Documentation for SIC/XE Assembler

Implemented by Java

Group Member

資訊二乙	孔憲文	D0527561
資訊二乙	陳欣惠	D0527783
資訊二乙	賈 澤	D0571783
資訊二丙	赵晨羽	D0571796
資訊二丁	賴恬甜	D0571898

Table of Contents

How to use our Programme

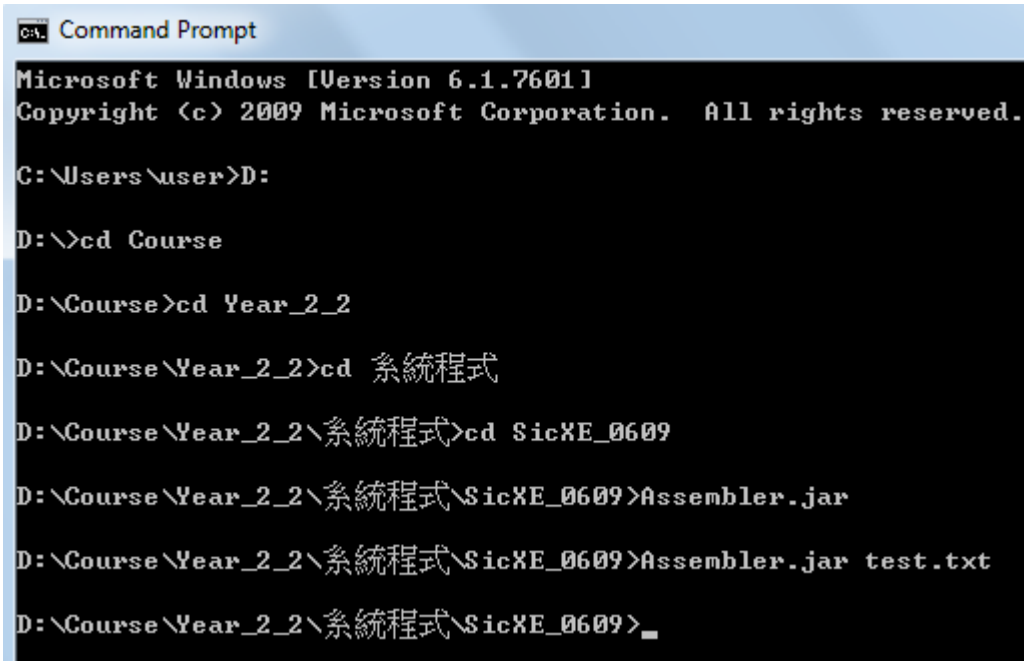
You may import our project to Eclipse and execute the main method in SicXEAssembler.java.
Or else way you can use command prompt until come to this way:

<Path> \ <Project Name> \ Assembler.jar <Source File Name or leave it blank>

*** If you leave the Source File Name blank then it will use “test.xe” by default.

For example:

C:\.....\SicXEAssembler\Assembler.jar **or**
C:\..... \SicXEAssembler\Assembler.jar try.txt **or**
C:\.....\SicXEAssembler\Assembler.jar (other file name that you would like to be executed)



Please make sure your source assembly code follow our format and intend.

Source statement	Show paragraph marks and other hidden formatting symbols
PROG START 1000 CLEAR T END	PROG→START→1000¶ → CLEAR→T¶ → END¶

Introduction to our Programme

Programming Language and Development Environment used

- Java and Eclipse

Fundamental Function

This part will introduce the basic functions of our assembler in both machine-dependent feature and machine independent feature as well. Machine-dependent features include instruction formats handling, addressing modes handling and program relocation. Machine-independent features include literals, symbol-defining statements and expression.

(1) Instruction Formats
- Format 1, 2, 3 and 4

(2) Addressing Modes
- Index, Immediate, Indirect, Direct and Relative(PC and Base)

(3) Assembler Directives

- START, END, RESB, RESW, BYTE, WORD

(4) Literals

- be able to recognize duplicate literals by comparing the character strings that define them

(5) Symbol-Defining Statements

– EQU, ORG

(6) Expression

– be able to handle the expressions which have label name, constant, mathematical symbol (+, -, *, /) and parentheses in the correct order

Limitations

The assembler that we designed was unable to handle following situation:

(1) Program blocks, control sections and program linking.

(2) Can handle BASE <Label Name> but cannot handle BASE * and BASE <Address, such as 1000>.

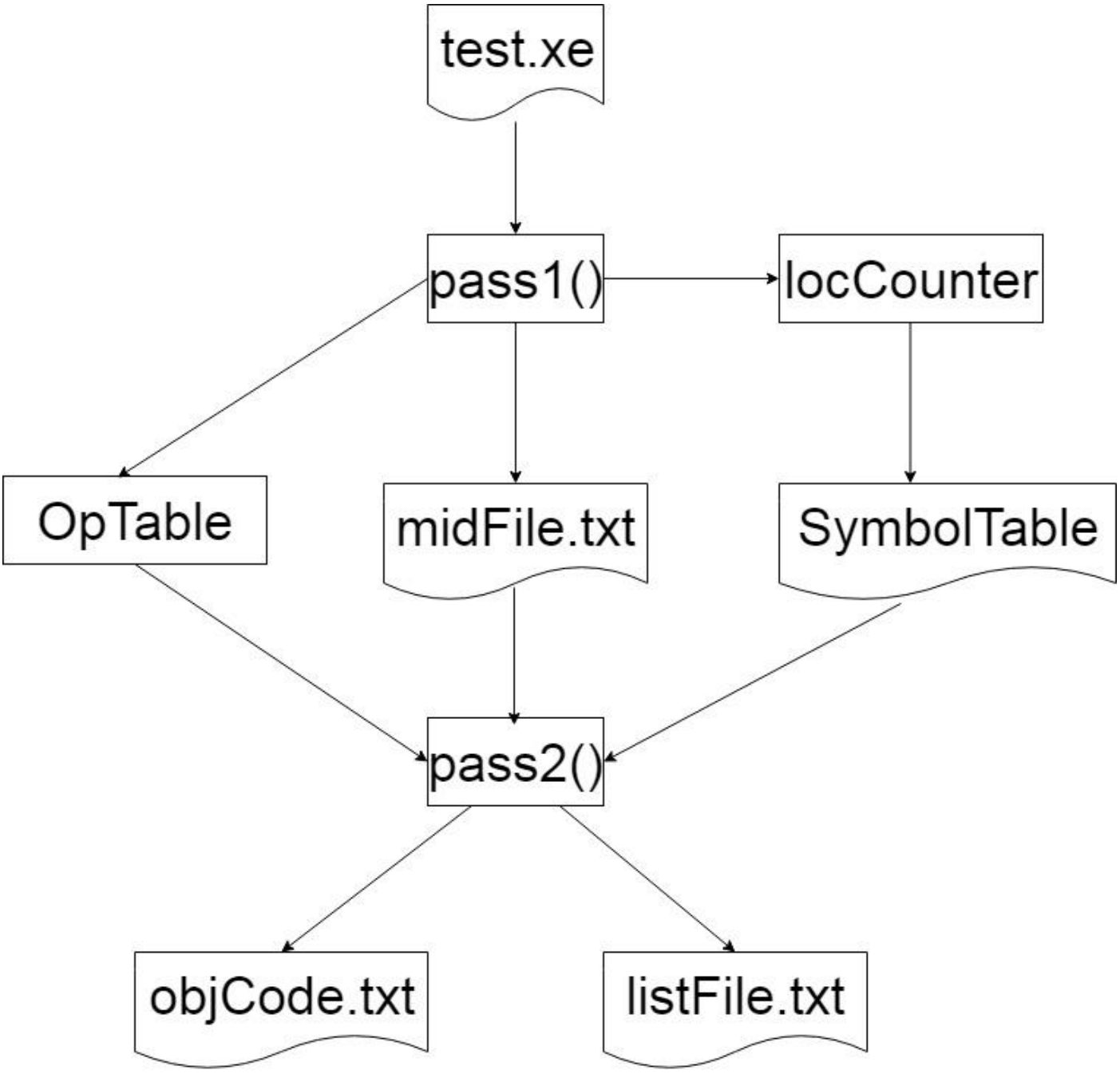
(3) Read input line as string input (so that **make sure your source assembly code follows our format and intend**).

Source statement	Show paragraph marks and other hidden formatting symbols
PROG START 1000 CLEAR T END	PROG→START→1000¶ → CLEAR→T¶ → END¶

(4) Comment in assembly code after operand will be ignored.
Ex: ABC LDX #3 ~this is comment~

opTable[][]	[0]	[1]	[2]	X
Data represent	Mnemonic Opcode	Instruction Format	Machine Code	Comment
Example	ADD	3	18	X
	ADDF	3	58	
	CLEAR	2	B4	

Flow Chart


test.xe
pass1()
locCounter
OpTable
midFile.txt
SymbolTable
pass2()
objCode.txt
listFile.txt

Data Structure

- Operation Code Table - Class OpTable
  - Implemented by 3-Dimension Array
  - Example:

```
opTable[0][0][0] = "ADD";     →   Mnemonic Opcode
opTable[0][0][1] = "3";       →   Format 3 Instruction
opTable[0][0][2] = "18";      →   Machine Code
```

opTable[][]	[0]	[1]	[2]
Data represent	Mnemonic Opcode	Instruction Format	Machine Code
Example	ADD	3	18
	ADDF	3	58
	CLEAR	2	B4

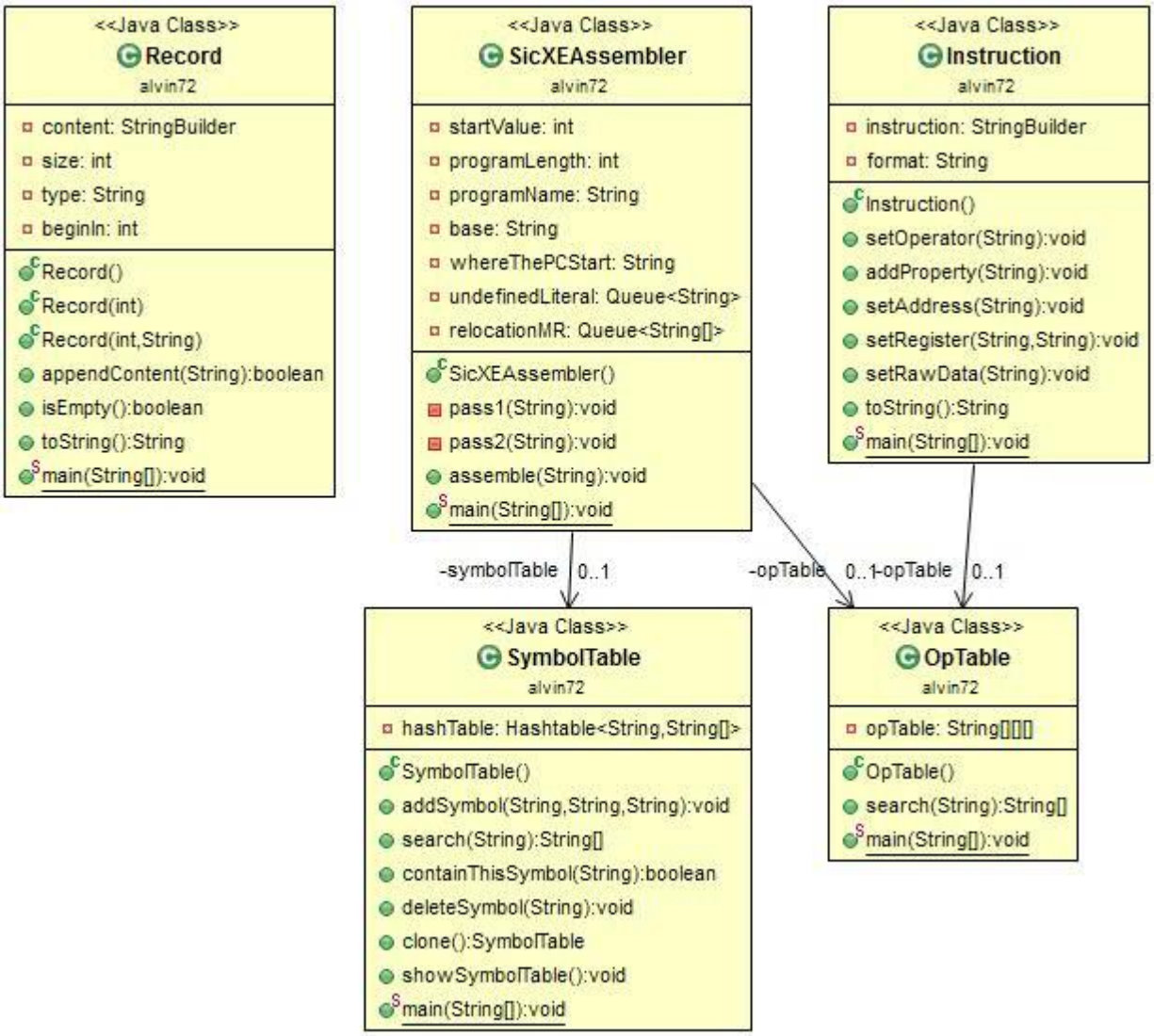
- Symbol Table - Class SymbolTable
  - Implemented by Hashtable and 1-Dimension Array
  - Example:

```
private Hashtable<String, String[]> hashTable;

String[] inputCopy = new String[3];
inputCopy[0] = new String(inputName);
inputCopy[1] = new String(value);
inputCopy[2] = new String(type);

hashTable.put(new String(inputName), inputCopy);
```

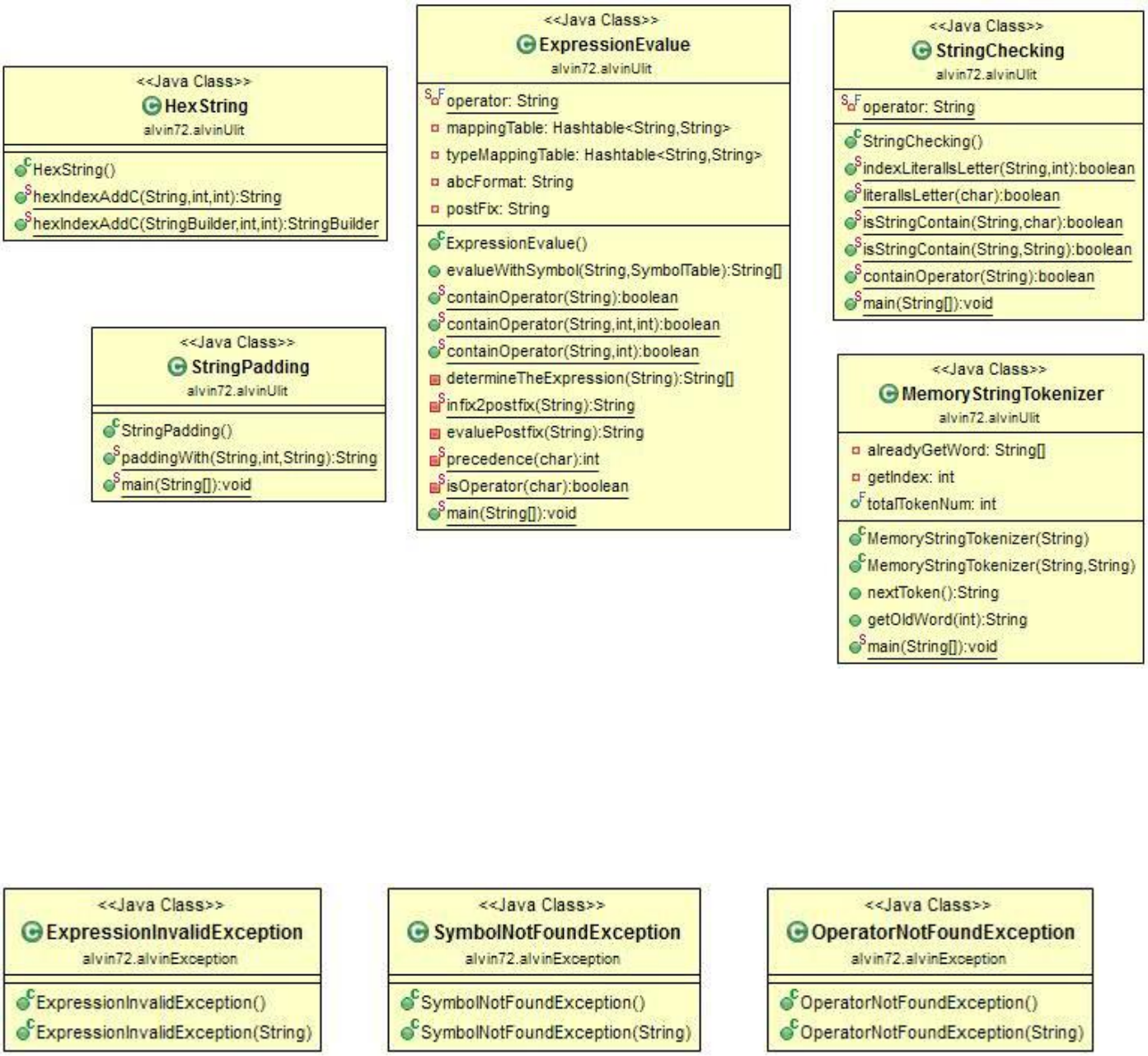
inputCopy	[0]	[1]	[2]
Variable name in method definition	inputName	value	type
Represent	Label Name	Address	A: Absolute/ R: Relative
Example	COPY EQU 8		
	COPY	0000	A

- Location Counter - locCounter
  - locCounter is a variable accumulated for address assignment
  - is initialized to be the beginning address specified in the START statement
  - after each source statement is processed during pass 1, instruction length or data area is added to locCounter

Class Diagram



Class Diagram



Test Case

We write a main method in almost all the class definition to test the function of that class. So, you may ignore the main methods in other classes, only the main method in SicXEAssembler class will make changes to those output files. In addition, we prepare some assembly code for testing purpose to make sure that it works what we think it should be.

Test Function		Checked?	Reference
Instruction Format	1	✔	
	2	✔	
	3	✔	
	4	✔	
Program Relocation		✔	
Addressing Modes	Index	✔	
	Immediate	✔	
	Indirect	✔	
	Direct	✔	
	PC Relative	✔	
	Base Relative	✔	
Assembler Directives	START	✔	
	END	✔	
	RESB	✔	
	RESW	✔	
	BYTE	✔	
	WORD	✔	
Literals	=C’	✔	
	=x’	✔	
	Duplicate literals	✔	
	LTORG - literal pool	✔	
	END - literal pool	✔	
Symbol-Defining Statement	EQU	✔	
	ORG	✔	
Expression			
Relative	Label name	✔	
	*(current location)	✔	
	Error(-R, R+R, 3*R, R/2)	✔	
Absolute	Constant	✔	
	+,-,*,/	✔	
	(,)	✔	
Object Code	H	✔	
	T	✔	
	T > 30 bytes	✔	
	T > 30 bytes(break on literal)	✔	
	M	✔	
	E	✔	

	Assembly Program		List File	
	PROG START 1000 FLOAT 3 CLEAR T LDA B +LDB #4096 END		line 1|location: 0000| |PROG START 1000 line 2|location: 1000|C0 | FLOAT 3 line 3|location: 1001|B450 | CLEAR T line 4|location: 1003|030000 | LDA B line 5|location: 1006|69101000| +LDB #4096 line 6|location: 100A| | END	→program relocation →instruction format 1 →instruction format 2 →instruction format 3 →instruction format 4
	PROG START 0 LDA TAG,X TAG LDB #8 LDT @TAG LDX TAG END		line 1|location: 0000| |PROG START 0 line 2|location: 0000|03A000 | LDA TAG,X line 3|location: 0003|690008 |TAG LDB #8 line 4|location: 0006|762FFA |LDT @TAG line 5|location: 0009|072FF7 |LDX TAG line 6|location: 000C| | END	→index addressing mode →immediate addressing mode →indirect addressing mode →direct addressing mode
	PROG START 0 LDB #VBASE BASE VBASE LDA TAGPC LDB TAGZ TAGPC LDT #8 VBASE RESW 1 TAGB RESB 4096 TAGZ BYTE X'12' WORD 32 END		line 1|location: 0000| |PROG START 0 line 2|location: 0000|692009 | LDB #VBASE line 3|location: 0003| | BASE VBASE line 4|location: 0003|032003 | LDA TAGPC line 5|location: 0006|6B4003 | LDB TAGZ line 6|location: 0009|750008 |TAGPC LDT #8 line 7|location: 000C| |VBASE RESW 1 line 8|location: 000F| |TAGB RESB 4096 line 9|location: 100F|12 |TAGZ BYTE X'12' line 10|location: 1010|000020 | WORD 32 line 11|location: 1013| | END	→assembler directive: START →PC-relative →Base-relative →assembler directive: RESW →assembler directive: RESB →assembler directive: BYTE →assembler directive: WORD →assembler directive: END
	PROG START 0 LDA =C'ABC' XYZ LDB =X'1234' LDX =C'ABC' END		line 1|location: 0000| |PROG START 0 line 2|location: 0000|032006 | LDA =C'ABC' line 3|location: 0003|6B2006 |XYZ LDB =X'1234' line 4|location: 0006|072000 | LDX =C'ABC' line 5|location: 0009| |414243 END -> LTORG line 5|location: 000C|1234 | END	→literal =C' →literal =X' →duplicate literal

	Assembly Program	Object Code
	PROG START 0 LDA =C'111' A002 LDA =C'EO4' LDA =C'107' LTORG LDA =C'ABC' BYTE X'123456' WORD 16 END	HPROG 00000000001E T0000001E032006032006032006313131454F34313037 032006123456000010414243 E000000 →able to build literal pool at LTORG statement → able to build literal pool at END statement

	Assembly Program			Console	
	PROG	START	0	|Name: A |Value: 2 |Type: R|	→EQU is functional
		CLEAR	T	|Name: PROG |Value: 0 |Type: R|	→R: Relative
	A	EQU	*	|Name: C |Value: 8 |Type: A|	→*(current loc) is functional,
	B	EQU	A-1	|Name: B |Value: 1 |Type: R|	A: Absolute
	C	EQU	8		
		END			

	Assembly Program			List File					HTE Object Code
	PGO	START	2000	line 1	location: 0000		PGO	START	2000	HPGO 002000000013
		STA	AAA	line 2	location: 2000	0F2003	STA	AAA		T002000060F200303A000
		LDA	BBB,X	line 3	location: 2003	03A000		LDA	BBB,X	T002006 03010003
	AAA	RESW	3	line 4	location: 2006		AAA	RESW	3	T00200F 045A000010
		ORG	AAA	line 5	location: 200F			ORG	AAA	E00200C
	BBB	lda	#3	line 6	location: 2006	010003	BBB	LDA	#3	
		ORG		line 7	location: 2009			ORG		→ORG is functional
	CCC	BYTE	X'5A'	line 8	location: 200F	5A	CCC	BYTE	X'5A'	
	DDD	WORD	16	line 9	location: 2010	000010	DDD	WORD	16	
		END	AAA+6	line 10	location: 2013			END	AAA+6	

	Assembly Program			List File
	PGE	START	0	|Name: M0 |Value: 32 |Type: A|
	CC	EQU	3→A	|Name: J0 |Value: 9 |Type: R|
		WORD	3	|Name: BB |Value: 6 |Type: R|
	AA	LDA	#3	|Name: G0 |Value: 9 |Type: R|
	BB	J	AA	|Name: A0 |Value: 3 |Type: R|
	DD	EQU	*	|Name: DD |Value: 9 |Type: R|
	EE	STA	AA,X	|Name: L0 |Value: E |Type: R|
	A0	EQU	DD-6	|Name: F0 |Value: 67 |Type: A|
	B0	EQU		|Name: AA |Value: 3 |Type: R|
		CC+EE		|Name: C0 |Value:FFFFFFFD |Type: A|
	C0	EQU	BB-EE	|Name: PGE |Value: 0 |Type: R|
	.D0	EQU	CC-BB→A-R	|Name: CC |Value: 3 |Type: A|
	.E0	EQU		|Name: K0 |Value: 9 |Type: A|
		DD+BB	→R+R	|Name: B0 |Value: C |Type: R|
	F0	EQU		|Name: EE |Value: 9 |Type: R|
		CC+100		→R-R+R-A=R
	G0	EQU	DD-AA+BB-CC	→3*R
	.H0	EQU	3*AA	→A-R
	.I0	EQU	100-AA	→(R-R)+(R-A)=R
	J0	EQU	(DD-AA)+(BB-CC)	→ +, -
	K0	EQU	3*4-6+CC	→ (,)
	L0	EQU		→*, /
		DD+(1+8/2)		**Invalid expression need to be comment line because our program does not handle error message for this kind of situation.**
	M0	EQU		
		1000*2/5/8		
		END		

	Assembly Program	Object Code
	PROGL START 0	HPROGL 00000000006D
	LDA =C'111'	T0000001B03203A03203A03203A03203A03203A03203A03203A03203A03203A
	LDA =C'EO1'	T00001B1C05101000032033032033032033032033032033032033032033032033
	LDA =C'EO2'	T0000371E12345600001031313145 4F31454F32454F33454F34313037454F35454F36
	LDA =C'EO3'	T00005518454F37313038454F39454F304541314541323130394142 43
	A002 LDA =C'EO4'	E000000
	LDA =C'107'	
	LDA =C'EO5'	
	LDA =C'EO6'	
	LDA =C'EO7'	→able to handle object code that > 30 bytes
	+LDX #4096	
	LDA =C'EO7'	
	LDA =C'108'	
	LDA =C'EO9'	
	LDA =C'EO0'	
	LDA =C'EA1'	
	LDA =C'EA2'	
	LDA =C'109'	
	LDA =C'ABC'	
	BYTE X'123456'	
	WORD 16	
	END	→able to handle literal pool that > 30 bytes

	Assembly Program	Object Code	
	PROG START 0	HPROG 000000000007	→H Record
	ABC LDA #8	T0000000701000807100000	→T Record
	+LDX ABC	M00000405+PROG	→M Record, program relocation
	END	E000000	→E Record