



System Software CSN-252 Assembler Design



Assemblers



- Features that are fundamental and should be found in any Assembler
- Features closely related to the machine architecture
- Features that are commonly found in implementations of this type of software and are relatively machine-independent

- Fundamental functions that any assembler must perform
 - Translating mnemonic operation codes to their machine language equivalents
 - Assigning machine addresses to symbolic labels used by programmers
- Features of the assembler that depend heavily on the source language it translates and the machine language it produces
 - Different machine instruction formats
 - Different addressing modes
- Features that have no direct relation to machine architecture
 - More than one program blocks, control sections etc., use of literals

Assembler directives

START	Specify name and starting address for the program
END	Indicates the end of the program and (optionally) specify the first executable instruction in the program
BYTE	Generate character or hex constant, occupying as many bytes as needed to represent the constant
WORD	generate one word integer constant
RESB	reserve the indicated number of bytes for a data area
RESW	reserve the indicated number of words for a data area

```
MOVE START 1000
```

```
FIRST LDX ZERO
```

```
MOVCH LDCH STR1, X
```

```
STCH STR2, X
```

```
TIX ELEVEN
```

```
JLT MOVCH
```

```
RSUB
```

```
STR1 BYTE C'TEST STRING'
```

```
STR2 RESB 11
```

```
ZERO WORD 0
```

```
ELEVEN WORD 11
```

```
END FIRST
```

The translation of the source program to object code requires

0. Process assembler directives
1. Convert mnemonic operation codes to their machine language equivalents
2. Convert symbolic operands to their equivalent machine addresses

```
MOVE START 1000
```

```
FIRST LDX ZERO
```

```
MOVCH LDCH STR1, X
```

```
STCH STR2, X
```

```
TIX ELEVEN
```

```
JLT MOVCH
```

```
RSUB
```

```
STR1 BYTE C'TEST STRING'
```

```
STR2 RESB 11
```

```
ZERO WORD 0
```

```
ELEVEN WORD 11
```

```
END FIRST
```

The translation of the source program to object code requires

3. Build the machine instructions in the proper format
4. Convert the data constants specified in the source program into their internal machine representations
5. Write the object program and the assembly listing

- Can we do all of the above by sequential processing of the source program?
- forward references
- Therefore most assemblers make two passes over the source program.
 - Pass 1: scan the source program for label definitions and assign addresses
 - Pass 2: perform actual translation and generate object code

SIC Assembler V1.2

1000	test	start	1000
1000	first	lda	five
1003		sta	alpha
1006		ldch	charz
1009		stch	c1
:		:	
100C	alpha	resw	1
100F	five	word	5
1012	charz	byte	c'Z'
1013	c1	resb	1
1014		end	first

In hex

In dec.

SIC Assembler V1.2

1000		test	start	1000
1000	00100F	first	lda	five
1003	0C100C		sta	alpha
1006	501012		ldch	charz
1009	541013		stch	c1
:	:		:	
100C		alpha	resw	1
100F	000005	five	word	5
1012	5A	charz	byte	c'Z'
1013		c1	resb	1
1014			end	first

Assemblers



- Translates the source program to object code.
- This requires
 0. Process **assembler directives**
 1. Convert mnemonic **operation codes** to their machine language equivalents
 2. Convert **symbolic operands** to their equivalent machine addresses
 3. Build the machine instructions in the proper format
 4. Convert the data constants specified in the source program into their internal machine representations
 5. Write the **object program** and the **assembly listing**

SIC Assembler (Pass 1 – Define Symbols)

Input: ?

- Assign addresses to all statements in the program
- Assign values to LABELs
- Save the values (addresses) assigned to all labels for use in Pass 2
- Perform some processing of assembler directives
- Record errors
- Generate intermediate file

Output: ?

Data Structures

- **Operation Code Table (OPTAB):**
 - mnemonic operation code and its machine language equivalent.
 - information about instruction format and length.
 - Usually organized as a HASH TABLE, with mnemonic operation code as the key.
 - OPTAB is a [static table](#)
- **Symbol table (SYMTAB):**
 - name and value (address) for each label together with flags to indicate error conditions
 - (May also contain) information about the instruction labeled
 - [Frequent insertions](#)
 - [rare deletions](#)
 - Usually organized as a HASH TABLE
- **Location Counter (LOCCTR)**

SIC Assembler (Pass 2 – Generate Object Program)

- Assemble instructions
 - Process OPCODE
 - Process LABELS
- Perform processing of assembler directives not done during pass 1
 - Generate data values defined by BYTE, WORD etc.
- Write the **object program** and the assembly listing.
- Report errors