SIC -XE Assembler

-Richa (20114081)

Github Link: https://github.com/Richa-iitr/SICXE-Assembler.git

The SIC-XE assembler is an advance Version of SIC assembler with features such as register to register instructions, immediate instructions etc which help it improve the execution speed of the program.

In this project, an assembler is written in CPP language which follows pass 1 and pass 2. It includes all the instructions for SIC-XE, opcodes in OPTABLE file for all formats 1,2,3 and 4. Apart from this the addressing modes and program relocation are also taken care of for all the scenarios. Also, error checking is done and errors are written in a separate file. The output of pass1 i.e. the intermediate file is also stored and the symbol table, and literal table are also stored.

Instructions:

```
ADD, ADDF, ADDR, AND, CLEAR, COMP, COMPF,COMPR, DIV,
DIVF, DIVR, FIX, FLOAT, HIO, J, JEQ, JGT, JLT, JSUB,
LDA, LDB, LDCH, LDF, LDL, LDS, LDT, LDX, LPS, MUL,
MULF, MULR, NORM, OR, RD, RMO, RSUB, SHIFTL, SHIFTR,
SIO, SSK, STA, STB, STCH, STF, STI, STL, STS, STSW,
STT, STX, SUB, SUBF, SUBR, SVC, TD, TIO, TIX, TIXR, WD
```

The addressing modes :

- **Immediate:** n=0, i=0, x=0
- **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
- **Indirect:** n=1, i=0, x=0
- **Indexed:** n=0,i=0, x=1 or n=1, i=1, x=1
- **Extended:** format 3- e=0, format 4- e=1

The instruction formats:

Format 1		1 byte		op											
Format 2		2 bytes				op		r1		r2					
Format 3		3 bytes				op		n	i	x	b	p	e	disp	
Format 4		4 bytes				op		n	i	x	b	p	e	address	

Other features

- Literals (for e.g. =X'01')
- Symbol defining instructions (for e.g. EQU)
- Expressions (for e.g. BUFEND-BUFFER)
- Program blocks (for e.g. USE cblock)
- Control Sections (not implemented in assembler)

Structure of the code

Data structures used

Structs

- blocktab: includes details of the block such as start address of the block, name of the block, location counter corresponding to that block, number of block, whether the block exists or not.
- littab: includes details needed to be stored in the literal table such as value, address, exists, blockNumber
- optab: includes the details needed to be stored in the optable. These includes opcode, format of the instruction, whether the opcode exists or not.
- regtab: stores the details to be stored in the register table including num, whether the register exists or not.
- symtab: stores the details to be stored in the symbol table including address or value, name, whether the symbol is relative or not, blocknumber, whether the symbol exists or not.

Maps

- BLOCKS: stores name of block as key and blocktab struct as value. Initialized with default Block. Other blocks are added in the passes.
- LITTAB: represents the literal pool, stores name of literal to littab struct. Is filled when the LTORG instruction is encountered during passes.
- OPTAB: opcode as key and optab struct as value. All the opcodes of the instructions supported by SIC-XE are stored here.

- REGTAB: register name to regtab struct. Stores details of the registers available in SIC-XE architecture.
- SYMTAB: name of symbol to symbol data.

Functions

pass1() : The pass 1 of the assembler works as:

- It searches for the literal in the literal table after parsing the line from the input file. If the symbol is not present we need to add it to the literal table(with the address unassigned) else do nothing. If LTORG or END is encountered scan the LITTAB and assign addresses and update the location counter.
- The function checks for the input file, displays error in case of any discrepancies. If the file is opening properly then each lines is parsed one by one. We also initialise variables such as LOCCTR=0, line number=0, blocknumber=0, labels, opcode, operands etc. If the line is comment line, we take the line and print it to our intermediate file in the format, updating the line numbers.
- If the line is not comment, we check if the START of the program is encountered.
    - If START, write to intermediate file, read next input line.
    - Else If the opcode = USE, in this case if the operand is empty then block name is default use the default block name, else the new blockname is the operand.
    - Insert the block name in the blocktable if any new block has been encountered.
    - Check for the symbol in the symbol table
        - If present display error that label name is repeated, if symbol also exists in extDef then we need to add the address in the extdef table.
        - Elseif not present, add the symbol to symbol table along with the details, update Location counter.
    - Search for the OPCODE in the optable, if it exists find the format and increment the location counter, LOCCTR. I the symbol isn't in the symtab we check if the symbol is used to reserve memory or not i.e. compare the label with 'RESW', 'RESB', 'WORD', 'BYTE', 'EXTDEF' etc. Update the symbol table accordingly.
        - In case of LTORG, call forLTORG() method
            - The method, printed the literal pool present till time taking values as arguments from pass1, line number is also updated. If address isn't added store current address in LITTAB and increment the LOCCTR on the basis of the literal

        - In case of EQU we find the expression, evaluate it if it is valid using evaluateExpression() method.
            - evaluateExpression(): it parses the expression, if symbol doesn't exist in the symtab we generate error message, else check the pairs to know relative or absolute expressions, print errors if any.
        - Appropriate error messages are printed at each step and the intermediate file is updated.
    - Store the details such program length, intermediate file is made and relevant details for the pass2 are stored.

pass2() : The pass 2 of the assembler works as:

- If the literal is in the instruction is present in the literal tab, search for the literal table and search for the address. Insert values in appropriate places in the object program. It also generates the modification record if the literal value represents an address in the program.
- The pass 2 uses the intermediate file generated from the pass 1 using the function.
- It generates the listing file and the object program. Print he appropriate error messages if any error is encountered.
- Iterate through the lines of the intermediate file. Read them one by one.
- If the lines are not comment, check for opcode START, initialise start address as LOCCTR, write the line in the listing file.
- Write the header record,.
- While opcode is not END, take lines from the intermediate file read them, and update the listing file. Store the data in the Text records.
- We will write the object code on the basis of the types of formats used in the instruction. Based on different types of opcodes such as 'BYTE','WORD','BASE','NOBASE','EXTDEF','EXTREF','CSECT', we will generate different types of object codes. For the format 3 and format 4 instruction format, we will use the createObjectCodeFormat34() function in the pass2.cpp. For writing the end record, we use the writeEndRecord() function.
- For the instructions with immediate addressing, we will write the modification record.
- Functions:
    - readTillTab()- takes in the string as input and reads the string until tab('\t') occurs.
    - readIntermediateFile()- takes in line number, LOCCTR, opcode, operand, label and input output files. If the line is comment returns true and takes in the next input line. Then using the readTillTab() function, it reads the label, opcode, operand and the comment.

Based on the different types of opcodes, it will count in the necessary conditions to take in the operand.
    - createObjectCodeFormat34()- It checks the situations in which the opcode can be and then according to the operand and the number of half bytes calculates the object code for the instruction. It also modifies the modification record when there is a need to do so.
    - writeEndRecord()- It will write the end record for the program.

Helper Functions: It contains useful functions that will be required by the other files.

- intToStringHex()-converts int to hex string
- expandString()- takes input string and character to be added and expands the string
- stringHexToInt()- converts the hexadecimal string to integer.
- stringToHexString()- converts the string into its hexadecimal equivalent string.
- checkWhiteSpace()- checks if blanks are present.checkCommentLine()- check the comment by looking at the first character of the input string, and then accordingly returns true if comment or else false.
- if_all_num()- checks if all the elements of the string of the input string are number digits.
- readFirstNonWhiteSpace()- takes in the string and iterates until it gets the first non-spaced character. It is a pass by reference function which updates the index of the input string until the blank space characters end and returns void.
- writeToFile()- takes in the name of the file and the string to be written on to the file. Then writes the input string onto the new line of the file.
- getOpcode()- for opcodes of format 4, for example +JSUB the function will see whether if the opcode contains some additional bit like '+' or some other flag bits, then it returns the opcode leaving the first flag bit.
- getOpcodeFormat()- returns the flag bit if present in the input string or else it returns null string.
- Class EvaluateString – contains the functions :
    - -peek()- returns the value at the present index.
    - -get()- returns the value at the given index and then increments the index by one.
    - -number()- returns the value of the input string in integer format.

Steps to compile and run

```
PS D:\Semester 3\CSN-252\Assembler> g++ Pass2.cpp
PS D:\Semester 3\CSN-252\Assembler> ./a
****Input file and executable(assembler.out) should be in same folder****

Enter name of input file:sample.asm

Loading OPTAB

Performing PASS1
Writing intermediate file to 'intermediate_sample.asm'
Writing error file to 'error_sample.asm'
Writing SYMBOL TABLE
Writing LITERAL TABLE

Performing PASS2
Writing object file to 'object_sample.asm'
Writing listing file to 'listing_sample.asm'
PS D:\Semester 3\CSN-252\Assembler>
```

Screenshots of the tables

Intermediate file:

```
Line    Address BlockNumber Label   OPCODE  OPERAND Comment
5   00000   0   SUM START   0
10  00000   0   FIRST   LDX #0
15  00003   0       LDA #0
20  00006   0       +LDB    #0
25  0000A   0       +LDB    #TABLE2
30  0000E   0       BASE    TABLE2
35  0000E   0   LOOP    ADD TABLE,X
40  00011   0       ADD TABLE2,X
45  00014   0       TIX COUNT
50  00017   0       JLT LOOP
55  0001A   0       +STA    TOTAL
60  0001E   0       RSUB
65  00021   0   COUNT   RESW    1
70  00024   0   TABLE   RESW    2000
75  01794   0   TABLE2  RESW    2000
80  02F04   0   TOTAL   RESW    2
85  02F0A           END FIRST
```

Input:

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

SYMTAB

```
1  **********************************SYMBOL TABLE*****************************
2
3  :-  name:undefined  |address:0  |relative:00000
4  0:- name:    |address:0  |relative:00000
5  COUNT:- name:COUNT  |address:00021  |relative:00001
6  FIRST:- name:FIRST  |address:00000  |relative:00001
7  LOOP:-  name:LOOP   |address:0000E  |relative:00001
8  TABLE:- name:TABLE  |address:00024  |relative:00001
9  TABLE2:-    name:TABLE2 |address:01794  |relative:00001
10 TOTAL:- name:TOTAL  |address:02F04  |relative:00001
11
12 **********************************LITERAL TABLE*****************************
13
14
15
```

LISTING FILE

```
1  Line    Address Label   OPCODE  OPERAND ObjectCode  Comment
2  5   00000   0   SUM START   0
3  10  00000   0   FIRST   LDX #0  050000
4  15  00003   0       LDA #0  010000
5  20  00006   0       +LDB    #0  69100000
6  25  0000A   0       +LDB    #TABLE2 69101794
7  30  0000E   0       BASE    TABLE2
8  35  0000E   0   LOOP    ADD TABLE,X 1BA013
9  40  00011   0       ADD TABLE2,X    1BC000
10 45  00014   0       TIX COUNT   2F200A
11 50  00017   0       JLT LOOP    3B2FF4
12 55  0001A   0       +STA    TOTAL   0F102F04
13 60  0001E   0       RSUB        4F0000
14 65  00021   0   COUNT   RESW    1
15 70  00024   0   TABLE   RESW    2000
16 75  01794   0   TABLE2  RESW    2000
17 80  02F04   0   TOTAL   RESW    2
18
```

OBJECT FILE

```
H^SUM   ^000000^002F0A
T^000000^1E^05000001000069100000691017941BA0131BC0002F200A3B2FF40F102F04
T^00001E^03^4F0000
M^00000B^05
M^00001B^05
E^000000
```