Alexandria University,

Faculty of engineering,

Systems programming.

Assembler project.

Names:

- Mohamed Mohamed Abdlhakem (43).
- Mohamed Salah Osman (41).
- Ahmed Nabil mohamed Anwar (7).
- Yehia Elsayed mohamed Mohamed (59).

Requirement specifications:

1) The assembler is to execute by entering: assemble <source-file-name>.
2) The source file for the main program for this phase is to be named assemble.cpp.
3) The output of the assembler should include (at least):
    i) 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.
    ii) A report at the end of pass2. Pass1 and Pass2 errors should be included as part of the assembler report, exhibiting both the erroneous lines of source code and the error.
4) The assembler should support:
    i) EQU and ORG statements.
    ii) Simple expression evaluation. A simple expression includes simple (A <op> B) operand arithmetic, where <op> is one of +, -, *, / and no spaces surround the operation: A+B.

Design:

- Ob.cpp Class is a class that holds the mnemonic, its format and its object code.
- Objectcodemap.cpp maps all the supported mnemonics to its object code and format.
- Class parser.cpp parses the input file lines and check for the syntax and the matching between mnemonics and operands.
- Class readfile.cpp reads the file input file and the instruction builder holds an instance of read file as well as write file class.
- Instruction.cpp is the class that holds all the components of the instruction line (label, mnemonic and operands).
- Pass1.cpp is class that passes over the instruction of the input and generates the symbol table, handle literals and directives.
- WriteFile.cpp is class that write the output file.
- Print.cpp is class that write the object file that contains text records.
- Pass2.cpp is the main class for pass2 that iterates the instructions and calculates the object code for every instruction.
- AddressTranslation.cpp class translates every instruction to the corresponding object code.

Main data structures:

- Unordered map :
    1. An instance holds the object code and format for every mnemonic.
    2. An instance holds the symbol table.
    3. An instance holds the literals.
- Vector:
    1. An instance holds the instruction lines.
    2. An instance holds the text records.
    3. An instance holds the operations.

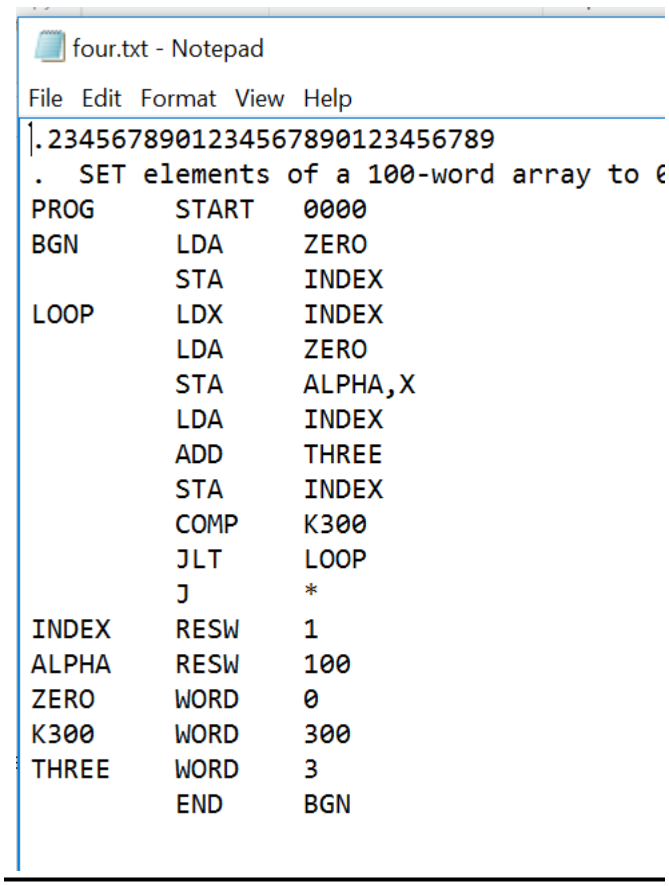
Algorithms description:

- Most of the project is built on hashing as we pass data to the map and retrieves it from map in O (1).
- Some algorithms of data conversions from type to type.
- The main algorithm of pass 1 is the for-loop that iterates the instruction lines and checks for all the syntax and if literal found it handles it as well as the expressions.it checks for the matching between mnemonics and operands.
- The main algorithm of pass 2 is the for-loop that iterates the instruction lines and calculates the object code for every instruction and generates text records.

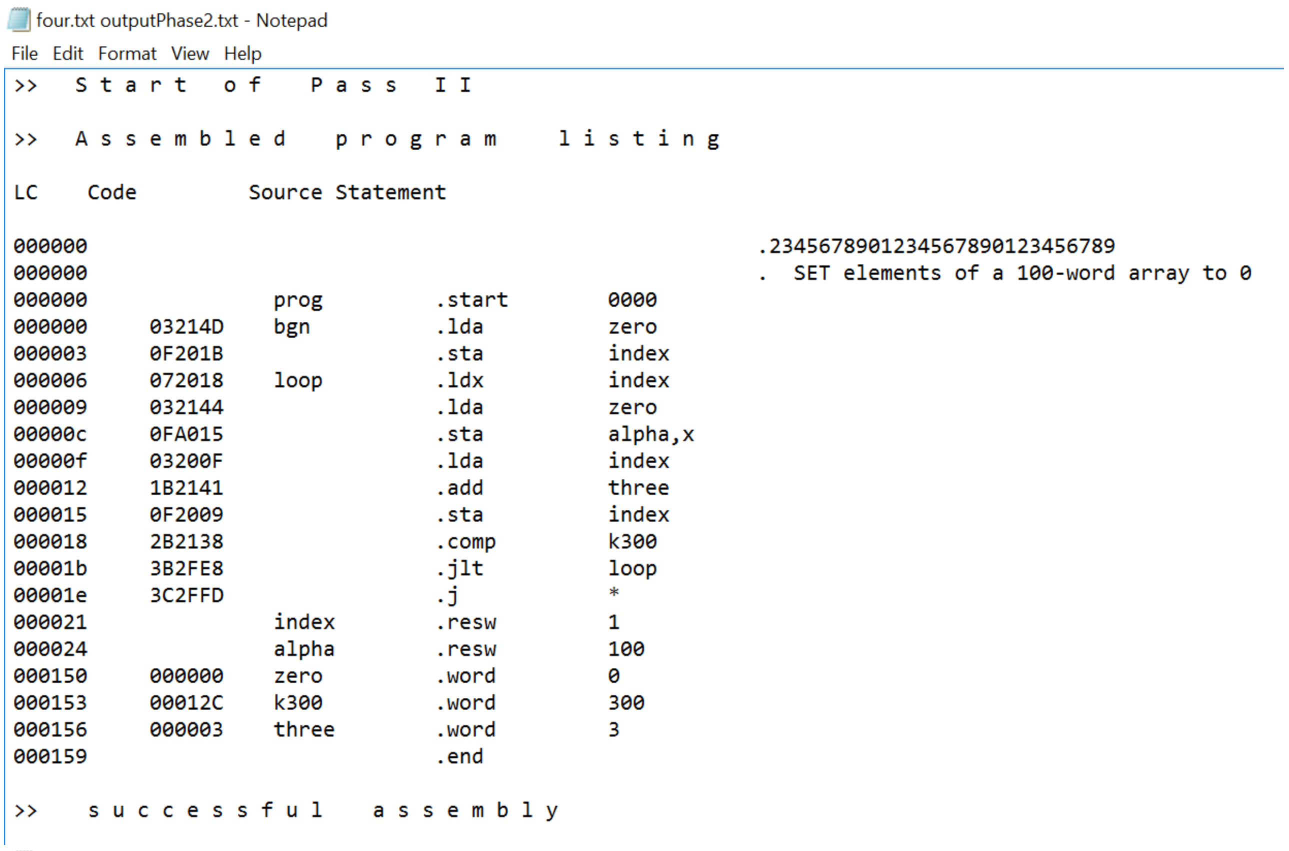
Bonus Features:

- The assembler supports general expressions in operand field.
- Literals are also supported.
- The assembler provides free-formatted statements.

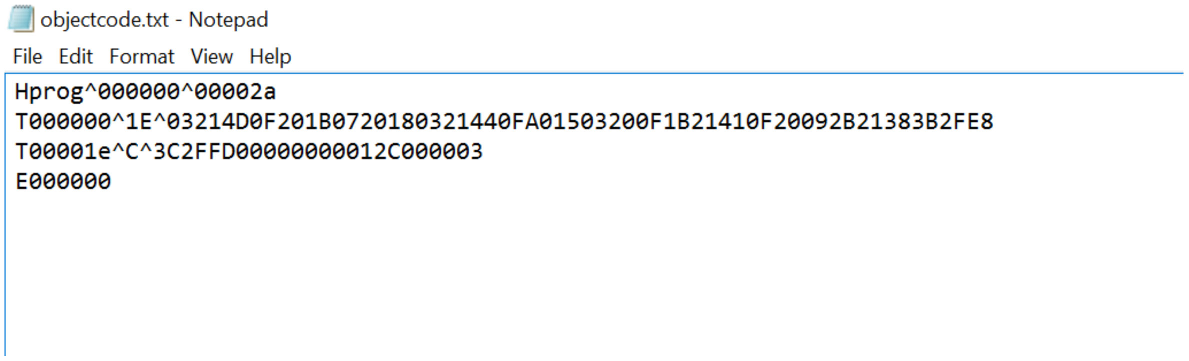
Sample runs:



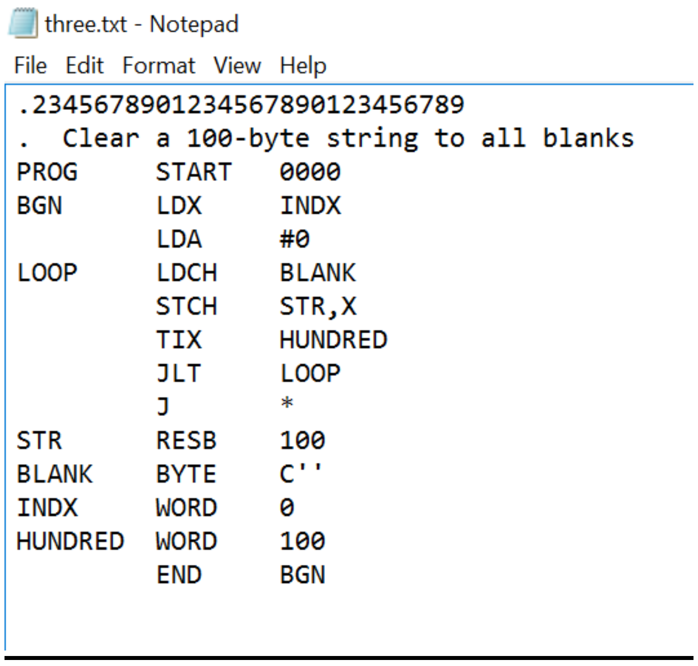
```
.23456789012345678901234567890123456789
.  SET elements of a 100-word array to 
PROG      START    0000
BGN       LDA      ZERO
          STA      INDEX
LOOP      LDX      INDEX
          LDA      ZERO
          STA      ALPHA,X
          LDA      INDEX
          ADD      THREE
          STA      INDEX
          COMP     K300
          JLT      LOOP
          J        *
INDEX     RESW     1
ALPHA     RESW     100
ZERO      WORD     0
K300      WORD     300
THREE     WORD     3
          END      BGN
```


four.txt outputPhase2.txt - Notepad
File Edit Format View Help

```
>>   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

LC     Code         Source Statement

000000                                                       .23456789012345678901234567890
000000                                                       .  SET elements of a 100-word array to 0
000000              prog         .start       0000
000000     03214D   bgn          .lda         zero
000003     0F201B                .sta         index
000006     072018   loop         .ldx         index
000009     032144                .lda         zero
00000c     0FA015                .sta         alpha,x
00000f     03200F                .lda         index
000012     1B2141                .add         three
000015     0F2009                .sta         index
000018     2B2138                .comp        k300
00001b     3B2FE8                .jlt         loop
00001e     3C2FFD                .j           *
000021              index        .resw        1
000024              alpha        .resw        100
000150     000000   zero         .word        0
000153     00012C   k300         .word        300
000156     000003   three        .word        3
000159                           .end

>>    s u c c e s s f u l   a s s e m b l y
```

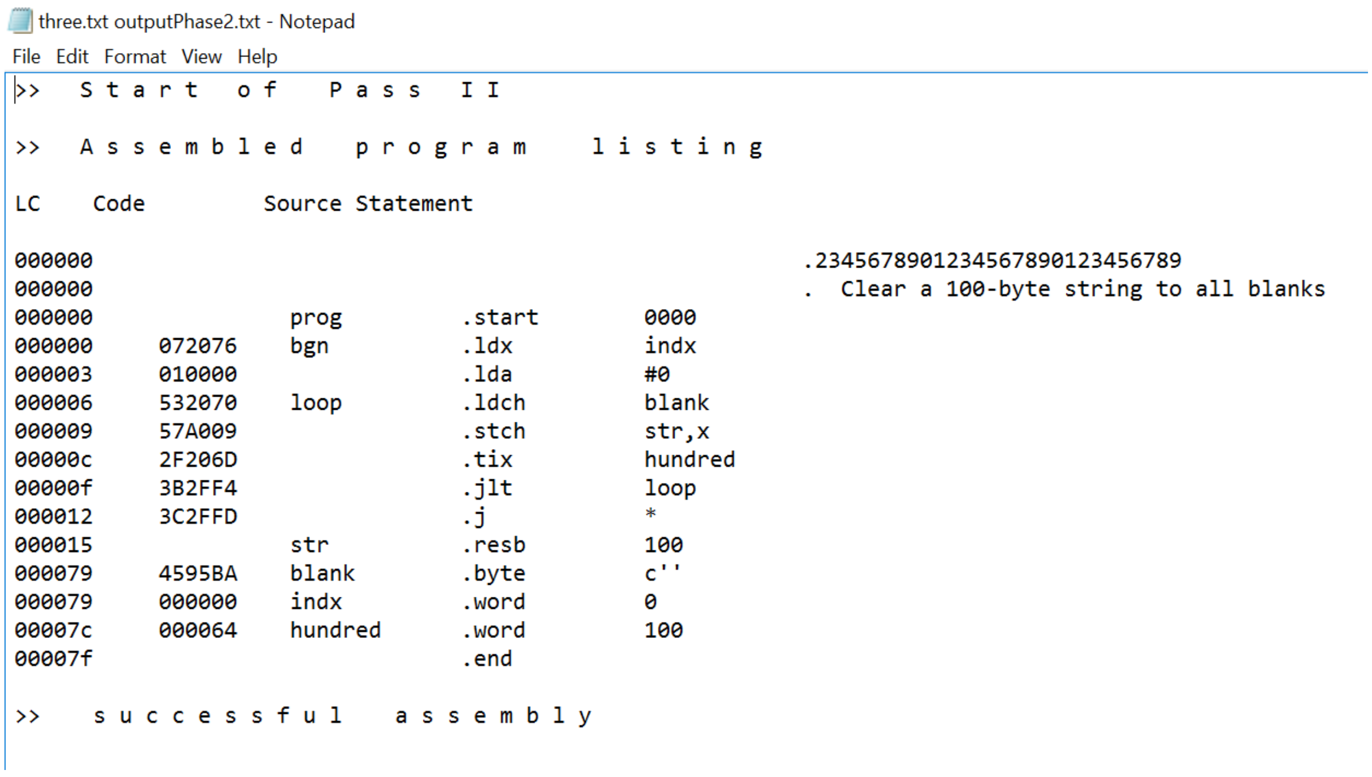


```
Hprog^000000^00002a
T000000^1E^03214D0F201B0720180321440FA01503200F1B21410F20092B21383B2FE8
T00001e^C^3C2FFD00000000012C000003
E000000
```


three.txt - Notepad

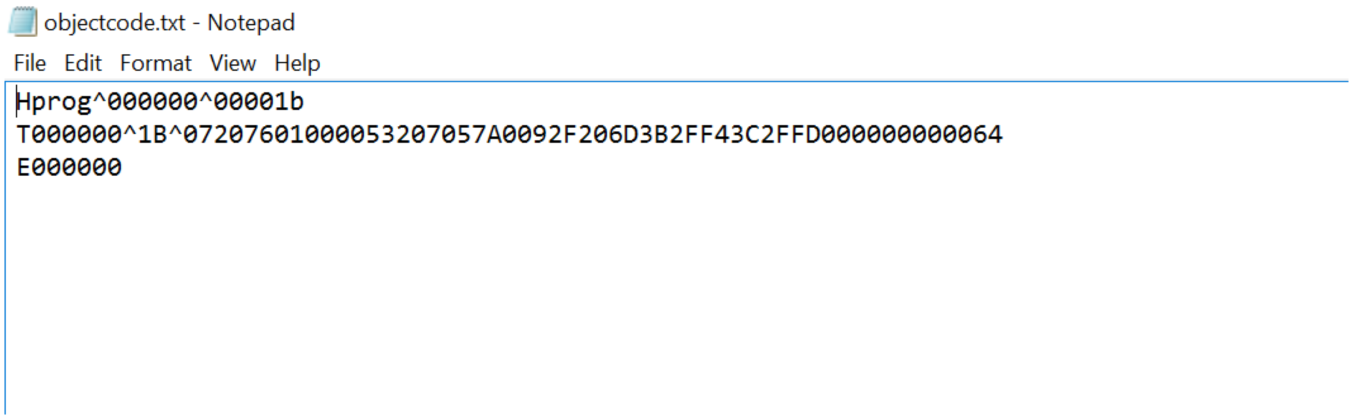
File Edit Format View Help

```
.2345678901234567890123456789
.  Clear a 100-byte string to all blanks
PROG     START   0000
BGN      LDX     INDX
         LDA     #0
LOOP     LDCH    BLANK
         STCH    STR,X
         TIX     HUNDRED
         JLT     LOOP
         J       *
STR      RESB    100
BLANK    BYTE    C' '
INDX     WORD    0
HUNDRED  WORD    100
         END     BGN
```


three.txt outputPhase2.txt - Notepad

File Edit Format View Help

```
>>   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

LC     Code          Source Statement

000000                                                            .23456789012345678901234567890
000000                                                            .  Clear a 100-byte string to all blanks
000000               prog         .start        0000
000000     072076    bgn          .ldx          indx
000003     010000                 .lda          #0
000006     532070    loop         .ldch         blank
000009     57A009                 .stch         str,x
00000c     2F206D                 .tix          hundred
00000f     3B2FF4                 .jlt          loop
000012     3C2FFD                 .j            *
000015               str          .resb         100
000079     4595BA    blank        .byte         c''
000079     000000    indx         .word         0
00007c     000064    hundred      .word         100
00007f                            .end

>>    s u c c e s s f u l    a s s e m b l y
```


objectcode.txt - Notepad

File Edit Format View Help

```
Hprog^000000^00001b
T000000^1B^072076010000532070 57A0092F206D3B2FF43C2FFD000000000064
E000000
```