Pass 2 Report

Names:

1. Ahmed Khaled El Saeid. (9)
2. Mohamed Fathallah. (14)
3. Islam Salah Mohamed. (16)
4. Abd-El Hady Mohamed Aly. (42)
5. Mohamed Nabil Hassan. (58)

Requirements Specifications:

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.

Specifications :

- The assembler executes by entering assemble<src name>.
- The output includes :
    - Object code file .
    - A report at the end of pass2 containing errors of both pass1 and pass2 .
- The assembler supports EQU and ORG .
- The assembler supports simple expression evaluation .

EQU and ORG statements :

EQU and ORG statements are handled in the addressing part in pass1 as they are related to addressing of each statement and the symbol table .

- Whenever an EQU statement is encountered , the operand is either an expression or not . a method – is_supect_expression(string operand)- is called in order to determine if the operand is an expression or not .
- If the operand is an expression , a method- eval_expression(statement)- is called in order to evaluate the expression and return its value as an integer .
- If the operand is not an expression it is evaluated . if it is a constant its value is taken , if it is a variable its address is obtained from the symbol table .
- After evaluating the operand this value is stored as the address of the label using EQU statement.
- Whenever an ORG statement is encountered , the operand is evaluated in the same way as the EQU statement .
- After the operand is evaluated , the program counter is set with this value and the program continues the addressing process.
- Both statements are validated before the addressing process in order to assure that no forward reference is used for EQU and ORG.

Expressions:

Validation :

- Any operand is suspect to be expression, so using is_suspect_exp(string s) method return true if the operand suspect to be expression then it should be validated
- The valid expression has only one arithmetic operation excluding multiplication as it's symbol stands for another concept
- The operand of the expression can be user defined symbol , special symbol as (*) or number.
- Only legal expression take one of the following forms :

sym - sym	* + c	c + c
sym + c	* - c	c - c
sym - c		c + sym
		c + *

Expressions :

- Simple expression is validated and if it isn't a valid expression , the assembler will set the flag with 'Illegal Expression' error .
- Legal Simple expressions :
  - (Symbols | *)- (Symbol | * | constant)
  - Constant - Constant
  - (symbol | * | constant)+(constant)
  - constant / constant
- Illegal Simple expression :
  - (Constant) – (Symbol | *)
  - (Symbol | *) + (Symbol | *)
  - Symbol / constant
  - constant / Symbol
  - Symbol / symbol
  - (Symbol |*|constant) <any operation other than [+, -, /]>
  - (Symbol | * | constant)
- Forward reference in EQU and ORG is illegal and the assembler will set the error flag in the case of forward reference .

Object code:

Design :

In this part we are going to talk about the object code generation. This part is divided into 2 main parts helper functions and main function. The helper functions like int_to_hexa, bin_to_int, neg_int_to_hexa and output_object_code_per_statement, the main function is ob_code_generate.

- int_to_hexa: The function of it to produce a string representing the hexadecimal value of the number, it takes 2 parameters, the first is the number to be converted, the second parameter is the number of digits or places we want to put the hexadecimal string in.
- bin_to_int: Converts a binary number into its integer value, it works by taking the binary number as a string and converts it to integer.
- neg_int_to_hexa: Similar to int_to_hexa, but the main difference is that this function deals with the negative numbers, by taking its value from the 2's complement representation of the number in the computer.
- output_object_code_per_statement: Prints the current statement or line in the object file, this function takes statement as input.
- ob_code_generate: It calculates the object_code of each instruction and adds them to the current statement, and also it checks for some errors, for instance if the displacement is out of range meaning it exceeded the allowed range, other error is no defined operand, meaning that the programmer is trying to use a label that wasn't defined in the whole program either in the previous code or the forward code.

Algorithm Description:

The code works by calling the ob_code_generate function which by its role calls the helper functions. In the main function the at first we check if this statement was a comment, if yes then we pass the current loop without adding an object code, then we see if this statement was an expression operand, if yes we check if it has any undefined symbol so we can add an undefined error, however if we found all expressions are defined then we check if the current expression is absolute or

relocatable if it is absolute then we calculate the address and put it right away in the displacement and pass the current loop, after passing these if conditions we check if the current mnemonic is a directive we check whether it is a word or a byte mnemonic if any then we start generating the proper object code for any of them, if it wasn't a directive then check if it is a format 2 instruction, if true then we calculate the number of the first operand and if it was a CLEAR or TIXR instructions, then we stop at this point, otherwise we look for another register and also get its number and form the object code of a format 2 instruction, however if it wasn't a format 2 then we go for format 3 or 4, at first we calculate the first 3 digits of the object code as they are the same in either format 3 or 4, we get the first digit from the operation code, then calculate the second digit using a combination of both operation code second digit and the ni flags, after that we calculate the third digit in the object code using the xbpe flags. We check if the current format 4 if true then we calculate the target address right away and put it in the displacement otherwise we subtract from it the PC counter as the code works only using PC counter.

Object File:

Design :

The design of this part is divided into four main functions:

- The first one creates the header record
- The second function computes the text records for each statement given to it and decide when to start a new text record
- The third one decides for each statement if it needs a modification record and if needed it creates one.
- The fourth function creates the end record and print all of these records in a file

Algorithm Description:

- header_record:

This function first finds the start starting address and the name of the program by finding the present in the first statement and it also finds the length of the program by finding the address of the last statement

- generate_record(statement s):

This function takes a statement s as an argument and it adds its object code to the current text record and if the length of the text record exceeds 1E it generate a new one and adds to it the object code of the current statement

- generate_mrec():

This function iterates on all statements and if a statement is found to be in format 4 and its operand is an absolute or relocatable expression it will create a new modification record for this statement.

- end_record():

This function creates the end record which contain the start address and then prints all these records in the object file

Assumption:

The current text recorded is ended and a new one is created in the following cases:

1. If the current object code will make the length of the current text record exceeds 1E
2. If a RESB or RESW or ORG statement is found

Sample Runs:

```
.2345678901234567890123456789012345678901¶
COPY    start   0¶
FIRST   STL     #16¶
        LDB     #LENGTH¶
CLOOP   JSUB    ENDFIL¶
        +LDA    LENGTH+1¶
        COMP    #0¶
        +JEQ    ENDFIL¶
        +JSUB   WRREC¶
        J       CLOOP¶
ENDFIL  LDA     EOF¶
        STA     BUFFER¶
        LDA     #3¶
        STA     @111¶
        +JSUB   WRREC¶
        J       @RETADR¶
EOF     BYTE    C'EOF'¶
RETADR  RESW    1¶
LENGTH  RESW    1¶
BUFFER  RESB    4096¶
RDREC   CLEAR   X¶
        CLEAR   A¶
        CLEAR   S¶
        +LDT    #4096¶
RLOOP   TD      INPUT¶
        JEQ     RLOOP¶
        TD      INPUT¶
        COMPR   A,S¶
        JEQ     EXIT¶
        TIXR    T¶
        JLT     RLOOP¶
EXIT    STX     #10¶
        RSUB¶
INPUT   BYTE    X'F1'¶
.¶
.¶
WRREC   CLEAR   X¶
        LDT     #32¶
WLOOP   TD      OUTPUT¶
        JEQ     WLOOP¶
        WD      OUTPUT¶
        TIXR    T¶
        JLT     WLOOP¶
        RSUB¶
OUTPUT  BYTE    X'05'¶
        END
```

```
1       .2345678901234567890123456789012345678901¶
2   000000  COPY    start   0¶
3   000000  FIRST   STL     #16¶
4   000003          LDB     #LENGTH¶
5   000006  CLOOP   JSUB    ENDFIL¶
6   000009          +LDA    LENGTH+1¶
7   00000D          COMP    #0¶
8   000010          +JEQ    ENDFIL¶
9   000014          +JSUB   WRREC¶
10  000018          J       CLOOP¶
11  00001B  ENDFIL  LDA     EOF¶
12  00001E          STA     BUFFER¶
13  000021          LDA     #3¶
14  000024          STA     @111¶
15  000027          +JSUB   WRREC¶
16  00002B          J       @RETADR¶
17  00002E  EOF     BYTE    C'EOF'¶
18  000031  RETADR  RESW    1¶
19  000034  LENGTH  RESW    1¶
20  000037  BUFFER  RESB    4096¶
21  001037  RDREC   CLEAR   X¶
22  001039          CLEAR   A¶
23  00103B          CLEAR   S¶
24  00103D          +LDT    #4096¶
25  001041  RLOOP   TD      INPUT¶
26  001044          JEQ     RLOOP¶
27  001047          TD      INPUT¶
28  00104A          COMPR   A,S¶
29  00104C          JEQ     EXIT¶
30  00104F          TIXR    T¶
31  001051          JLT     RLOOP¶
32  001054  EXIT    STX     #10¶
33  001057          RSUB¶
34  00105A  INPUT   BYTE    X'F1'¶
35              .¶
36              .¶
37  00105B  WRREC   CLEAR   X¶
38  00105D          LDT     #32¶
39  001060  WLOOP   TD      OUTPUT¶
40  001063          JEQ     WLOOP¶
41  001066          WD      OUTPUT¶
42  001069          TIXR    T¶
43  00106B          JLT     WLOOP¶
44  00106E          RSUB¶
45  001071  OUTPUT  BYTE    X'05'¶
46  001072          END¶
```

```
Hcopy^000000^001072¶
T000000^1E^15001069202E4B20120310003529000033100O1B4B10105B3F2FEB032010¶
T00001E^13^0F20160100030E006F4B10105B3E2003454F46¶
T001037^1D^B410B400B44075101000E32016332FFAE32010A004332005B8503B2FED¶
T001054^1E^11000A4F0000F1B410750020E3200E332FFADF2008B8503B2FF24F000005¶
M00000A^05¶
M000011^05¶
M000015^05¶
M000028^05¶
E000000¶
```

```
0       COPY    start   0                       ¶
0       FIRST   STL     #16             150010  ¶
3               LDB     #LENGTH         69202E  ¶
6       CLOOP   JSUB    ENDFIL          4B2012  ¶
9               +LDA    LENGTH+1        03100035¶
13              COMP    #0              290000  ¶
16              +JEQ    ENDFIL          3310001B¶
20              +JSUB   WRREC           4B10105B¶
24              J       CLOOP           3F2FEB  ¶
27      ENDFIL  LDA     EOF             032010  ¶
30              STA     BUFFER          0F2016  ¶
33              LDA     #3              010003  ¶
36              STA     @111            0E006F  ¶
39              +JSUB   WRREC           4B10105B¶
43              J       @RETADR         3E2003  ¶
46      EOF     BYTE    C'EOF'          454F46  ¶
49      RETADR  RESW    1                       ¶
52      LENGTH  RESW    1                       ¶
55      BUFFER  RESB    4096                    ¶
4151    RDREC   CLEAR   X               B410    ¶
4153            CLEAR   A               B400    ¶
4155            CLEAR   S               B440    ¶
4157            +LDT    #4096           75101000¶
4161    RLOOP   TD      INPUT           E32016  ¶
4164            JEQ     RLOOP           332FFA  ¶
4167            TD      INPUT           E32010  ¶
4170            COMPR   A,S             A004    ¶
4172            JEQ     EXIT            332005  ¶
4175            TIXR    T               B850    ¶
4177            JLT     RLOOP           3B2FED  ¶
4180    EXIT    STX     #10             11000A  ¶
4183            RSUB                    4F0000  ¶
4186    INPUT   BYTE    X'F1'           F1      ¶
4187    WRREC   CLEAR   X               B410    ¶
4189            LDT     #32             750020  ¶
4192    WLOOP   TD      OUTPUT          E3200E  ¶
4195            JEQ     WLOOP           332FFA  ¶
4198            WD      OUTPUT          DF2008  ¶
4201            TIXR    T               B850    ¶
4203            JLT     WLOOP           3B2FF2  ¶
4206            RSUB                    4F0000  ¶
4209    OUTPUT  BYTE    X'05'           05      ¶
4210            END                             ¶
```