

# ASSEMBLER

## Systems Programming-1 Final-Project (phase-1)

- 1) Mohamed Nabil (62).
- 2) Nesma Emad El din (72).
- 3) Caroline Adel (49).
- 4) Amr Fathy Al dafrawy (48).



# Requirement Specification:

**It is required to implement Phase-1 of a (cross) Assembler for (a subset of) SIC/XE machines.**

**Phase-1 specification requires that an entity is to be designed as follows:**

## **1) Input:**

Source file name "src.txt" via a (.exe).

## **2) Process:**

The input source file is parsed in order to produce the output.

Parsing process handles the following:

- a) Source lines that are (instructions, storage declarations, comments, and assembler directives).
- b) Errors and unhandled directives are handled with warnings and error messages.
- c) For instructions, the parser is to minimally be capable of decoding 2, 3 and 4-byte instructions as follows:
  - ✓ 2-byte with 1 or 2 symbolic register reference (e.g., TIXR A, ADDR S,A)

- ✓ RSUB (ignoring any operand or perhaps issuing a warning).
- ✓ 3-byte PC-relative with symbolic operand to include immediate, indirect, and indexed addressing.
- ✓ 3-byte absolute with non-symbolic operand to include immediate, indirect, and indexed addressing.
- ✓ 4-byte absolute with symbolic or non-symbolic operand to include immediate, indirect, and indexed addressing.
- ✓ The parser is to handle all storage directives (BYTE, WORD, RESW, and RESB).

### **3) Output:**

- a) The symbol table.
- b) The source program in a format similar to a listing file ,also A meaningful error message is printed below the line in which an error occurred.

# Design:

---

## **Pass-1 assembler:**

- **Source lines are read in sequence.**
- **The lines are passed to a parser method.**
- **There are two kinds of parsers**
  - Fixed format parser.
  - Free format parser.
- **According to the desired parsing criterion a parser method is chosen.**
- **The parser method mainly performs the following tasks:**
  - Checks for line syntax and detect errors.
  - Create an Entry object to be added to the list file.
  - Add labels to the symbol table.
- **As these processes are performed methods implemented in the class are used to complement the required functionalities. These methods are thoroughly described in the Algorithms description section.**

## Algorithms Description:

The implementation is encapsulated into one class **Pass1.cpp** (as specified in the description) this class includes

### **A.Class Entry():**

- This class entity represents the lines which are written to the output file.
- Its member variables are (int loc) to hold address of the current entry, and a set of strings (label , op\_code , operand ,comment,error) to describe entry.
- This class includes 2 constructors:
  - 1) Entry(int loc , string label , string op\_code , string operand , string comment, string error)
  - 2) Entry()

### **B.Class Utilities, it includes a set of methods described below:**

#### **1) checkByte:**

##### **○ Parameters:**

A string which is the operand to the BYTE directive.

##### **○ Return type:**

Returns an integer which is the value of the operand.

- **Functionality:**

Checks for byte declaration syntax, returns its value if correct syntax, -1 otherwise.

## **2)checkWord:**

- **Parameters:**

A string which is the operand to the WORD directive.

- **Return type:**

Returns an integer which is the value of the operand.

- **Functionality:**

Checks for word declaration syntax, returns its value if correct syntax, -1 otherwise.

## **3)toDecimal:**

- **Parameters:**

A string which is to be converted.

- **Return type:**

An integer which is the hexadecimal conversion of the string.

- **Functionality:**

Convert a string into hexadecimal notation.

- **Description:**

The result is added up via converting every character into its integer notation and then adding the correct weighting according to the decimal place of the digit ( $16^{\text{weight}}$ ).

#### 4)toInteger:

- **Parameters:**

A string which is to be converted.

- **Return type:**

An integer which is the decimal conversion of the string.

- **Functionality:**

Convert a string into decimal notation.

- **Description:**

The result is added up via converting every character into its integer notation and then adding the correct weighting

according to the decimal place of the digit  
( $10^{\wedge}$  weight).

#### **5)toLower:**

- **Parameters:**

String to be converted.

- **Return type:**

String after conversion.

- **Functionality:**

Convert all characters of the input string to lower case.

#### **6)checkOperand:**

- **Parameters:**

Two strings, the first is an opcode and the second is its corresponding operand.

- **Return type:**

A Boolean to denote the result of the check.

- **Functionality:**

To check whether the operand is a valid match for the opcode or not.



- **Description:**

It first checks for single operand instructions, as "tixr" and "clear", if their corresponding operand is a register true value is returned otherwise false.

Then it checks the format for opcodes that take 2 operands, it checks both are register names and they are separated by a comma, if so it returns true otherwise false.

### **7) validateOpcode:**

- **Parameters:**

A string which is the opcode to be validated.

- **Return type:**

An integer value.

- **Functionality:**

Checks for the opcode in the opcode map, returns an integer value denoting the byte format for the given opcode if found, if the opcode is invalid it returns -1.

### **8) isDuplicateLabel:**

- **Parameters:**

The string of which existence is to be checked for duplicity.

- **Return type:**

A Boolean to denote the check.

- **Functionality:**

Returns true if the label is a duplicate, false if not.

### 9) oneWord:

- **Parameters:**

A vector of strings containing parameters of a source code line of length 1, a vector of strings holding comment line and an integer denoting the format.

- **Return type:**

Void.

- **Functionality:**

Creates a new Entry instance and adds it up to the source code entry table.

Updates the current address with the correct value.

### 10) twoWord:

- **Parameters:**

A vector of strings containing parameters of a source code line of length 2, a vector of strings holding comment line and an integer denoting the format.

- **Return type:**

Void.

- **Functionality:**

Creates a new Entry instance and adds it up to the source code entry table.

Updates the current address with the correct value.

## 11) **threeWord:**

- **Parameters:**

A vector of strings containing parameters of a source code line of length 3, a vector of strings holding comment line and an integer denoting the format.

- **Return type:**

Void.

- **Functionality:**

Creates a new Entry instance and adds it up to the source code entry table.

If a possible error is encountered it is added.

Updates the current address with the correct value.

## 12) `parse_sic`:

- **Parameters:**

String (line of the source code) to be parsed.

- **Return type:**

Void.

- **Functionality:**

Check line syntax, detect errors and convert the input into a form valid for later purpose (writing the list file).

Handles fixed format parsing.

- **Description:**

The input line is first checked for its type if It is a comment it is added as a comment, otherwise the parsing process both checks for syntax and starts separating line

entries, converting them to lower case strings, validating opcodes and whether they match their corresponding operands in order to add the line as a new Entry.

It also checks that appropriate types of code are in their correct positions according to sic machine fixed formatting rules.

### 13) **parse:**

- **Parameters:**

String (line of the source code) to be parsed.

- **Return type:**

Void.

- **Functionality:**

Check line syntax, detect errors and convert the input into a form valid for later purpose (writing the list file).

Handles the free format parsing.

- **Description:**

The input line is first checked for its type if It is a comment it is added as a comment, otherwise the parsing process both checks

for syntax and starts separating line entries, converting them to lower case strings, validating opcodes and whether they match their corresponding operands in order to add the line as a new Entry.

Only separates strings upon spaces since no kind of formatting specification is imposed in free formatting.

#### 14) **fillingMap:**

- **Parameters:**

Void.

- **Return type:**

Void.

- **Functionality:**

Filling up the map holding the opcodes and their formats from an external text file.

#### 15) **checkEndLine:**

- **Parameters:**

String which is the last line in the source code.

- **Return type:**

Void.

- **Functionality:**

Checks that the end line of the program is valid one.

## 16) checkSpace:

- **Parameters:**

String to be checked.

- **Return type:**

Boolean denoting check result.

- **Functionality:**

Checks for spaces in a string, returns true if at least one character was found, false if not (the line is all spaces).

## Main Data Structures:

---

### 1) Maps:

**A map data structure is used:**

- **For the symbol table:**

It is used to insert labels in the symbol table and to easily be able to retrieve them or check for their duplicity in constant time.

- **For saving up SIC/XE machine appendix:**

A two dimensional map is used to save up SIC/XE machine instruction set and their corresponding format.

## Assumptions:

### **1) Errors are produced as follows:**

- In case of an invalid operand
  - "\*\*\*\*Error: Invalid Operand".
- If line length is exceeded above limit
  - "\*\*\*\*Error: Invalid length of the line".
- Invalid line spaces
  - "\*\*\*\*Error: invalid spaces in this line".
- Invalid beginning
  - "\*\*\*\*Error: invalid start of the program".
- Invalid op code
  - "\*\*\*\*Error: Invalid OpCode".
- Duplicate symbols
  - "\*\*\*\*Error: Duplicate Symbol".
- Invalid entry



- "\*\*\*\*Error: Invalid Entry".
- Invalid end program
  - "\*\*\*\*Error: invalid end of the program".

**2) Operands and Labels in the free format cannot include space characters.**

# Sample Runs:

## First sample:

Input:

```
.23456789012345678901234567890
START 1000
ALPHA LDT #10
BETA +LDCH #6
      ADD @GAMMA
      J *
GAMMA BYTE X'01'
THETA RESB 2
      END
|
```

Output:

LineNo	Adress	Label	Op-code	Operand	Comment
0					.23456789012345678901234567890
1	1000		start	1000	
2	1000	alpha	ldt	#10	
3	1003	beta	+ldch	#6	
4	1007		add	@gamma	
5	100a		j	*	
6	100d	gamma	byte	x'01'	
7	100e	theta	resb	2	
8	1010		end		
*****Symbol Table*****					
Symbol		Address			
*****		*****			
alpha		1000			
beta		1003			
gamma		100d			
theta		100e			

## Second sample:

Input:

```
.23456789012345678901234567890
.MISSING START STATEMENT
ALPHA    LDT    #10
BETA     +LDCH  #6
         ADD    @GAMMA
         J      *
GAMMA    BYTE   X'01'
THETA    RESB   2
         END
|
```

Output:

LineNo	Adress	Label	Op-code	Operand	Comment
0					.23456789012345678901234567890
1					.MISSING START STATEMENT
2		alpha ldt #10			
			****Error: invalid start of the program		
3		beta	+ldch	#6	
4	3		add	@gamma	
5	6		j	*	
6	9	gamma	byte	x'01'	
7	a	theta	resb	2	
8	c		end		

\*\*\*\*\*Symbol Table\*\*\*\*\*

Symbol	Address
beta	ffffff
gamma	9
theta	a

## Third sample:

Input:

```
.23456789012345678901234567890
ALPHA    LDT    #10
BETA     +LDCH  #6
         ADD    @GAMMA
         J      *
GAMMA    BYTE   X'01'
THETA    RESB   2
. MISSING END STATEMENT
|
```

Output:

LineNo	Adress	Label	Op-code	Operand	Comment
0					.23456789012345678901234567890
1		alpha ldt #10			
			****Error: invalid start of the program		
2		beta	+ldch	#6	
3	3		add	@gamma	
4	6		j	*	
5	9	gamma	byte	x'01'	
6	a	theta	resb	2	
7	c				. MISSING END STATEMENT
			****Error: invalid end of the program		
			*****Symbol Table*****		
		Symbol	Address		
		*****	*****		
		beta	ffffff		
		gamma	9		
		theta	a		

### Fourth sample:

Input:

```

.2345678901234567890123
PROB2      START      1000
           LDX         INITL
LOOP       LDS         ZERO
           STS         ARRAY,X
           TIX         TEST
           JLT         LOOP
ARRAY      RESW        100
ZERO       WORD        0
INITL      WORD        0
TEST       WORD        100
           END

```

## Output:

LineNo	Adress	Label	Op-code	Operand	Comment
0					.2345678901234567890123
1	1000	prob2	start	1000	
2	1000		ldx	initl	
3	1003				
****Error: invalid spaces in this line					
4	1003	loop	lds	zero	
5	1006		sts	array,x	
6	1009		tix	test	
7	100c				
****Error: invalid spaces in this line					
8	100c		jlt	loop	
9	100f	array	resw	100	
10	113b	zero	word	0	
11	113e	initl	word	0	
12	1141				
****Error: invalid spaces in this line					
13	1141	test	word	100	
14	1144		end		
*****Symbol Table*****					
Symbol		Address			
*****		*****			
array		100f			
initl		113e			
loop		1003			
test		1141			
zero		113b			

## Fifth sample:

Input:

```
.12345678912345678912345678912345678
PROB2    START    1000
          LDX      INITL
LOOP     LDS      ZERO
          STS      ARRAY,X          This Is A Comment
          TIX      TEST
          JLT      LOOP
ARRAY    RESW     100
ZERO     WORD     -5
INITL    WORD     #3
TEST     WORD     100
          END
```

Output:

LineNo	Adress	Label	Op-code	Operand	Comment
0					.12345678912345678912345678912345678
1	1000	prob2	start	1000	
2	1000		ldx	initl	
3	1003	loop	lds	zero	
4	1006		sts	array,x	This Is A Comment
5	1009		tix	test	
6	100c		jlt	loop	
7	100f	array	resw	100	
8	113b	zero	word	-5	
9	113e	initl	word	#3	
10	1141	test	word	100	
11	1144		end		

\*\*\*\*\*Symbol Table\*\*\*\*\*

Symbol	Address
array	100f
initl	113e
loop	1003
test	1141
zero	113b

## Sixth sample:

Input:

```
.12345678912345678912345678912345678
PROB2      START      1000
           LDX         INITL
LOOP       LDS         ZERO
           STS         ARRAY,X          This Is A Comment
           TIX         TEST
           JLT         LOOP
ARRAY      RESW        100
ZERO       WORD        -5
INITL      WORD        #3
TEST       WORD        1knll
STR        BYTE        C'JBKbk'
FIRST      WORD        @-646
SECOND     BYTE        X'4564HJK'
THIRD      BYTE        X'454A'
THIRD      WORD        3
           END
```

Output:

LineNo	Adress	Label	Op-code	Operand	Comment
0					.12345678912345678912345678912345678
1	1000	prob2	start	1000	
2	1000		ldx	initl	
3	1003	loop	lds	zero	
4	1006		sts	array,x	This Is A Comment
5	1009		tix	test	
6	100c		jlt	loop	
7	100f	array	resw	100	
8	113b	zero	word	-5	
9	113e	initl	word	#3	
10	1141	test	word	1knll	
****Error: Invalid Operand					
11	1141	str	byte	c'jbkbk'	
12	1146	first	word	@-646	
13	1149	second	byte	x'4564hjk'	
****Error: Invalid Operand					
14	1149	third	byte	x'454a'	
15	114b	third	word	3	
****Error: Duplicate Symbol					
16	1150	end	end		
*****Symbol Table*****					
Symbol		Address			
*****		*****			
	array			100f	
	first			1146	
	initl			113e	
	loop			1003	
	second			1149	
	str			1141	
	test			1141	
	third			1149	
	zero			113b	

## Seventh sample(free fromat):

Input:

```
.12345678912345678912345678912345678
PROB2      START      1000
           LDX         INITL
LOOP       LDS         ZERO
           STS         ARRAY,X
           TIX         TEST
           JLT         LOOP

ARRAY      RESW        100
ZERO       WORD        -5
INITL      WORD        #3
TEST       WORD        1kn11
STR        BYTE        C'JBKbk'
FIRST      WORD        @-646
SECOND     BYTE        X'4564HJK'
THIRD      BYTE        X'454A'
THIRD      WORD        3
           END
```

Output:

LineNo	Adress	Label	Op-code	Operand	Comment
0					.12345678912345678912345678912345678
1	1000	prob2	start	1000	
2	1000		ldx	initl	
3	1003	loop	lds	zero	
4	1006		sts	array,x	
5	1009		tix	test	
6	100c		jlt	loop	
7	100f	array	resw	100	
8	113b	zero	word	-5	
9	113e	initl	word	#3	
10	1141	test	word	1kn11	
			****Error: Invalid Operand		
11	1141	str	byte	c'jbkbk'	
12	1146	first	word	@-646	
13	1149	second	byte	x'4564hjk'	
			****Error: Invalid Operand		
14	1149	third	byte	x'454a'	
15	114b	third	word	3	
			****Error: Duplicate Symbol		
16	1150		end		
*****Symbol Table*****					
Symbol		Address			
*****		*****			
	array			100f	
	first			1146	
	initl			113e	
	loop			1003	
	second			1149	
	str			1141	
	test			1141	
	third			1149	
	zero			113b	