Systems and Components

Term Project.

Phase 2.

Group Names :

1 – Mohamed Ahmed Aboelhassan. #54

2 – Aly Tarek Ibrahim. #39

3 – Shady Raafat Elbaroudy. #26

4 – Tarek Ashraf Amin. #30

5 – Omar Ahmed. #40

Requirements specifications:

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 programmingassignments.
In phase 2 of the project, you are going to build on the previous phase and use its output toimplement pass 2 of the assembler.

Specifications

a) The assembler is to execute by enteringassemble <*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 textbook in section 2.1.1 and 2.3.5.

2. A report at the end of pass2. Pass1 and Pass2 errors should be included aspart of the assembler report, exhibiting both the offending line of source codeand 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:

This project consists of 2 Main classes: pass 1 and pass 2.

Pass 1 produces Intermediate file and SYMTAB of all labels with their assigned addresses.

At the beginning of pass 2 this SYMTAB is loaded in the beginning in a hash table of key is the name and the value is a pair contain its address and its type (relative or absolute) and it is used whenever a label is found.

And also at the beginning the OPTAB is loaded in a hash map which contains the operation as key and its op code as value.

After all these are loaded, the first line of the intermediate file is read and the PC is set to the value in the operand field of start.

Then program reads the second line to get the PC of the next instruction and calculate the displacement which is equal to this op address– next op address.

This is done until we find the line that contains END statement.

If the operation is one that doesn’t have operand such as “NO BASE” or END then it is handled separately such that there is no opcode to be generated.

If it is WORD or BYTE then the operand value in hexadecimal is saved as the opcode.

If the operand is that which takes format 2, its opcode is get from the hash table and converted to binary.

And then the bits of n i x b p e are then added the previous from op code.

They are now 12 bits then we attach to them the 12 bits of the operand.

These n i x b p e are Boolean variables and are set according to the operand.

If the operand contains ‘+’ sign the this address is to be adjusted to format four and 20 bits of the address is to be added to the previous 12 bits.

If the operand contains # then I is set to true, if it contains @ n is to be true and so on.

And then we check if this combination is valid or not, if not an error is to be reported.

And the these bits is to be converted to hexadecimal representation.

During calculating the address, if it will > 2048 or <-2048 then base addressing is used.

If base addressing doesn’t fit (>4096) either an error to be reported also.

The error:"*** error could not generate object code ".

Data Structures:

:Op-Table code -1

A hash table which contains the code for each mnemonic <Mnemonic, Op-Code>.

:Op-Table size -2

A hash table which contains the size of each operation(mnemonic)
.<<Mnemonic, Size

:Symbol-Table -3

A hash table which contains the address of each label, assigned from pass 1
.<<Label, Address

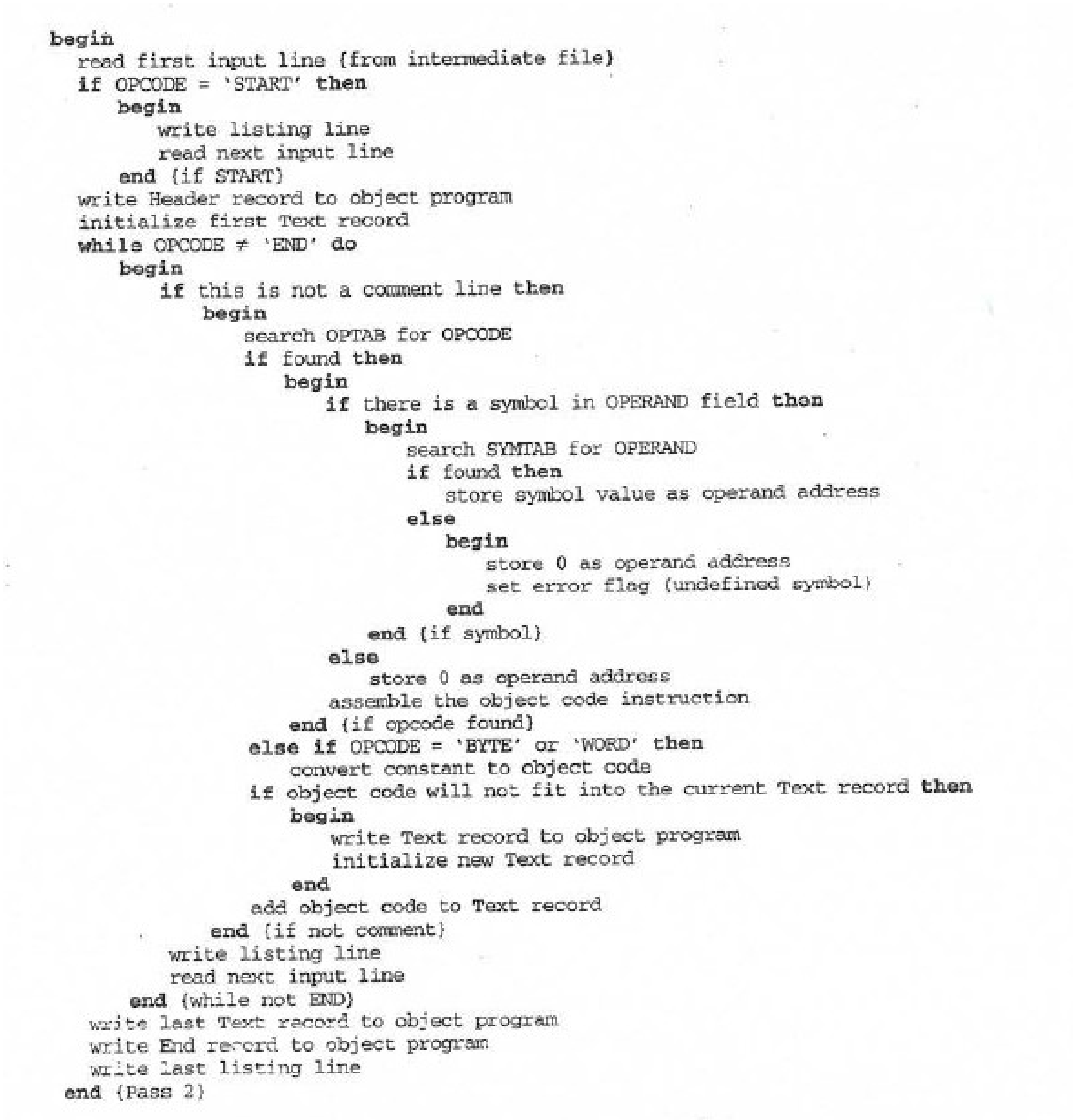
:Literals Queue -4

A Queue which contains the literals encountered during pass 1.

This queue is emptied once LTORG is used, and all the literals inside will be given a value.

At the end of the program, all the remaining literals will be added to a pool at the end of the program.

Algorithm:



```
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}
```

Assumptions:

The code should be written in upper case.

Screen Shots: