University *of* Alexandria
Faculty of Engineering
Computer and Systems Engineering Department

Term Project - SIC/XE Assembler
Phase (2)

Ahmed Elsayed Mahmoud (05)

Ahmed Eid Abdelmon'em (09)

Shehab Kamal El-samany (32)

Mohamed Ahmed Abd-ElTwab (52)

Mohamed Samir Shaaban (56)

Requirements specification:

- 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/XE programming assignments.

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

The assembler is to execute by entering assemble *<source-file-name>*

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

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.

EQU and ORG statements.

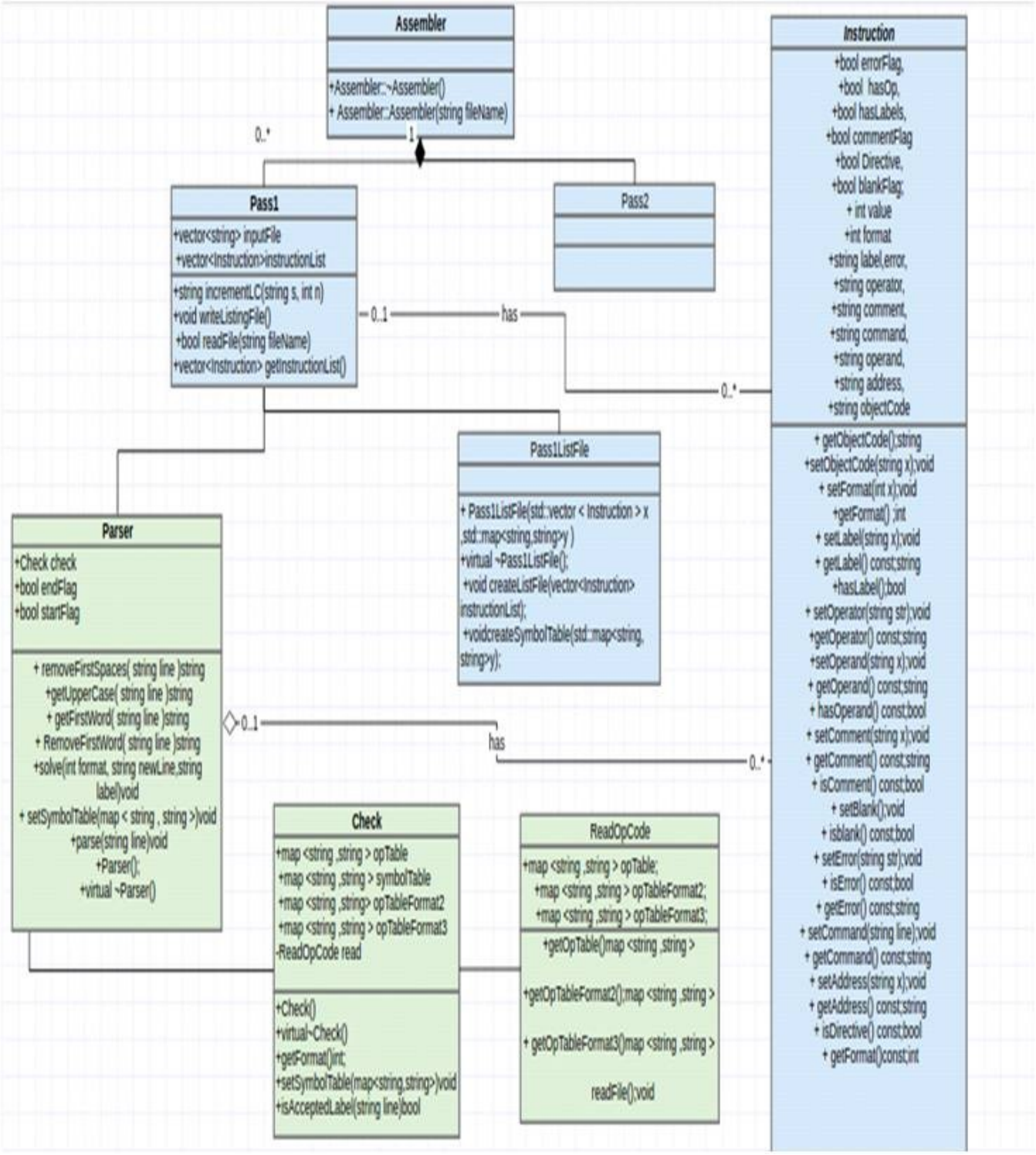
Simple expression evaluation. A simple expression includes simple (A<op> B) operand arithmetic, where <op> is one of +,-,*,/ and no spaces surround the operation, eg. A+B.

General expression evaluation.

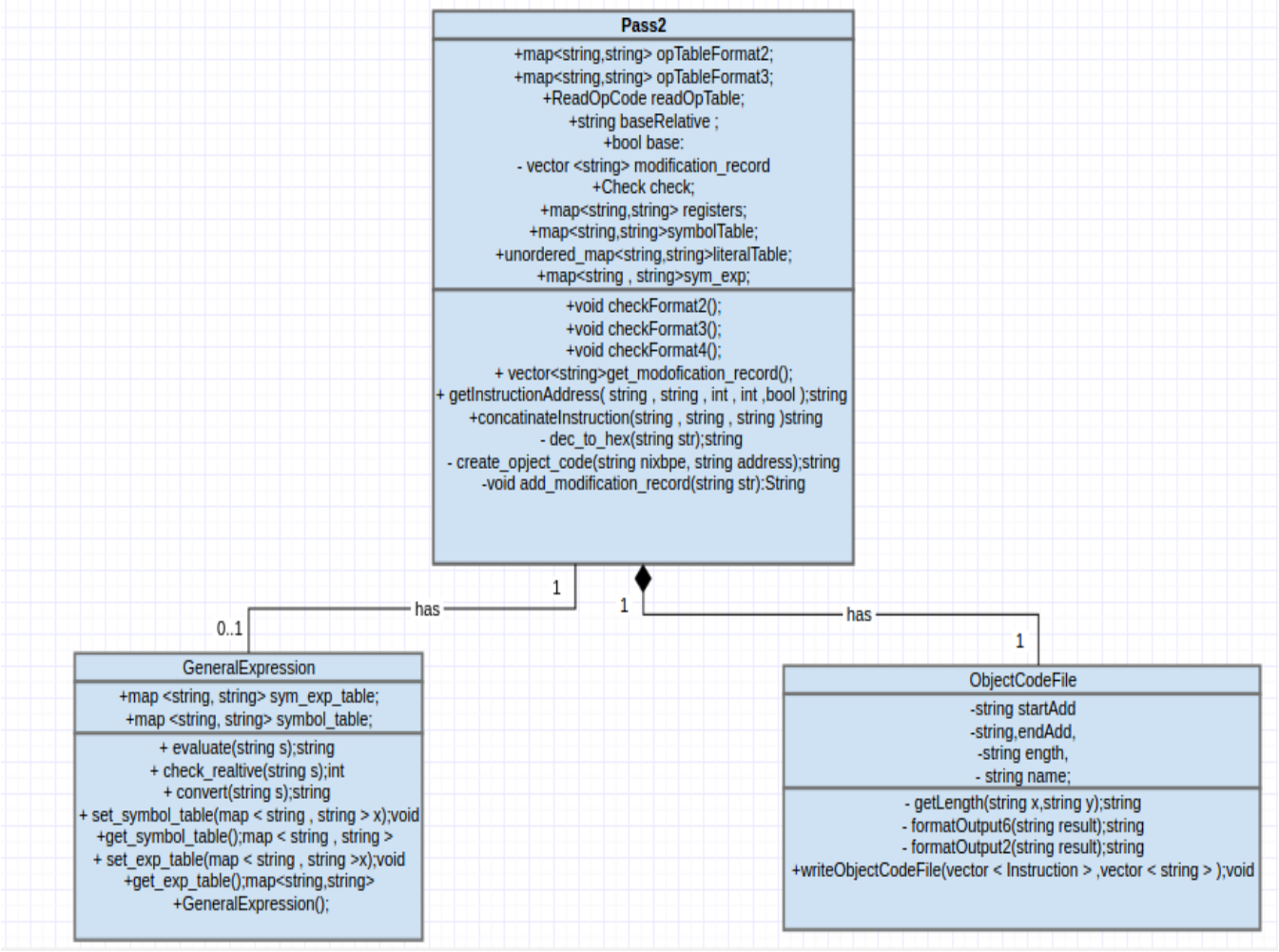
Literals (Including LTORG) =C'<ASCII-TEXT>' , =X'HEX-TEXT' , =<DECIMAL-TEXT> forms.

Design:

Phase 1 :



Phase 2 :


Pass2
+map<string,string> opTableFormat2;
+map<string,string> opTableFormat3;
+ReadOpCode readOpTable;
+string baseRelative ;
+bool base:
- vector <string> modification_record
+Check check;
+map<string,string> registers;
+map<string,string>symbolTable;
+unordered_map<string,string>literalTable;
+map<string , string>sym_exp;
+void checkFormat2();
+void checkFormat3();
+void checkFormat4();
+ vector<string>get_modofication_record();
+ getInstructionAddress(string , string , int , int ,bool);string
+concatinateInstruction(string , string , string)string
- dec_to_hex(string str);string
- create_opject_code(string nixbpe, string address);string
-void add_modification_record(string str):String
1
has
0..1
1
has
1
GeneralExpression
+map <string, string> sym_exp_table;
+map <string, string> symbol_table;
+ evaluate(string s);string
+ check_realtive(string s);int
+ convert(string s);string
+ set_symbol_table(map < string , string > x);void
+get_symbol_table();map < string , string >
+ set_exp_table(map < string , string >x);void
+get_exp_table();map<string,string>
+GeneralExpression();
ObjectCodeFile
-string startAdd
-string,endAdd,
-string ength,
- string name;
- getLength(string x,string y);string
- formatOutput6(string result);string
- formatOutput2(string result);string
+writeObjectCodeFile(vector < Instruction > ,vector < string >);void

Main data structures:

- map:

  - `map` is a sorted associative container that contains key-value pairs with unique keys. Keys are sorted by using the comparison function `Compare`. Search, removal, and insertion operations have logarithmic complexity. Maps are usually implemented as red-black tree.

  - we use it to store the op-Table , get the operator foramt , get operator op-code ,to store type of expression .

- vector:

  - `vector` is a sequence container that encapsulates dynamic size arrays.

  - why we use it ?

    - Random access - constant *O(1)*
    - Insertion or removal of elements at the end - amortized constant *O(1).*
    - Insertion or removal of elements - linear in distance to the end of the vector *O(n).*

  - we use it to store each instruction in it , to store modification record ,Store object code for each instruction .

Unordered Map :

-get the value to corresponding key in order (1) .

-Using in literal table .

Algorithms description:

-validateInstruction(vector<Instruction>instructionList):

To check validation of all Instructions and separate them to corresponding validate format (format **2 ,3,4 ,** directive).

-CheckFormat2():

To check validation of instructions of format 2 and generate its corresponding object code .

-checkFormat3():

To check validation of instructions of format 3 and generate its corresponding object code .

-checkFormat4():

To check validation of instructions of format 4 and generate its corresponding object code .

-dec_to_hex(string str):

To convert from decimal value to it's corresponding hex value .use it to calculate the displacement to generate object code .

-add_modification_record(string str):

add modification record to vector of modification records .

-create_object_code(string nixbpe, string address):

To create object code to instructions of format 4 .

-getInstructionAddress(string pc , string add , int flag, int format,bool isHex):

To get the displacement of instruction (last 3 bit in object code) .

-get_modofication_record():

Return modification record vector , use it to print in object code file .

-concatinateInstruction(string opCode , string nixbpe , string address):

Concatenate between first 3 bits and last 3 bits in object code to generate the object code .

- GetInstructionList():

Return Instructions list after adding its object code to print object code file .

writeObjectCodeFile(std::vector<Instruction>x,std::vector < string > modificationRecords):

Write object code file after validate its format .

- getLength(string x,string y):

Get the length of the code.

- formatOutput6(string result):

To update the format of 6 digits to print it in object code file .

- formatOutput2(string result):

To update the format of 2 digits to print it in object code file .

-void insertIntoLiteralTable(string operands) :

To insert into literal table .

- unordered_map < string , string > getLiteralTable()

return literal table .

void setLiteralTable(unordered_map<string , string > literal):

To set literal table .

- void insertAddressIntoLiteralTable()

to set address for each literal after LTORG or END .

-string evaluate(string s) :

calculate the expression using postfix algorithm .

-int check_realtive(string s) :

check type of expression return 0 in case of absolute , 1 in case of relative , otherwise return -1 .

-string convert(string s)

Check validation of expression usnig symbol table and get the address for each variable .

Assumptions :

- default start address equal zero in case not mentioned in the code.
- free format is handled.
- no comment is supported on the same line of code.
- code is case insensitive.
- line which has an error has won't be saved in memory.
- The operand for End instruction must be the same for Start Instruction .
- Base relative is handled .
- Literals are handled .
- General Expression is handled .
- Expressions don't have braces .
- No object code is case of corrupted code.
- The location for instructions which contain * have location for same instruction .
- Any number in operand field considered decimal .
- literals is defined with a name of it's value .
- in case of =* it's names with the *+address of instruction
- C'EOF' is different from x'454f46'

- For format 4 can use immediate , indexing and direct addressing

Sample Runs :

```
 1 copy start 0
 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'EOF'
18 retadr resw 1
19 length resw 1
20 buffer resb 4096
21 rdrec clear x
22 clear a
23 clear s
24 +ldt #4096
```

```
27 rloop td input
28 jeq rloop
29 rd input
30 compr a,s
31 jeq exit
32 stch buffer,x
33 tixr t
34 jlt rloop
35 exit stx length
36 rsub
37 input byte x'F1'
38 wrrec clear x
39 ldt length
40 wloop  td output
41 jeq wloop
42 ldch buffer,x
43 wd output
44 tixr t
45 jlt wloop
46 rsub
47 output byte x'05'
48 end copy
```

```
Object Code :
--------------
--------------


H^COPY  ^000000^001077
T^000000^1d^17202d^69202d^4b100000^032026^290000^332007^4b10105d^3f2fec^032010^
T^00001d^13^0f2016^010003^0f200d^4b10105d^3e2003^454f46^
T^001036^1d^B410^B400^B440^75101000^e32019^332ffa^db2013^A004^332008^57c003^B850^
T^001053^1d^3b2fea^134000^4F0000^F1^B410^774000^e32011^332ffa^53c003^df2008^B850^
T^001070^07^3b2fef^4F0000^05^
M00000705
M00001405
M00002705
M00002b05
M00103d05
E^000000
```

Sample Run 2 :

```
PROB1    START   0
         LDS     #0
         LDT     #10
TESTDEV  TD      INDEV
         JEQ     TESTDEV
         LDA     #0
         RD      INDEV
         COMP    EOF
         JEQ     EXIT
         SUB     #48

.Test lower bound
LCHECK   COMP    #0
         JEQ     UCHECK
         JGT     UCHECK
         J       EXIT

.Test upper bound
UCHECK   COMP    #10
         JLT     VNUMBER
         J       EXIT

.valid number
VNUMBER  MULR    T,S
         ADDR    A,S
         J       TESTDEV
EXIT     RMO     S,A
         J       *
.******Assembler Directives****
INDEV    BYTE    X'F3'
EOF      WORD    4
         END
```

```
Object Code :
--------------
--------------

H^PROB1 ^000000^000040
T^000000^1e^6d0000^75000a^e32033^332ffa^010000^db202a^2b2028^33201f^1d0030^290000^
T^00001e^1e^332006^372003^3f2010^29000a^3b2003^3f2007^9854^9004^3f2fcf^AC40^3f0039^
T^00003c^04^F3^000004^
M00003a05
E^000000
```

Sample run 3 :

```
1  .234567890123456789123
2  PROB3    START   0
3           LDA     #0
4  . READ TARGET CHARCACTER FROM DEVICE F2 .
5  TESTDEV2 TD      READCHAR
6           JEQ     TESTDEV2
7           RD                READCHAR
8           STA             TARGET
9  . READ STRING FROM DEVICE F3 .
10          LDX     #0
11 TESTDEV3 TD      READSTR
12          JEQ     TESTDEV3
13                LDA     #0
14          RD              READSTR
15          STA     STRING,X
16                  COMP    TARGET
17          JEQ     FOUND
18          COMP    EOT
19          JEQ     EXIT
20          TIX     #256
21                      J       TESTDEV3
22 FOUND    LDT     #STRING
23          ADDR    X,T
24 EXIT     RMO     T,A
25          J       *
26 READCHAR     BYTE    X'F2'
27 READSTR  BYTE    X'F3'
28                  TARGET  RESW    1
29 STRING   RESB    100
30 EOT      WORD    4
31          END
```

```
Object Code :
--------------
--------------
H^PROB3 ^000000^0000a9
T^000000^1e^010000^e32037^332ffa^db2031^0f2030^050000^e32029^332ffa^010000^db2020^
T^00001e^1c^0fa021^2b201b^33200c^2b207c^33200b^2d0100^3f2fdf^75200c^9015^AC50^
T^00003a^05^3f003a^F2^F3^
T^0000a6^03^000004^
M00003b05
E^000000
```