = 1 X = 0 B = 0 P = 1 E = 0
23      002065          J           BREAK                       ..... 3F2003
                .
```

```
                .
                                        RELC N = 1 I = 1 X = 0 B = 0 P = 1 E = 0
24      002068  JSWAP   JSUB            SWAP                    ..... 4B2022
                                        RELC N = 1 I = 1 X = 0 B = 0 P = 1 E = 0
25      00206b  BREAK   LDX             LNGTH                   ..... 072044
                                        RELC N = 1 I = 1 X = 0 B = 0 P = 1 E = 0
26      00206e          LDT             =X'1234'                ..... 77204C
                                        RELC N = 1 I = 1 X = 0 B = 0 P = 1 E = 0
27      002071          SUBR            X,T                     ..... 9415
                                        RELC N = 1 I = 1 X = 0 B = 0 P = 1 E = 0
28      002073          LDX             JJJ                     ..... 072039
                                        RELC N = 1 I = 1 X = 0 B = 0 P = 1 E = 0
29      002076          TIXR            T                       ..... B850
                                        RELC N = 1 I = 1 X = 0 B = 0 P = 1 E = 0
30      002078          STX             JJJ                     ..... 132034
                .
                                        RELC N = 1 I = 1 X = 0 B = 0 P = 1 E = 0
31      00207b          JLT             JLOOP
                        Error:*** Out of boundry!!
                                        RELC N = 1 I = 1 X = 0 B = 0 P = 1 E = 0
32      00207e          LDX             III                     ..... 07202B
                                        RELC N = 1 I = 1 X = 0 B = 0 P = 1 E = 0
33      002081          TIX             LNGTH                   ..... 2F202E
                                        RELC N = 1 I = 1 X = 0 B = 0 P = 1 E = 0
34      002084          STX             III                     ..... 132025
                                        RELC N = 1 I = 1 X = 0 B = 0 P = 1 E = 0
35      002087          JLT             ILOOP
                        Error:*** Out of boundry!!
                .
                                        RELC N = 1 I = 1 X = 0 B = 0 P = 1 E = 0
36      00208a          J               HALT                    ..... 3F2018
                                        RELC N = 1 I = 1 X = 0 B = 0 P = 1 E = 0
37      00208d  SWAP    LDX             JJJ                     ..... 07201F
                                        RELC N = 1 I = 1 X = 1 B = 0 P = 1 E = 0
38      002090          LDCH            STR,X                   ..... 53A015
                                        RELC N = 1 I = 1 X = 0 B = 0 P = 1 E = 0
39      002093          STCH            TEMP                    ..... 572022
                                        RELC N = 1 I = 1 X = 1 B = 0 P = 1 E = 0
40      002096          LDCH            STR+1,X                 ..... 53A010
                                        RELC N = 1 I = 1 X = 1 B = 0 P = 1 E = 0
41      002099          STCH            STR,X                   ..... 57A00C
                                        RELC N = 1 I = 1 X = 0 B = 0 P = 1 E = 0
42      00209c          LDCH            TEMP                    ..... 532019
                                        RELC N = 1 I = 1 X = 1 B = 0 P = 1 E = 0
43      00209f          STCH            STR+1,X                 ..... 57A007
                                        RELC N = 1 I = 1 X = 0 B = 0 P = 0 E = 0
44      0020a2          RSUB                                    ..... 4F0000
                                        RELC N = 1 I = 1 X = 0 B = 0 P = 1 E = 0
45      0020a5  HALT    J               *                       ..... 3F2017

46      0020a8  STR     BYTE            C'3142'                 ..... 33313432


46      0020a8  STR     BYTE    C'3142'                 ..... 33313432

47      0020ac  III     WORD    0                       ..... 000000

48      0020af  JJJ     WORD    0                       ..... 000000

49      0020b2  LNGTH   WORD    4                       ..... 000004

50      0020b5  LNGTH2  WORD    3                       ..... 000003

51      0020b8  TEMP    RESB    1
        0020B9          =C'ASDA'
        0020BD          =X'1234'
52      0020b9          END
```

```
H^BUBBLE^001000^0010bf
T^001000^0f^03400001000075000005000060000
T^001091^0b^05000013400153A000AC04
T^0010bd^05^53A000A004
T^002062^1c^3720033F20034B202207204477204C9415072039B8501320343B2000
T^00207e^1e^07202B2F202E1320253B20003F201807201F53A01557202253A01057A00C
T^00209c^1c^53201957A0074F00003F2017333134320000000000000000004000003
E^001000
```

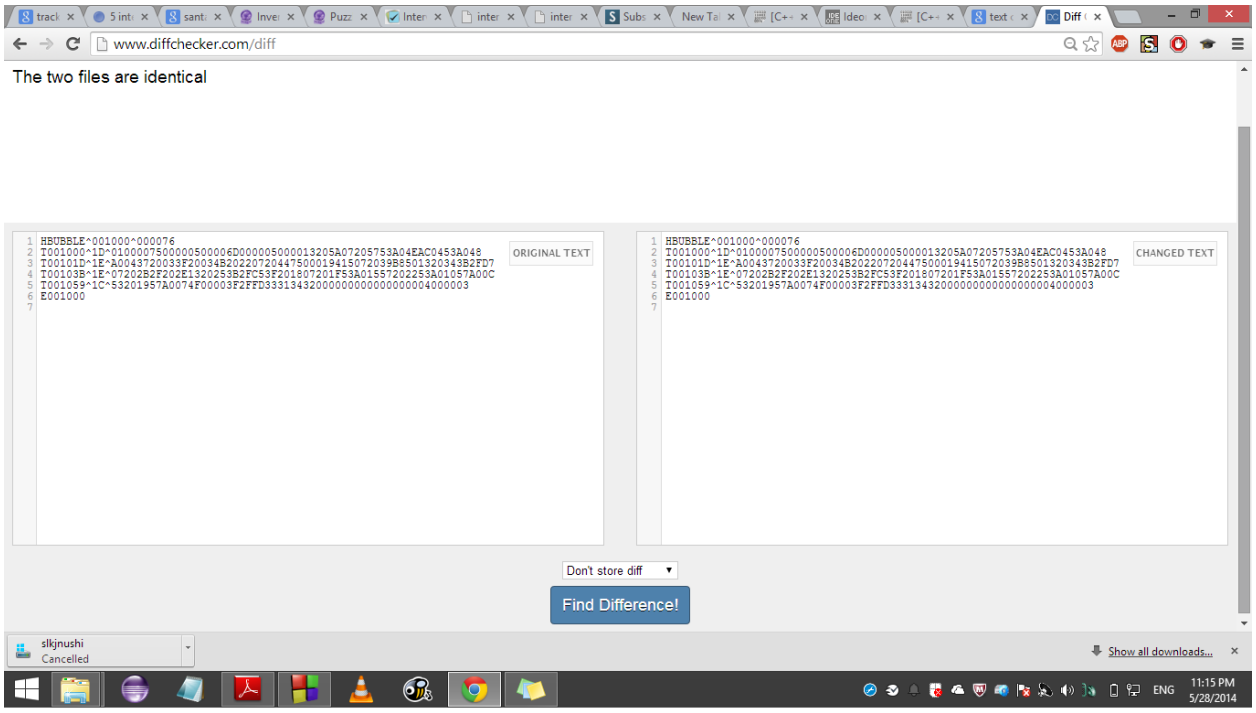
```
            .LTORG
            base    temp
            LDA     temp
            LDAf     #0
            LDT     #0
            LDX     #0
            LDS     #0
```

```
Output
#       Address  Label      Mnemonic    Operand                        Comments
                 .BUBBLE SORT
                 .TEMP = T
                 .23456789012345678901234
1       001000   BUBBLE     START       1000
                 .LTORG
2       001000              BASE        TEMP
3       001000              LDA         TEMP
4       001003   LDAF       #0
                            Error:*** invalid operator !!
5       001003              LDT         #0
6       001006              LDX         #0
7       001009              LDS         #0
8       00100c   ONE        RESW        10
9       00102a   TWO        RESB        100
                 .
10      00108e   ILOOP      LDX         #0
11      001091              STX         =c'asda'
                 .
12      001094   JLOOP      LDX         3**+ILOOP+*
13      001097              LDCH        STR+1,X
                 .LTORG
14      00109a              RMO         A,S
15      00109c   THREE      RESW        10
16      0010ba              LDCH        STR,X
17      0010bd              COMPR       A,S
18      0010bf   TAR        RESB        4000
                 .SHE      EQU     TAR
19      00205f   DAS        EQU         TAR-JLOOP
                 .ASDA   EQU *
20      00205f   D          EQU         TAR-TAR
21      00205f   YY         EQU         3*TAR-3*TAR+100-40
22      00205f              JGT         JSWAP
23      002062              J           BREAK
24      002065   JSWAP      JSUB        SWAP
                 .
25      002068   BREAK      LDX         LNGTH
26      00206b              LDT         =X'1234'
27      00206e              SUBR        X,T
28      002070              LDX         JJJ
29      002073              TIXR        T
30      002075              STX         JJJ
31      002078              JLT         JLOOP
                 .
32      00207b              LDX         III
33      00207e              TIX         LNGTH
34      002081              STX         III
35      002084              JLT         ILOOP
36      002087              J           HALT
                 .
37      00208a   SWAP       LDX         JJJ
```

```
37    00208a   SWAP       LDX         JJJ
38    00208d              LDCH        STR,X
39    002090              STCH        TEMP
40    002093              LDCH        STR+1,X
41    002096              STCH        STR,X
42    002099              LDCH        TEMP
43    00209c              STCH        STR+1,X
44    00209f              RSUB
45    0020a2   HALT       J           *
46    0020a5   STR        BYTE        C'3142'
47    0020a9   III        WORD        0
48    0020ac   JJJ        WORD        0
49    0020af   LNGTH      WORD        4
50    0020b2   LNGTH2     WORD        3
51    0020b5   TEMP       RESB        1
52    0020b6              END
      0020B6              =C'ASDA'
      0020BA              =X'1234'
>>  Incomplete   p a s s   1
>>  ********************************************************************************************************
>>  ********************************************************************************************************
>>  s y m b o l     t a b l e   (values in Hexa.)
     Name        value       Absol/Reloc
     ---------------------------------
     BREAK       2068        Relocatable
     D           0           Absolute
     DAS         2b          Absolute
     HALT        20a2        Relocatable
     III         20a9        Relocatable
     ILOOP       108e        Relocatable
     JJJ         20ac        Relocatable
     JLOOP       1094        Relocatable
     JSWAP       2065        Relocatable
     LDAF        1003        Relocatable
     LNGTH       20af        Relocatable
     LNGTH2      20b2        Relocatable
     ONE         100c        Relocatable
     STR         20a5        Relocatable
     SWAP        208a        Relocatable
     TAR         10bf        Relocatable
     TEMP        20b5        Relocatable
     THREE       109c        Relocatable
     TWO         102a        Relocatable
     YY          3c          Absolute
```

```
                        J       HALT
                .
SWAP            LDX     jjj
                LDCH    STR,X
                STCH    TEMhP
                LDCH    STR+1,X
                STCH    STR,X
                LDCH    TEMP
                STCH    STR+1,X
                RSUB

                        RELC N = 1 I = 1 X = 0 B = 0 P = 0 E = 0
002093          STCH    TEMHP
Error:*** undefined symbol in operand!!
                        RELC N = 1 I = 1 X = 1 B = 0 P = 1 E = 0
002096          LDCH    STR+1,X                 ..... 53A010
                        RELC N = 1 I = 1 X = 1 B = 0 P = 1 E = 0
002099          STCH    STR,X                   ..... 57A00C
                        RELC N = 1 I = 1 X = 0 B = 0 P = 1 E = 0
```


www.diffchecker.com/diff
The two files are identical
ORIGINAL TEXT
HBUBBLE^001000^000076
T001000^1D^01000075000005000006D000005000013205A07205753A04EAC0453A048
T00101D^1E^A0043720033F20034B2022072044750001941507203 9B8501320343B2FD7
T00103B^1E^07202B2F202E1320253B2FC53F2018072015 3A01557202253A01057A00C
T001059^1C^S3201957A0074F00003F2FFD3331343200000000000000004000003
E001000
CHANGED TEXT
HBUBBLE^001000^000076
T001000^1D^01000075000005000006D000005000013205A07205753A04EAC0453A048
T00101D^1E^A0043720033F20034B2022072044750001941507203 9B8501320343B2FD7
T00103B^1E^07202B2F202E1320253B2FC53F2018072015 3A01557202253A01057A00C
T001059^1C^S3201957A0074F00003F2FFD3331343200000000000000004000003
E001000
Don't store diff
Find Difference!
slkjnushi
Cancelled
Show all downloads...
ENG
11:15 PM
5/28/2014