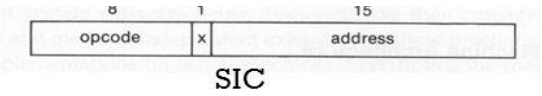
CSN - 252
ASSEMBLER DESIGN (SIC/XE MACHINE)

- Pratyush Kumar
- 20114076

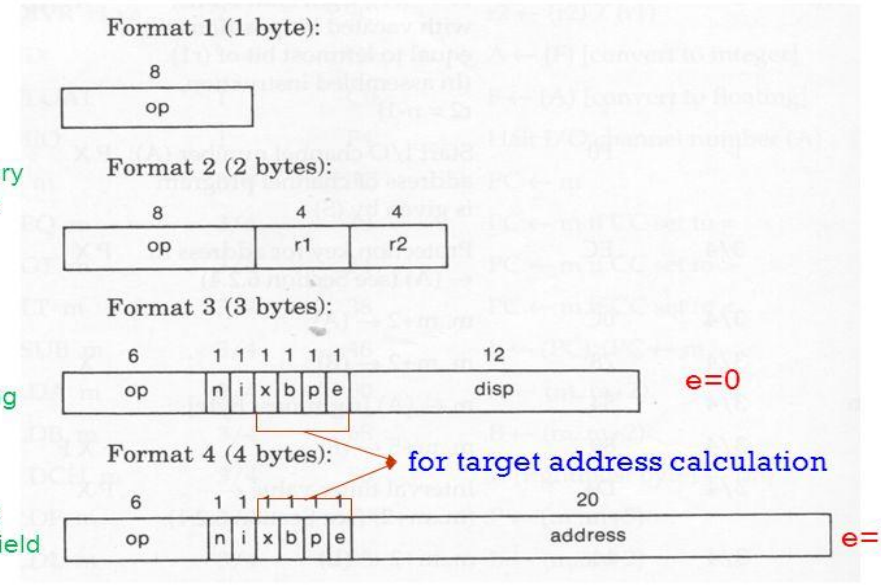
OBJECTIVE:

The objective of this project is to implement a two pass assembler for the SIC/XE architecture.
The assembler will support all 4 instruction formats and all the various addressing modes.



SIC

- Instruction formats



Addressing modes

- Base relative (n=1, i=1, b=1, p=0)
- Program-counter relative (n=1, i=1, b=0, p=1)
- Direct (n=1, i=1, b=0, p=0)
- Immediate (n=0, i=1, x=0)
- Indirect (n=1, i=0, x=0)
- Indexing (both n & i = 0 or 1, x=1)
- Extended (e=1 for format 4, e=0 for format 3)

Features implemented by the assembler:

1. Literals
2. Symbol defining statements
3. Expressions
4. Program Blocks

We give input.txt as the input to the assembler. This file contains the machine instructions which the assembler converts into object code.

Execution of the Assembler:

1. Pass1 generates a symbol table and an intermediate file for Pass2.
2. Pass2 generates a listing file containing the input assembly code and address, block number, object code of each instruction.
3. Pass 2 also generates an object program including the following type of record: H, D, R, T, M and E types.
4. An error file is also generated to identify any errors in the assembly program.

Working of the Assembler:

PASS 1(defines symbol)

1. Assigns address to all statements in the program.

2. Saves the values(addresses) assigned to labels for use in PASS2.
3. Perform some processing of the assembler directives(This includes processing that affects address assignment, such as determining the length of data areas defined by BYTE, RESW, etc).
4. Write any errors identified in the error_input.txt file under PASS1 heading.

Algorithm for PASS1:

Pass 1:

```
begin
  read first input line
  if OPCODE = 'START' then
     begin
        save #[OPERAND] as starting address
        initialize LOCCTR to starting address
        write line to intermediate file
        read next input line
     end {if START}
  else
     initialize LOCCTR to 0
  while OPCODE ≠ 'END' do
     begin
        if this is not a comment line then
           begin
              if there is a symbol in the LABEL field then
                 begin
                    search SYMTAB for LABEL
                    if found then
                       set error flag (duplicate symbol)
                    else
                       insert (LABEL,LOCCTR) into SYMTAB
                 end {if symbol}
              search OPTAB for OPCODE
              if found then
                 add 3 {instruction length} to LOCCTR
              else if OPCODE = 'WORD' then
                 add 3 to LOCCTR
              else if OPCODE = 'RESW' then
                 add 3 * #[OPERAND] to LOCCTR
              else if OPCODE = 'RESB' then
                 add #[OPERAND] to LOCCTR
              else if OPCODE  = 'BYTE' then
                 begin
                    find length of constant in bytes
                    add length to LOCCTR
                 end {if BYTE}
              else
                 set error flag (invalid operation code)
           end {if not a comment}
        write line to intermediate file
        read next input line
     end {while not END}
  write last line to intermediate file
  save (LOCCTR - starting address) as program length
end {Pass 1}
```

PASS 2(assemble instructions and generate object program):

1. Assemble instructions (translating operation codes and looking up addresses).
2. Generate data values defined by BYTE, WORD, etc.
3. Perform processing of assembler directives not done during PASS1.
4. Write the object program in the object_input.txt and the assembly listing.
5. Write any errors identified in the error_input.txt file under PASS2 heading.

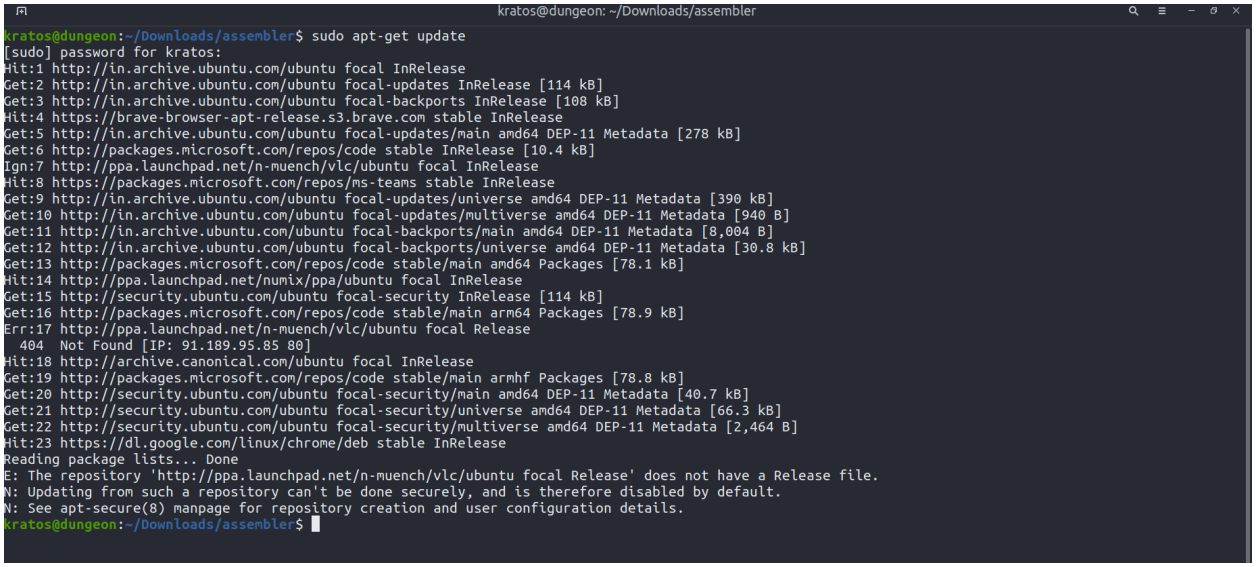
Algorithm for PASS2:

Pass 2:

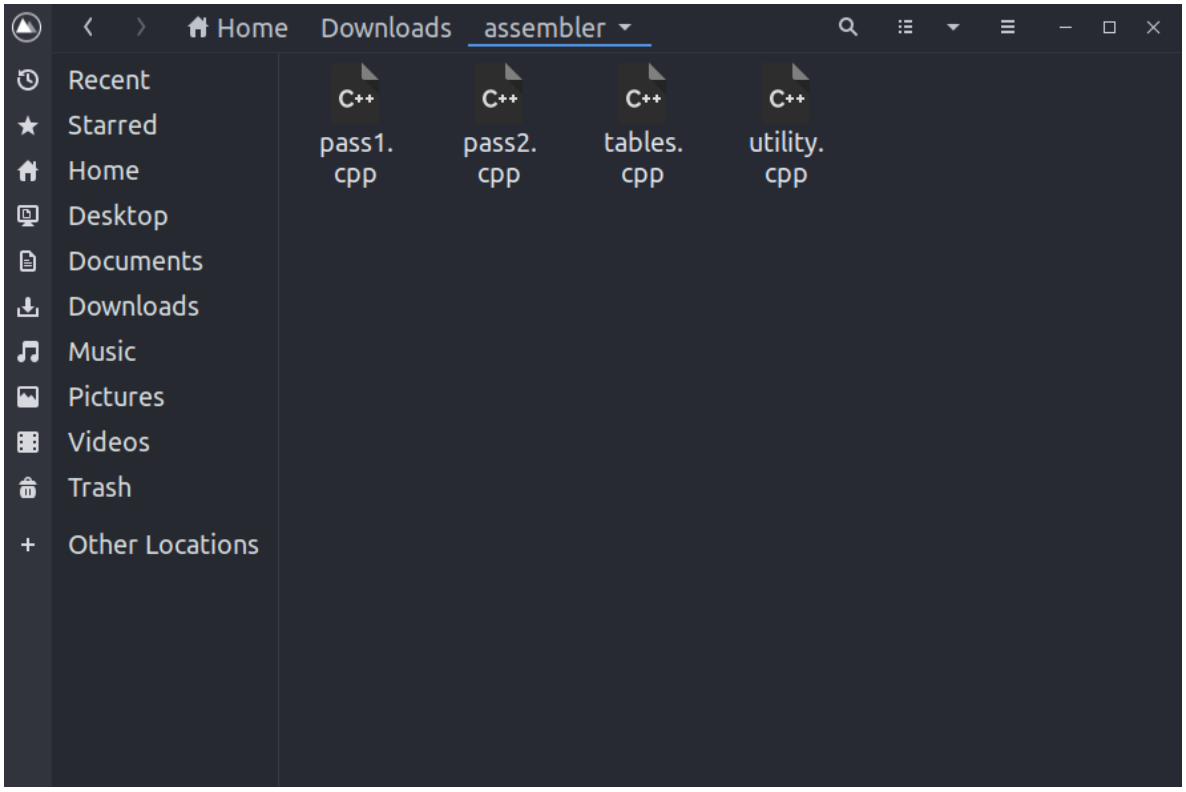
```
begin
  read first input line {from intermediate file}
  if OPCODE = 'START' then
     begin
        write listing line
        read next input line
     end {if START}
  write Header record to object program
  initialize first Text record
  while OPCODE ≠ 'END' do
     begin
        if this is not 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}
```

Steps to Use the Assembler

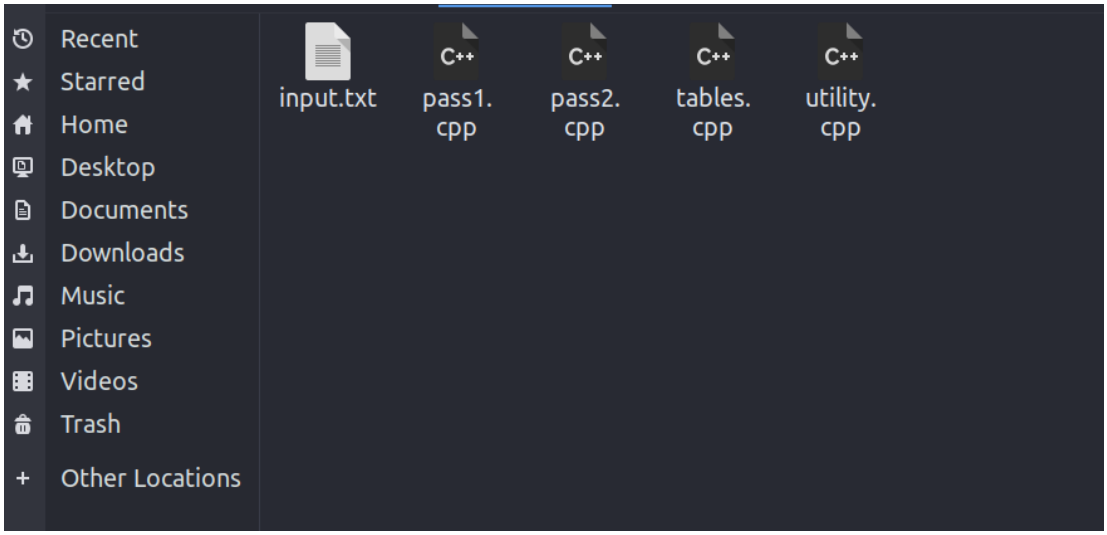
1. **Update all the packages.**
   sudo apt-get update

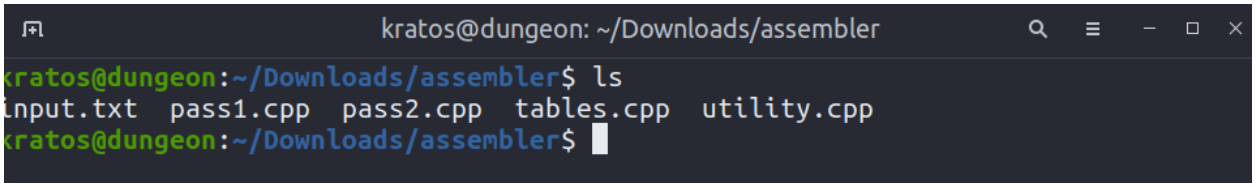


2. **Save all the four files : pass1.cpp, pass2.cpp, tables.cpp, utilities.cpp into one directory.**



3. **Add input.txt, which contains the SIC/XE instructions, to this directory.**

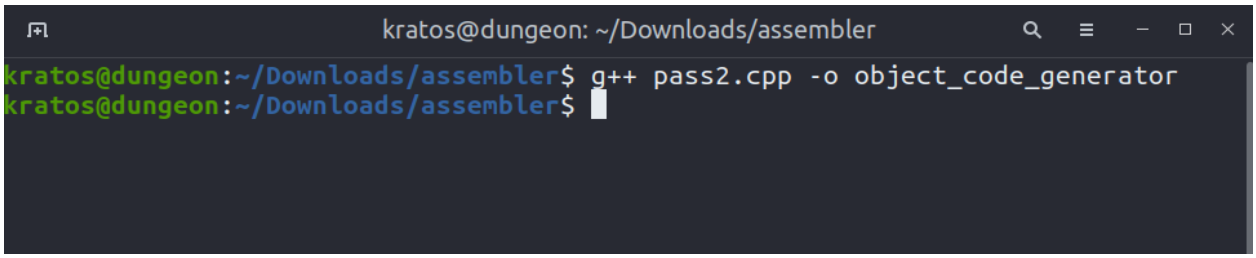




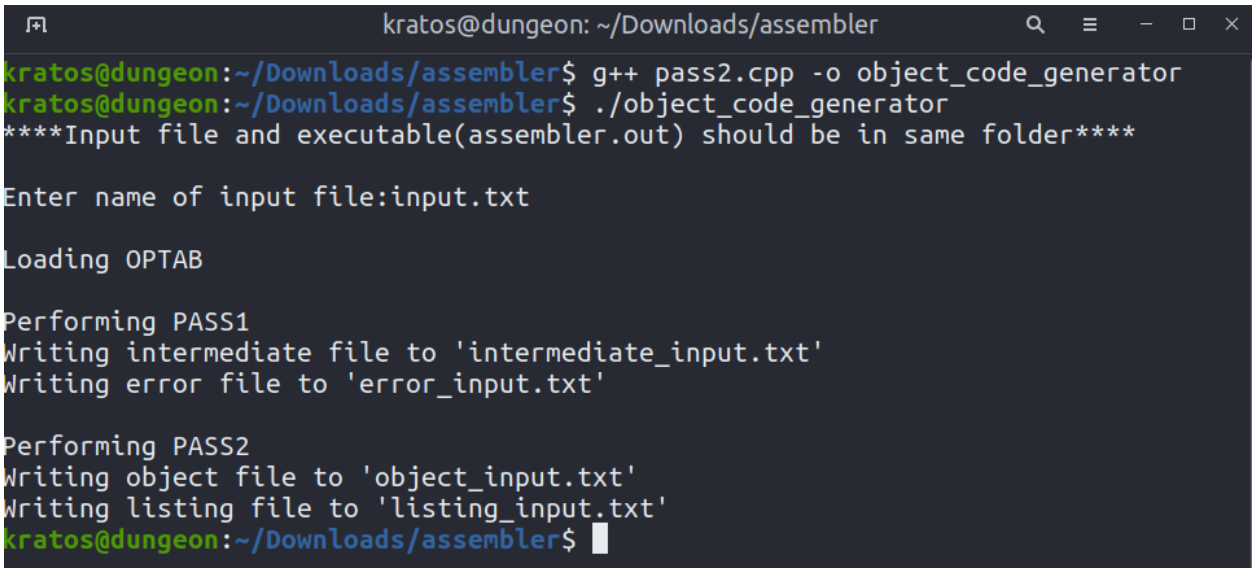
input.txt

```
1   SUM     START 0
2   FIRST   LDX   #0
3           LDA   #0
4           +LDB  #TABLE2
5           BASE  TABLE2
6   LOOP    ADD   TABLE,X
7           ADD   TABLE2,X
8           TIX   COUNT
9           JLT   LOOP
10          +STA  TOTAL
11          RSUB
12  COUNT   RESW  1
13  TABLE   RESW  2000
14  TABLE2  RESW  2000
15  TOTAL   RESW  1
16          END   FIRST
17
```

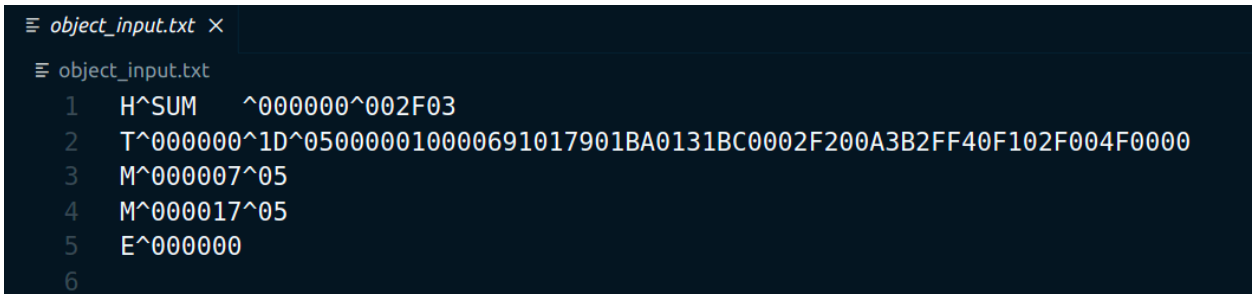
4. **Compile pass2.cpp with -o flag to generate an executable file.**
   G++ pass2.cpp -o <exe_file_name>



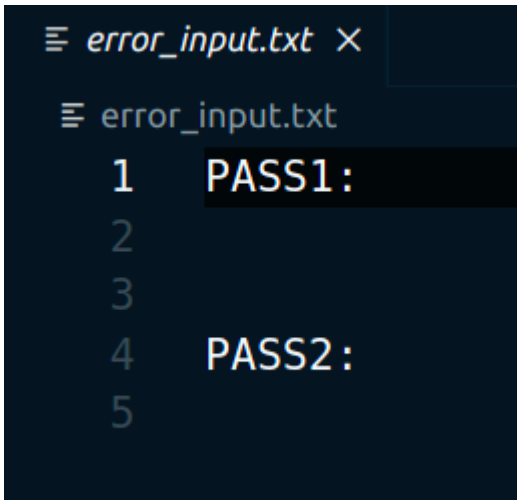
5. **Run the executable file and give input.txt as filename when prompted.**
   ./<exe_file_name>



6. **The object code is generated and saved in the file object_input.txt.**



7. **The file error_input.txt is also generated which reports any irregularity which may have been present in the instructions contained in the input file.**



Conclusion:

I have implemented an assembler in the SIC/XE architecture. This assembler converts the SIC/XE instructions into machine understandable object code.