Assembler Project

Phase I

Team members:

- Bassem Mohamed # 13
- Hassan Abdo # 17
- Abdelrhman ALi # 34
- Abdelrhman moustfa # 36
- Mohamed Hassan # 57

Problem Statement :-

The term project is to implement a (cross) assembler for (a subset of) SIC/XE assembler, written in **C/C++,** producing code for the absolute loader used in the SIC programming assignments.
In phase 2 of the project, you are going to build on the previous phase and use its output to implement pass 2 of the assembler.

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 :-

In Pass1 Report, We illustrated the Design of pass1, We continued work on pass1 and added some new classes to support pass2.
The new Classes :-

- ObjectCode.cpp : this class in responsible about writing the objectFile and generating the object code of each instruction and evaluating the appropriate address as PC relative or absolute.
- Pass2.cpp : makes the second pass over the source code and by the help of ObjectCode class and symTable and litTable created during pass1 , object code of each instruction could be obtained. Pass2 writes the second part of list file.
- Postfix.cpp : While iterating over the infix expression, when encountering
  Any Variable >> add to postfix expression
  Left Parenthesis >> push onto the stack and remove only when the right parenthesis is found
  Right Parenthesis >> remove symbol from stack to postfix expression till incounter a left parenthesis then discard it.
  Operator >> it's precedence is surely equal to that in the top of the stack since we only have + or -
  then pop the element on the top of the stack to the post fix expression till the stack is empty
  then push the operator in.
  End of Expression >> Pop all the stack to the post fix expression

Main DateStructures :-

1- Map : used in storing operation table.
2- Unordered_map : used in storing symTable and litTable.
3- Vector : used to store the instruction after dividing it using regex.

Algorithms Description :-

1- After pass1 finishes , the output.txt file contains the instructions and their location counters, SymTable contains all label with their locations ,LitTable contains all literals with their locations.
2- Pass2 Constructor initializes a new parser on the output.txt file to get each instruction with its location counter , and initializes a new ObjectCode object to use it in generating object codes.
3- The parser returns each instruction until end of file and ObjectCode objects calls addRecord() method to get the object code of each instruction.
4- After above steps , pass2 has all the information about the instruction to write it list file.
5- The above steps are repeated until the parser return null(EOF).
6- Evaluating Postfix

   1- Scan postfix expression from left to right
   2- Let the 1st element of the postfix expression be X ,Push X to the stack
   3- Let the second element be Y, Push Y to the stack.
   4- The third element must be an operator. let it be ()
   5- Evaluate Z = X () Y
   6- Push Z to the stack
   7-Continue till the end of the post fix expression

(A)bsolute(0) or (R)elative(1) or (I)nvalid(2)

	(+)	(-)
0 0	A	A
0 1	R	I
0 2	I	I
1 0	R	R
1 1	I	A
1 2	I	I
2 0	I	I
2 1	I	I
2 2	I	I

Regex is used for the ease of extracting operands.

Sample Runs :-

Test 1:

```
>>  Source Program statements with value of LC indicated


000000    .234567890123456789012345678901234567890
000000              START   0000
000000    L1        RESW    1
000003    L2        RESW    2
000009    L3        EQU     L2-L1+1
000009    L4        EQU     *+L3+1
000009    L5        EQU     (*+(L2-L1))+3
000009              J       *
00000C              END
>>   ********************************************************


>>    s y m b o l     t a b l e   (values in hexa)


      Symbol Hexa Value     Addressing type
      ---------------------------------------
      L1     000000         Relocatable
      L4     00000E         Relocatable
      L5     00000F         Relocatable
      L2     000003         Relocatable
      L3     000004           Absolute
>>   ********************************************************


>>    e n d    o f   p a s s   1
```

```
>>>>   ************************************************************
>>   S t a r t   o f   P a s s   I I

lcCntr objCde    label opration operand


000000         L1       RESW    1


000003         L2       RESW    2


000009         L3       EQU     L2-L1+1


000009         L4       EQU     *+L3+1


000009         L5       EQU     (*+(L2-L1))+3


                                Rlc n=1 i=1 x=0 b=0 p=1 e=0
000009 3F2FFD           J       *


00000C                  END
```

Object Code

```
File  Edit  Format  View  Help
T^000009^03^3F2FFD
E^000000
```

Test 2:

```
>>  Source Program statements with value of LC indicated

000000    .23456789012345678901234567890
001000    PROB2    START   1000
001000             LDA     #0
001003             LDX     #0
001006    AGAIN    TD      DEVF3
001009             JEQ     AGAIN
00100C             RD      DEVF3
00100F             SUB     #48
001012             MUL     #10
001015             RMO     A,X
001017    LOOOP    TD      DEVF3
00101A             JEQ     LOOOP
00101D             RD      DEVF3
001020             SUB     #48
001023             ADDR    X,A
001025             J       *
001028    DEVF3    BYTE    X'F3'
001029             END
>>   ************************************************************

>>    s y m b o l     t a b l e   (values in hexa)

       Symbol    Hexa Value     Addressing type
       ----------------------------------------
       AGAIN     001006         Relocatable
       LOOOP     001017         Relocatable
       DEVF3     001028         Relocatable
>>   ************************************************************

>>    e n d    o f    p a s s   1
```

```
>>>>   ********************************************************
>>   S t a r t   o f   P a s s   I I

lcCntr objCde    label opration operand

                                Abs n=0 i=1 x=0 b=0 p=0 e=0
001000 010000           LDA     #0

                                Abs n=0 i=1 x=0 b=0 p=0 e=0
001003 050000           LDX     #0

                                Rlc n=1 i=1 x=0 b=0 p=1 e=0
001006 E3201F AGAIN     TD      DEVF3

                                Rlc n=1 i=1 x=0 b=0 p=1 e=0
001009 332FFA           JEQ     AGAIN

                                Rlc n=1 i=1 x=0 b=0 p=1 e=0
00100C DB2019           RD      DEVF3

                                Abs n=0 i=1 x=0 b=0 p=0 e=0
00100F 1D0030           SUB     #48

                                Abs n=0 i=1 x=0 b=0 p=0 e=0
001012 21000A           MUL     #10

001015 AC01           RMO     A,X

                                Rlc n=1 i=1 x=0 b=0 p=1 e=0
001017 E3200E LOOOP     TD      DEVF3

                                Rlc n=1 i=1 x=0 b=0 p=1 e=0
00101A 332FFA           JEQ     LOOOP

                                Rlc n=1 i=1 x=0 b=0 p=1 e=0
00101D DB2008           RD      DEVF3
```

```
                                Rlc n=1 i=1 x=0 b=0 p=1 e=0
00101A 332FFA           JEQ     LOOOP

                                Rlc n=1 i=1 x=0 b=0 p=1 e=0
00101D DB2008           RD      DEVF3

                                Abs n=0 i=1 x=0 b=0 p=0 e=0
001020 1D0030           SUB     #48

001023 9010           ADDR    X,A

                                Rlc n=1 i=1 x=0 b=0 p=1 e=0
001025 3F2FFD           J       *

001028 F3 DEVF3     BYTE    X'F3'

001029                  END
```

```
T^001000^1D^010000^050000^E3201F^332FFA^DB2019^1D0030^21000A^AC01^E3200E^332FFA
T^00101D^0C^DB2008^1D0030^9010^3F2FFD^F3
E^000000
```

Test3 :

```
>>  Source Program statements with value of LC indicated

000000    .234567890123456789012345678901234567890
001000              START   1000
001000    ALPHA     LDT     #10
001003              lps     =C'dfe'
001006    BETA     +LDCH    =X'12'
00100A              LTORG
00100A                      =C'dfe'
00100D                      =X'12'
00100E              WD      =C'd'
001011    . INDEXING
001011              ADD     GAMMA,X
001014              J       *
001017    GAMMA     BYTE    X'01'
001018    THETA     RESB    2
00101A              END     ALPHA
00101A                      =C'd'
>>   ******************************************************

>>    s y m b o l     t a b l e   (values in hexa)

       Symbol    Hexa Value     Addressing type
       ----------------------------------------
       ALPHA     001000         Relocatable
       BETA      001006         Relocatable
       GAMMA     001017         Relocatable
       THETA     001018         Relocatable
>>   ******************************************************

>>    e n d    o f   p a s s   1

>>>>   ******************************************************
>>   S t a r t   o f    P a s s   I I

lcCntr objCde    label opration operand

                                Abs n=0 i=1 x=0 b=0 p=0 e=0
001000 75000A ALPHA    LDT      #10

                                Rlc n=1 i=1 x=0 b=0 p=1 e=0
001003 D32004          lps      =C'dfe'
```

```
00100A 646665       =C'dfe'

00100D 12       =X'12'

                                Rlc n=1 i=1 x=0 b=0 p=1 e=0
00100E DF2009           WD      =C'd'

                                Rlc n=1 i=1 x=1 b=0 p=1 e=0
001011 1BA003           ADD     GAMMA,X

                                Rlc n=1 i=1 x=0 b=0 p=1 e=0
001014 3F2FFD           J       *

001017 01 GAMMA     BYTE    X'01'

001018          THETA     RESB    2

00101A 64       =C'd'

00101A                  END     ALPHA
```

```
T^001000^18^75000A^D32004^5310100D^646665^12^DF2009^1BA003^3F2FFD^01
T^00101A^01^64
M^001007^05
E^000000
```