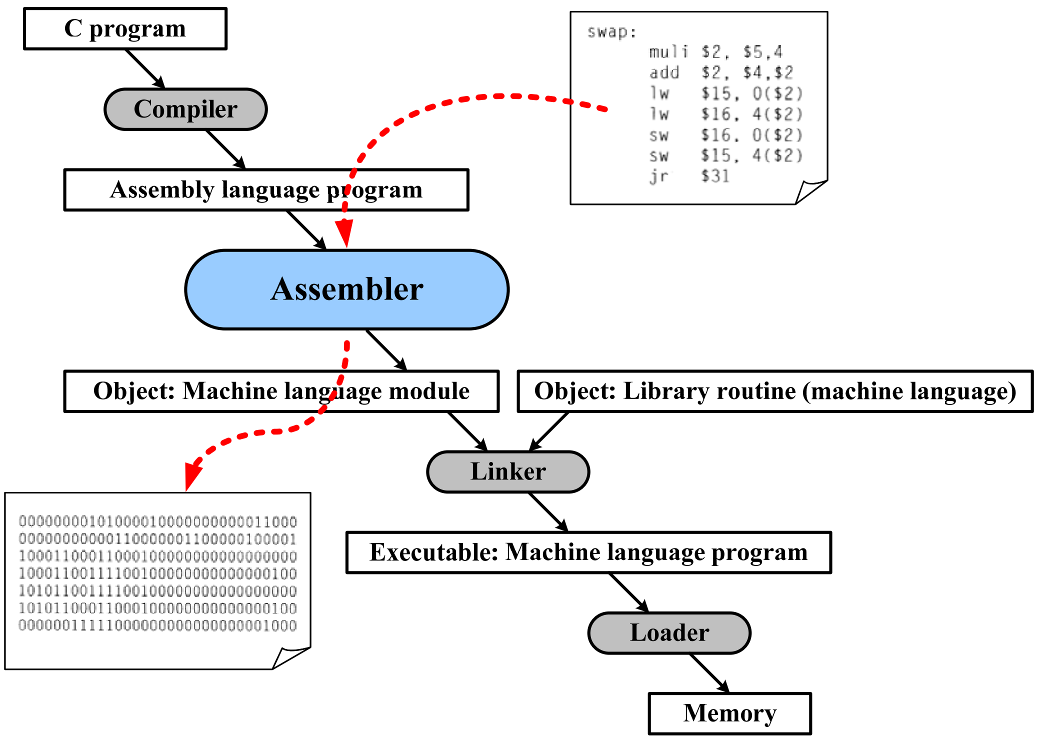
Chapter 2: Assemblers

王振傑 (Chen-Chieh Wang)
ccwang@mail.ee.ncku.edu.tw

Assemblers



Department of Electrical Engineering, Feng-Chia University

Outline

Department of Electrical Engineering, Feng-Chia University

Assembler Directives

- Assembler directives (or pseudo-instructions) are not translated into machine instructions. Instead, they provide instructions to the assembler itself.

START	Specify name and starting address for the program.
END	Indicate the end of the source program and (optionally) specify the first executable instruction in the program.
BYTE	Generate character or hexadecimal 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.

A SIC Assembly Program Example

[Figure 2.1]

Line Source statement

```
COPY     START   1000        COPY FILE FROM INPUT TO OUTPUT
FIRST    STL     RETADR      SAVE RETURN ADDRESS
CLOOP    JSUB    RDREC       READ INPUT RECORD
         LDA     LENGTH      TEST FOR EOF (LENGTH = 0)
         COMP    ZERO
         JEQ     ENDFIL      EXIT IF EOF FOUND
         JSUB    WRREC       WRITE OUTPUT RECORD
         J       CLOOP       LOOP
ENDFIL   LDA     EOF         INSERT END OF FILE MARKER
         STA     BUFFER
         LDA     THREE       SET LENGTH = 3
         STA     LENGTH
         JSUB    WRREC       WRITE EOF
         LDL     RETADR      GET RETURN ADDRESS
         RSUB                RETURN TO CALLER
EOF      BYTE    C'EOF'
THREE    WORD    3
ZERO     WORD    0
RETADR   RESW    1
LENGTH   RESW    1           LENGTH OF RECORD
BUFFER   RESB    4096        4096-BYTE BUFFER AREA
```


Department of Electrical Engineering, Feng-Chia University

SUBROUTINE TO READ RECORD INTO BUFFER

```
RDREC    LDX     ZERO        CLEAR LOOP COUNTER
         LDA     ZERO        CLEAR A TO ZERO
RLOOP    TD      INPUT       TEST INPUT DEVICE
         JEQ     RLOOP       LOOP UNTIL READY
         RD      INPUT       READ CHARACTER INTO REGISTER A
         COMP    ZERO        TEST FOR END OF RECORD (X'00')
         JEQ     EXIT        EXIT LOOP IF EOR
         STCH    BUFFER,X    STORE CHARACTER IN BUFFER
         TIX     MAXLEN      LOOP UNLESS MAX LENGTH
         JLT     RLOOP         HAS BEEN REACHED
EXIT     STX     LENGTH      SAVE RECORD LENGTH
         RSUB                RETURN TO CALLER
INPUT    BYTE    X'F1'       CODE FOR INPUT DEVICE
MAXLEN   WORD    4096
```

SUBROUTINE TO WRITE RECORD FROM BUFFER

```
WRREC    LDX     ZERO        CLEAR LOOP COUNTER
WLOOP    TD      OUTPUT      TEST OUTPUT DEVICE
         JEQ     WLOOP       LOOP UNTIL READY
         LDCH    BUFFER,X    GET CHARACTER FROM BUFFER
         WD      OUTPUT      WRITE CHARACTER
         TIX     LENGTH      LOOP UNTIL ALL CHARACTERS
         JLT     WLOOP         HAVE BEEN WRITTEN
         RSUB                RETURN TO CALLER
OUTPUT   BYTE    X'05'       CODE FOR OUTPUT DEVICE
         END     FIRST
```

Outline

A Simple SIC Assembler

1. Convert mnemonic operation codes to their machine language equivalents. (e.g., translate STL to 14)
2. Convert symbolic operands to their equivalent machine addresses. (e.g., translate RETADR to 1033)
3. Build the machine instructions in the proper format.
4. Convert the data constants specified in the source program into their internal machine representations. (e.g., translate EOF to 454F46)
5. Write the object program and the assembly listing.

All of these functions except number 2 can easily be accomplished by sequential processing of the source program, one line at a time.

Program from Figure 2.1 with Object Code

[Figure 2.2]

Line	Loc	Source statement			Object code
5	1000	COPY	START	1000	
10	1000	FIRST	STL	RETADR	141033
15	1003	CLOOP	JSUB	RDREC	482039
20	1006		LDA	LENGTH	001036
25	1009		COMP	ZERO	281030
30	100C		JEQ	ENDFIL	301015
35	100F		JSUB	WRREC	482061
40	1012		J	CLOOP	3C1003
45	1015	ENDFIL	LDA	EOF	00102A
50	1018		STA	BUFFER	0C1039
55	101B		LDA	THREE	00102D
60	101E		STA	LENGTH	0C1036
65	1021		JSUB	WRREC	482061
70	1024		LDL	RETADR	081033
75	1027		RSUB		4C0000
80	102A	EOF	BYTE	C'EOF'	454F46
85	102D	THREE	WORD	3	000003
90	1030	ZERO	WORD	0	000000
95	1033	RETADR	RESW	1	
100	1036	LENGTH	RESW	1	
105	1039	BUFFER	RESB	4096	

SUBROUTINE TO READ RECORD INTO BUFFER

125	2039	RDREC	LDX	ZERO	041030
130	203C		LDA	ZERO	001030
135	203F	RLOOP	TD	INPUT	E0205D
140	2042		JEQ	RLOOP	30203F
145	2045		RD	INPUT	D8205D
150	2048		COMP	ZERO	281030
155	204B		JEQ	EXIT	302057
160	204E		STCH	BUFFER,X	549039
165	2051		TIX	MAXLEN	2C205E
170	2054		JLT	RLOOP	38203F
175	2057	EXIT	STX	LENGTH	101036
180	205A		RSUB		4C0000
185	205D	INPUT	BYTE	X'F1'	F1
190	205E	MAXLEN	WORD	4096	001000

SUBROUTINE TO WRITE RECORD FROM BUFFER

210	2061	WRREC	LDX	ZERO	041030
215	2064	WLOOP	TD	OUTPUT	E02079
220	2067		JEQ	WLOOP	302064
225	206A		LDCH	BUFFER,X	509039
230	206D		WD	OUTPUT	DC2079
235	2070		TIX	LENGTH	2C1036
240	2073		JLT	WLOOP	382064
245	2076		RSUB		4C0000
250	2079	OUTPUT	BYTE	X'05'	05
255			END	FIRST	

Object Program Format

	Column	Contents
Header Record	1	H
	2-7	Program name
	8-13	Starting address of object program (HEX)
	14-19	Length of object program in bytes (HEX)
Text Record	1	T
	2-7	Starting address for object code in this record (HEX)
	8-9	Length of object code in this record in bytes (HEX)
	10-69	Object code (HEX, 2 columns per byte of object code)
End Record	1	E
	2-7	Address of first executable instruction in object program (HEX)

Object Program (Corresponding to Figure 2.2)

[Figure 2.3]

```
H^COPY  ^001000^00107A
T^001000^1E^141033^482039^001036^281030^301015^482061^3C1003^00102A^0C1039^00102D
T^00101E^15^0C1036^482061^081033^4C0000^454F46^000003^000000
T^002039^1E^041030^001030^E0205D^30203F^D8205D^281030^302057^549039^2C205E^38203F
T^002057^1C^101036^4C0000^F1^001000^041030^E02079^302064^509039^DC2079^2C1036
T^002073^07^382064^4C0000^05
E^001000
```

Note that there is no object code corresponding to addressed 1033-2038. This storage is simply reserved by the loader for use by the program during execution.

Department of Electrical Engineering, Feng-Chia University

Two-pass SIC Assembler

Pass 1 (define symbols):

1. Assign **addresses** to all statements in the program.
2. Save the values (addresses) assigned to all **labels** for use in Pass 2.
3. Perform some processing of **assembler directives**. (This includes processing that affects address assignment, such as determining the length of data areas defined by BYTE, RESW, etc.)
4. Write **intermediate file**.

Pass 2 (assemble instructions and generate object program):

1. Assemble **instructions** (translating operation codes and looking up addresses).
2. Generate **data values** defined by BYTE, WORD, etc.
3. Perform processing of **assembler directives** not done during Pass1.
4. Write the **object program** and the **assembly listing**.

Department of Electrical Engineering, Feng-Chia University

Outline

2.1 Basic Assembler Functions
- 2.1.1 A Simple SIC Assembler
- 2.1.2 Assembler Algorithm and Data Structures

2.2 Machine-Dependent Assembler Features
- 2.2.1 Instruction Formats and Addressing Modes
- 2.2.2 Program Relocation

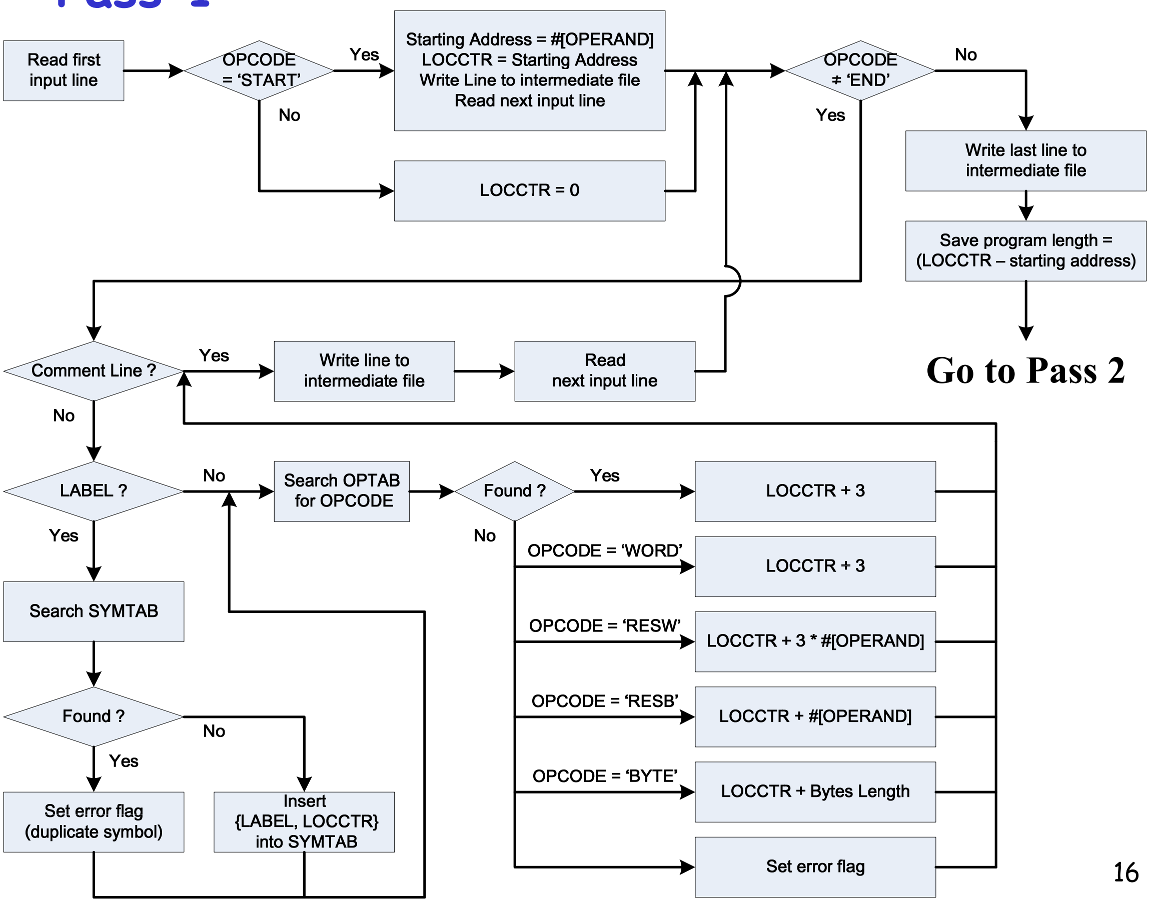
2.3 Machine-Independent Assembler Features
- 2.3.1 Literals
- 2.3.2 Symbol-Defining Statements
- 2.3.3 Expressions
- 2.3.4 Program Blocks
- 2.3.5 Control Sections and Program Linking

2.4 Assembler Design Options
- 2.4.1 One-Pass Assemblers
- 2.4.2 Multi-Pass Assemblers

Data Structures

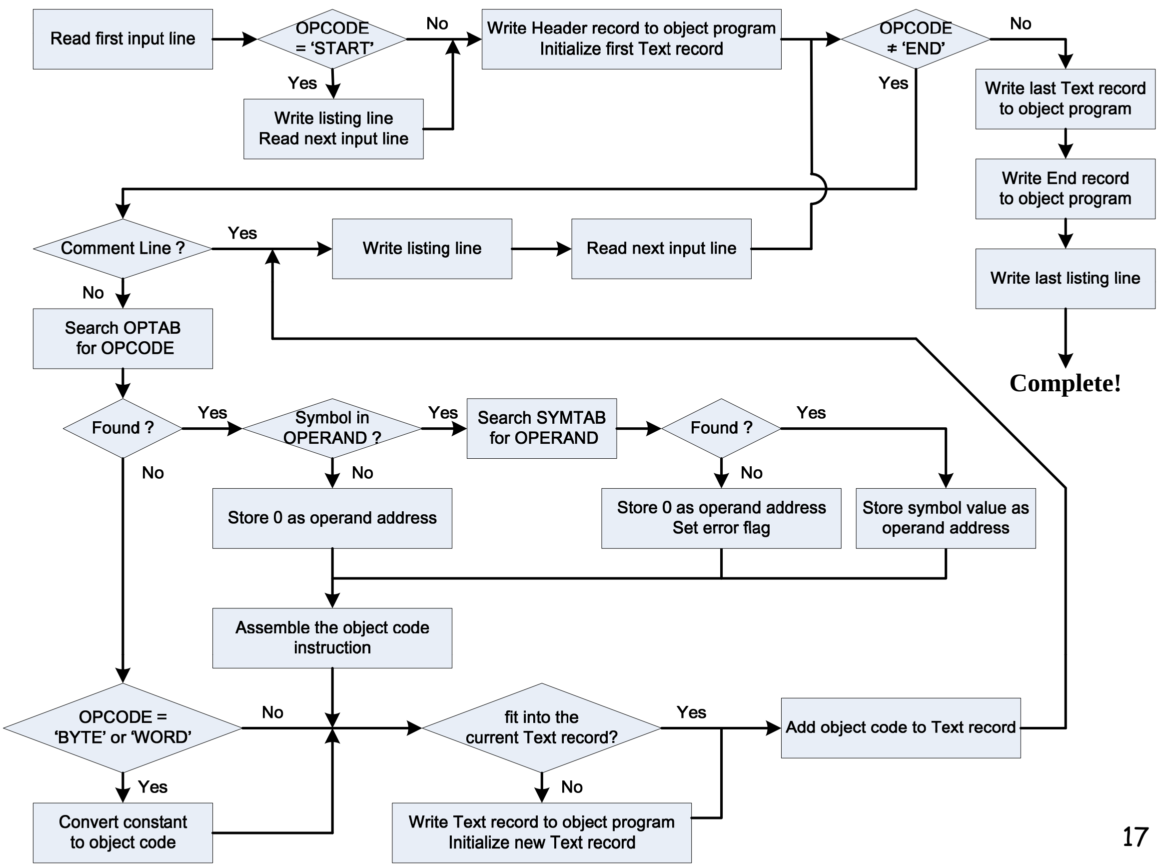
- **Operation Code Table (OPTAB)** is used to look up mnemonic operation codes and translate them to their machine language equivalents.
- **Symbol Table (SYMTAB)** is used to store values (addresses) assigned to labels.
- **Location Counter (LOCCTR)** is a variable that is used to help in the assignment of addresses.

Pass 1



Department of Electrical Engineering, Feng-Chia University

Pass 2



Department of Electrical Engineering, Feng-Chia University

Outline

- 2.1 Basic Assembler Functions
  - 2.1.1 A Simple SIC Assembler
  - 2.1.2 Assembler Algorithm and Data Structures
- 2.2 Machine-Dependent Assembler Features
  - 2.2.1 Instruction Formats and Addressing Modes
  - 2.2.2 Program Relocation
- 2.3 Machine-Independent Assembler Features
  - 2.3.1 Literals
  - 2.3.2 Symbol-Defining Statements
  - 2.3.3 Expressions
  - 2.3.4 Program Blocks
  - 2.3.5 Control Sections and Program Linking
- 2.4 Assembler Design Options
  - 2.4.1 One-Pass Assemblers
  - 2.4.2 Multi-Pass Assemblers

A SIC/XE Assembly Program Example

SIC (Figure 2.1)

Line	Source statement		
5	COPY	START	1000
10	FIRST	STL	RETADR
15	CLOOP	JSUB	RDREC
20		LDA	LENGTH
25		COMP	ZERO
30		JEQ	ENDFIL
35		JSUB	WRREC
40		J	CLOOP
45	ENDFIL	LDA	EOF
50		STA	BUFFER
55		LDA	THREE
60		STA	LENGTH
65		JSUB	WRREC
70		LDL	RETADR
75		RSUB	
80	EOF	BYTE	C'EOF'
85	THREE	WORD	3
90	ZERO	WORD	0
95	RETADR	RESW	1
100	LENGTH	RESW	1
105	BUFFER	RESB	4096

SIC/XE (Figure 2.5)

Line	Source statement		
5	COPY	START	0
10	FIRST	STL	RETADR
12		LDB	#LENGTH
13		BASE	LENGTH
15	CLOOP	+JSUB	RDREC
20		LDA	LENGTH
25		COMP	#0
30		JEQ	ENDFIL
35		+JSUB	WRREC
40		J	CLOOP
45	ENDFIL	LDA	EOF
50		STA	BUFFER
55		LDA	#3
60		STA	LENGTH
65		+JSUB	WRREC
70		J	@RETADR
80	EOF	BYTE	C'EOF'
95	RETADR	RESW	1
100	LENGTH	RESW	1
105	BUFFER	RESB	4096

SUBROUTINE TO READ RECORD INTO BUFFER

125	RDREC	LDX	ZERO
130		LDA	ZERO
135	RLOOP	TD	INPUT
140		JEQ	RLOOP
145		RD	INPUT
150		COMP	ZERO
155		JEQ	EXIT
160		STCH	BUFFER,X
165		TIX	MAXLEN
170		JLT	RLOOP
175	EXIT	STX	LENGTH
180		RSUB	
185	INPUT	BYTE	X'F1'
190	MAXLEN	WORD	4096

125	RDREC	CLEAR	X
130		CLEAR	A
132		CLEAR	S
133		+LDT	#4096
135	RLOOP	TD	INPUT
140		JEQ	RLOOP
145		RD	INPUT
150		COMPR	A,S
155		JEQ	EXIT
160		STCH	BUFFER,X
165		TIXR	T
170		JLT	RLOOP
175	EXIT	STX	LENGTH
180		RSUB	
185	INPUT	BYTE	X'F1'

SUBROUTINE TO WRITE RECORD FROM BUFFER

210	WRREC	LDX	ZERO
215	WLOOP	TD	OUTPUT
220		JEQ	WLOOP
225		LDCH	BUFFER,X
230		WD	OUTPUT
235		TIX	LENGTH
240		JLT	WLOOP
245		RSUB	
250	OUTPUT	BYTE	X'05'
255		END	FIRST

210	WRREC	CLEAR	X
212		LDT	LENGTH
215	WLOOP	TD	OUTPUT
220		JEQ	WLOOP
225		LDCH	BUFFER,X
230		WD	OUTPUT
235		TIXR	T
240		JLT	WLOOP
245		RSUB	
250	OUTPUT	BYTE	X'05'
255		END	FIRST

Department of Electrical Engineering, Feng-Chia University

Outline

Department of Electrical Engineering, Feng-Chia University

Program from Figure 2.5 with Object Code

Line	Loc	Source statement			Object code
5	0000	COPY	START	0	
10	0000	FIRST	STL	RETADR	17202D
12	0003		LDB	#LENGTH	69202D
13			BASE	LENGTH	
15	0006	CLOOP	+JSUB	RDREC	4B101036
20	000A		LDA	LENGTH	032026
25	000D		COMP	#0	290000
30	0010		JEQ	ENDFIL	332007
35	0013		+JSUB	WRREC	4B10105D
40	0017		J	CLOOP	3F2FEC
45	001A	ENDFIL	LDA	EOF	032010
50	001D		STA	BUFFER	0F2016
55	0020		LDA	#3	010003
60	0023		STA	LENGTH	0F200D
65	0026		+JSUB	WRREC	4B10105D
70	002A		J	@RETADR	3E2003
80	002D	EOF	BYTE	C'EOF'	454F46
95	0030	RETADR	RESW	1	
100	0033	LENGTH	RESW	1	
105	0036	BUFFER	RESB	4096	

SUBROUTINE TO READ RECORD INTO BUFFER

125	1036	RDREC	CLEAR	X	B410
130	1038		CLEAR	A	B400
132	103A		CLEAR	S	B440
133	103C		+LDT	#4096	75101000
135	1040	RLOOP	TD	INPUT	E32019
140	1043		JEQ	RLOOP	332FFA
145	1046		RD	INPUT	DB2013
150	1049		COMPR	A,S	A004
155	104B		JEQ	EXIT	332008
160	104E		STCH	BUFFER,X	57C003
165	1051		TIXR	T	B850
170	1053		JLT	RLOOP	3B2FEA
175	1056	EXIT	STX	LENGTH	134000
180	1059		RSUB		4F0000
185	105C	INPUT	BYTE	X'F1'	F1

SUBROUTINE TO WRITE RECORD FROM BUFFER

210	105D	WRREC	CLEAR	X	B410
212	105F		LDT	LENGTH	774000
215	1062	WLOOP	TD	OUTPUT	E32011
220	1065		JEQ	WLOOP	332FFA
225	1068		LDCH	BUFFER,X	53C003
230	106B		WD	OUTPUT	DF2008
235	106E		TIXR	T	B850
240	1070		JLT	WLOOP	3B2FEF
245	1073		RSUB		4F0000
250	1076	OUTPUT	BYTE	X'05'	05
255			END	FIRST	

Addressing

- Program-Counter relative addressing (Format 3)
  - -2048 ≤ Displacement ≤ +2047
- Base relative addressing (Format 3)
  - 0 ≤ Displacement ≤ 4095
- Extended instruction format (Format 4)
  - 20-bit address field, which is large enough to contain the full memory address.
  - Using the prefix +

Department of Electrical Engineering, Feng-Chia University

Example 1: PC Relative Addressing

Line	Loc	Source statement			Object code
10	0000	FIRST	STL	RETADR	17202D

Hex	Binary		
	op	n i x b p e	disp/address
1 7 2 0 2 D	0 0 0 1 0 1	1 1 0 0 1 0	0 0 0 0 0 0 1 0 1 1 0 1

Department of Electrical Engineering, Feng-Chia University

Example 2: PC Relative Addressing

Line	Loc	Source statement			Object code
40	0017		J	CLOOP	3F2FEC

Hex	Binary		
	op	n i x b p e	disp/address
3 F 2 F E C	0 0 1 1 1 1	1 1 0 0 1 0	1 1 1 1 1 1 1 0 1 1 0 0

Department of Electrical Engineering, Feng-Chia University

Base Relative Addressing

- Difference between PC-relative and Base-relative addressing
  - The assembler knows what the contents of the Program Counter will be at execution time. Base register ?
  - The base register is under control of the programmer. Therefore, the programmer must tell the assembler what the base register will contain during execution of the program
- Assembler Directives

BASE	Informs the assembler that the base register will contain the *address* of #[Operand]
NOBASE	Informs the assembler that the contents of the base register can no longer be relied upon for addressing

Department of Electrical Engineering, Feng-Chia University

Example : Base Relative Addressing

Line	Loc	Source statement		Object code
160	104E	STCH	BUFFER, X	57C003

Hex	Binary	
	op	n i x b p e	disp/address
5 7 C 0 0 3	0 1 0 1 0 1	1 1 1 1 0 0	0 0 0 0 0 0 0 0 0 0 1 1

Department of Electrical Engineering, Feng-Chia University

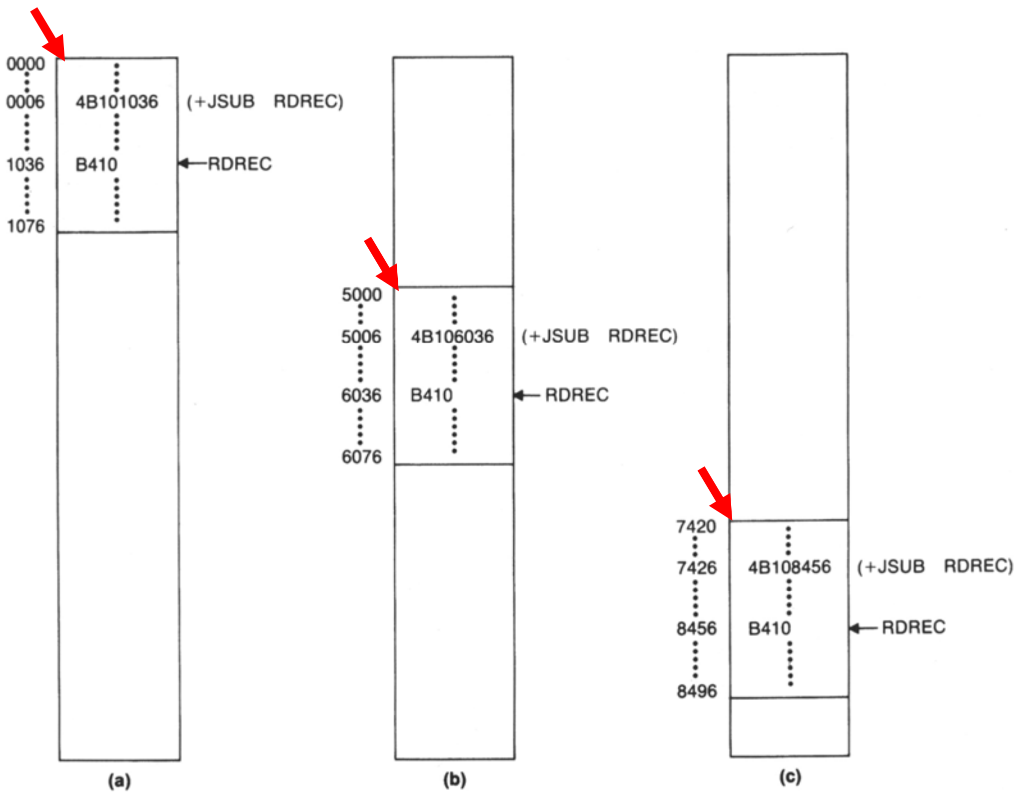
Outline

Department of Electrical Engineering, Feng-Chia University

Program Relocation

- **Multiprogramming**
  - Running multiple programs (processes) that share system resources (e.g. memory, CPU)
- **Absolute Programs**
  - Must be loaded at exact address in order to execute properly
- **Relocatable Programs**
  - Can be loaded into memory wherever these is room, rather than specifying a fixed address at assembly time

Examples of Program Relocation



Department of Electrical Engineering, Feng-Chia University

Object Program Format

	Column	Contents
Header Record	1	H
	2-7	Program name
	8-13	Starting address of object program (HEX)
	14-19	Length of object program in bytes (HEX)
Text Record	1	T
	2-7	Starting address for object code in this record (HEX)
	8-9	Length of object code in this record in bytes (HEX)
	10-69	Object code (HEX, 2 columns per byte of object code)
Mod. Record	1	M
	2-7	Starting location of the address field to be modified, relative to the beginning of the program (HEX)
	8-9	Length of the address field to be modified, in half-bytes (HEX)
End Record	1	E
	2-7	Address of first executable instruction (HEX)

Department of Electrical Engineering, Feng-Chia University

Object program
(Corresponding to Figure 2.6)

[Figure 2.8]

```
H^COPY  ^000000^001077
T^000000^1D^17202D^69202D^4B101036^032026^290000^332007^4B10105D^3F2FEC^032010
T^00001D^13^0F2016^010003^0F200D^4B10105D^3E2003^454F46
T^001036^1D^B410^B400^B440^75101000^E32019^332FFA^DB2013^A004^332008^57C003^B850
T^001053^1D^3B2FEA^134000^4F0000^F1^B410^774000^E32011^332FFA^53C003^DF2008^B850
T^001070^07^3B2FEF^4F0000^05
M^000007^05
M^000014^05
M^000027^05
E^000000
```


Department of Electrical Engineering, Feng-Chia University

Outline

Additional assembler features

Figure 2.5

Line	Source statement		
5	COPY	START	0
10	FIRST	STL	RETADR
12		LDB	#LENGTH
13		BASE	LENGTH
15	CLOOP	+JSUB	RDREC
20		LDA	LENGTH
25		COMP	#0
30		JEQ	ENDFIL
35		+JSUB	WRREC
40		J	CLOOP
45	ENDFIL	LDA	EOF
50		STA	BUFFER
55		LDA	#3
60		STA	LENGTH
65		+JSUB	WRREC
70		J	@RETADR
80	EOF	BYTE	C'EOF'
95	RETADR	RESW	1
100	LENGTH	RESW	1
105	BUFFER	RESB	4096

Figure 2.9

Line	Source statement		
5	COPY	START	0
10	FIRST	STL	RETADR
13		LDB	#LENGTH
14		BASE	LENGTH
15	CLOOP	+JSUB	RDREC
20		LDA	LENGTH
25		COMP	#0
30		JEQ	ENDFIL
35		+JSUB	WRREC
40		J	CLOOP
45	ENDFIL	LDA	=C'EOF'
50		STA	BUFFER
55		LDA	#3
60		STA	LENGTH
65		+JSUB	WRREC
70		J	@RETADR
93		LTORG	
95	RETADR	RESW	1
100	LENGTH	RESW	1
105	BUFFER	RESB	4096
106	BUFEND	EQU	*
107	MAXLEN	EQU	BUFEND-BUFFER

Department of Electrical Engineering, Feng-Chia University

SUBROUTINE TO READ RECORD INTO BUFFER

125	RDREC	CLEAR	X
130		CLEAR	A
132		CLEAR	S
133		+LDT	#4096
135	RLOOP	TD	INPUT
140		JEQ	RLOOP
145		RD	INPUT
150		COMPR	A,S
155		JEQ	EXIT
160		STCH	BUFFER,X
165		TIXR	T
170		JLT	RLOOP
175	EXIT	STX	LENGTH
180		RSUB	
185	INPUT	BYTE	X'F1'

125	RDREC	CLEAR	X
130		CLEAR	A
132		CLEAR	S
133		+LDT	#MAXLEN
135	RLOOP	TD	INPUT
140		JEQ	RLOOP
145		RD	INPUT
150		COMPR	A,S
155		JEQ	EXIT
160		STCH	BUFFER,X
165		TIXR	T
170		JLT	RLOOP
175	EXIT	STX	LENGTH
180		RSUB	
185	INPUT	BYTE	X'F1'

SUBROUTINE TO WRITE RECORD FROM BUFFER

210	WRREC	CLEAR	X
212		LDT	LENGTH
215	WLOOP	TD	OUTPUT
220		JEQ	WLOOP
225		LDCH	BUFFER,X
230		WD	OUTPUT
235		TIXR	T
240		JLT	WLOOP
245		RSUB	
250	OUTPUT	BYTE	X'05'
255		END	FIRST

210	WRREC	CLEAR	X
212		LDT	LENGTH
215	WLOOP	TD	=X'05'
220		JEQ	WLOOP
225		LDCH	BUFFER,X
230		WD	=X'05'
235		TIXR	T
240		JLT	WLOOP
245		RSUB	
255		END	FIRST

Outline

Literals (1/3)

- It is often convenient for the programmer to be able to write the value of a constant operand as a part of the instruction that uses it.
- This avoids having to define the constant elsewhere in the program and make up a label for it.
- Such an operand is called a literal because the value is stated “literally” in the instruction.
- In SIC/XE assembler language notation, a literal is identified with the prefix =, which is followed by a specification of the literal value, using the same notation as in the BYTE statement.

Literals (2/3)

- The difference between a literal and an immediate operand.
  - (#) Immediate addressing: the operand value is assembled as part of the machine instruction.
  - (=) Literal addressing: the assembler generates the specified value as a constant at some other memory location.

40	0017		J	CLOOP	3F2FEC
45	001A	ENDFIL	LDA	=C'EOF'	032010
50	001D		STA	BUFFER	0F2016
55	0020		LDA	#3	010003
60	0023		STA	LENGTH	0F200D
65	0026		+JSUB	WRREC	4B10105D
70	002A		J	@RETADR	3E2003
93			LTORG		
	002D	*	=C'EOF'		454F46
95	0030	RETADR	RESW	1	

Literals (3/3)

- All of the literal operands used in a program are gathered together into one or more literal pools. Normally literals are placed into a pool at the end of the program.
- When the assembler encounters a LTORG statement, it creates a literal operands used since the previous LTORG (or the beginning of the program).
- Most assemblers recognize duplicate literals and store only one copy of the specified data value.
  - By comparison of the character strings defining them.
  - EX: the literal =X'05' (Figure 2.9, Line 215 and 230)
  - EX: =C'EOF' and =X'454F45' ?

The implementation of literals

- The basic data structure needed is a literal table (LITTAB).
  - Literal name, value, length, and address
- Pass1:
  - Search and update LITTAB for the specified literal name
  - When encounters a LTORG statement or the end of the program, the assembler makes a scan of the LITTAB and assigns an address for all unallocated literals
  - Update the location counter to reflect the number of bytes occupied by each literal
- Pass2:
  - Search LITTAB for the address of each literal encountered
  - Literal values placed at correct locations in the object program
  - If a literal value represents an address in the program, the assembler must also generate the appropriate Modification record.

Outline

Department of Electrical Engineering, Feng-Chia University

EQU assembler directive

- EQU (for "equate") assembler directive allows the programmer to define symbols and specify their values.
  - Improve readability in place of numeric values
  - EX: "MAXLEN" and " * " (Figure 2.9, Line 106 and 107)

```
 93                              LTORG
        002D        *           =C'EOF'                     454F46
 95     0030        RETADR       RESW     1
100     0033        LENGTH       RESW     1
105     0036        BUFFER       RESB     4096
106     1036        BUFEND       EQU      *
107     1000        MAXLEN       EQU      BUFEND-BUFFER
110                 .
115                 .            SUBROUTINE TO READ RECORD INTO BUFFER
120                 .
125     1036        RDREC        CLEAR    X                 B410
130     1038                     CLEAR    A                 B400
132     103A                     CLEAR    S                 B440
133     103C                    +LDT      #MAXLEN           75101000
135     1040        RLOOP        TD       INPUT             E32019
```


Department of Electrical Engineering, Feng-Chia University

EQU assembler directive

- The resulting object code is exactly the same as in the original version of the instruction; however, the source statement is easier to understand.

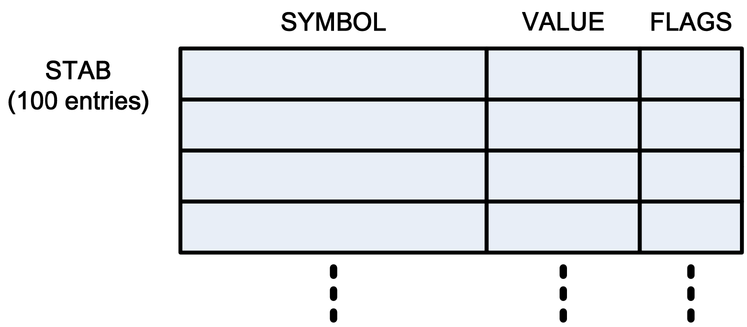
- Another common use of EQU is in defining mnemonic names for registers.

```
A        EQU    0
X        EQU    1
L        EQU    2
```

```
INDEX    EQU    X
BASE     EQU    B
FLOAT    EQU    F
```

ORG assembler directive

- ORG (for "origin") assembler directive
  - When ORG is encountered during assembly of a program, the assembler resets its location counter (LOCCTR) to the specified value.
- Example:
  - SYMBOL is 6-byte, VALUE is 1-word, and FLAGS is 2-byte



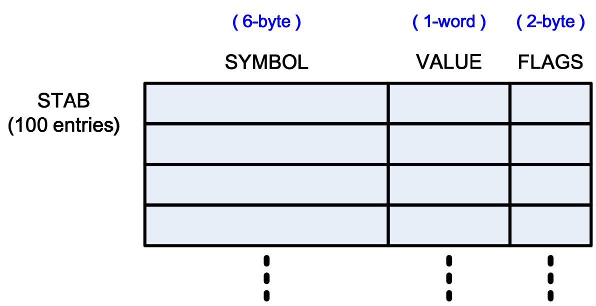
Department of Electrical Engineering, Feng-Chia University

- Use **EQU** assembler directives

```
STAB      RESB   1100

SYMBOL    EQU    STAB
VALUE     EQU    STAB+6
FLAGS     EQU    STAB+9
```

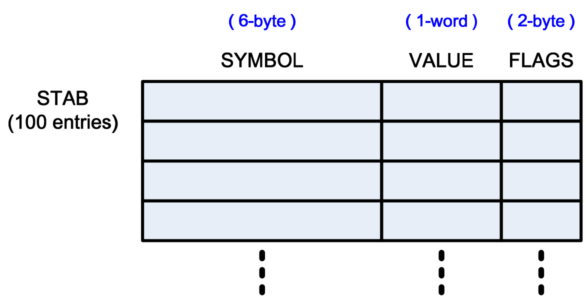
```
LDA       VALUE, X
```



- Use **ORG** assembler directives

```
STAB      RESB   1100
          ORG    STAB
SYMBOL    RESB   6
VALUE     RESW   1
FLAGS     RESB   2
          ORG    STAB+1100
```

```
LDA       VALUE, X
```



Restrictions

- EQU:
  all symbols used on the right-hand side of the statement must have been defined previously in the program.

✓
```
ALPHA     RESW     1
BETA      EQU      ALPHA
```

✗
```
BETA      EQU      ALPHA
ALPHA     RESW     1
```

- ORG:
  all symbols used to specify the new location counter value must have been previously defined.

✗
```
          ORG      ALPHA
BYTE1     RESB     1
BYTE2     RESB     1
BYTE3     RESB     1
          ORG
ALPHA     RESW     1
```


Outline

Department of Electrical Engineering, Feng-Chia University

Expressions

- Assemblers generally allow arithmetic expressions formed according to the normal rules using the operators +, -, *, / .
- Division is usually defined to produce an integer result.
- Individual terms in the expression may be constants, user-defined symbols, or special terms.
- " * " : This special term represents the value of the next unassigned memory location.

106	BUFEND	EQU	*

- Expression Terms
  - Relative terms: defined relative to the beginning of the program
  - Absolute terms: independent of program location

Department of Electrical Engineering, Feng-Chia University

Absolute and Relative Expressions

- Absolute Expressions
  - Contains only absolute terms
  - Contains relative terms provided the relative terms occur in pairs with opposite signs; the dependency on the program starting address is canceled out; the result is an absolute value

107	MAXLEN	EQU	BUFEND-BUFFER

- Relative Expressions
  - Contains an odd number of relative terms, with one more positive terms than negative terms
  - No relative term may enter into a multiplication or division operation

Department of Electrical Engineering, Feng-Chia University

Defining Symbol Types in the Symbol Table

- To determine the type of an expression, we must keep track of the types of all symbols defined in the program.
- For this purpose we need a flag in the symbol table to indicate type of value (absolute or relative) in addition to the value itself.

Symbol	Type	Value
RETADR	R	0030
BUFFER	R	0036
BUFEND	R	1036
MAXLEN	A	1000

- With this information the assembler can easily determine the type of each expression used as an operand and generate Modification records in the object program for relative values.

Department of Electrical Engineering, Feng-Chia University

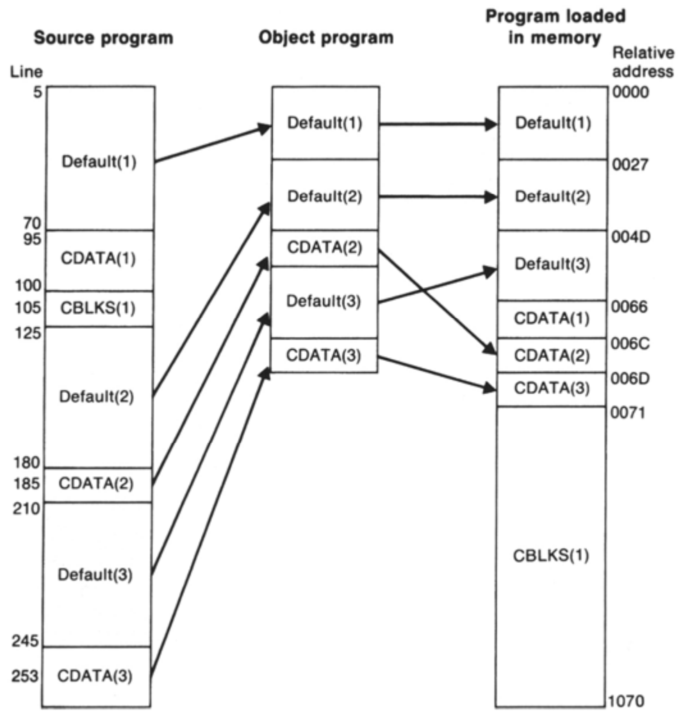
Outline

Program Blocks vs. Control Sections

- Program Blocks
  - Refer to segments of code that are rearranged within a single object program unit
- Control Sections
  - Refer to segments that are translated into independent object program units

Department of Electrical Engineering, Feng-Chia University

Program Blocks



Department of Electrical Engineering, Feng-Chia University

Program Blocks (1/2)

- The source programs logically contained subroutines, data areas, etc. However, they were handled by the assembler as one entity, resulting in a single block of object code.
- Many assemblers provide features that allow more flexible handling of the source and object programs.
  - Some features allow the generated machine instructions and data to appear in the object program in a different order from the corresponding source statements.
  - Other features result in the creation of several independent parts of the object program.
  - These parts maintain their identity and are handled separately by the loader.

Department of Electrical Engineering, Feng-Chia University

Program Blocks (2/2)

- The assembler directive USE indicates which portions of the source program belong to the various blocks.
- Each program block may actually contain several separate segments of the source program. The assembler will (logically) rearrange these segments to gather together the pieces of each block.

Program from Fig. 2.11 with object code

[Figure 2.12]

Line	Loc/Block		Source statement			Object code
5	0000	0	COPY	START	0	
10	0000	0	FIRST	STL	RETADR	172063
15	0003	0	CLOOP	JSUB	RDREC	4B2021
20	0006	0		LDA	LENGTH	032060
25	0009	0		COMP	#0	290000
30	000C	0		JEQ	ENDFIL	332006
35	000F	0		JSUB	WRREC	4B203B
40	0012	0		J	CLOOP	3F2FEE
45	0015	0	ENDFIL	LDA	=C'EOF'	032055
50	0018	0		STA	BUFFER	0F2056
55	001B	0		LDA	#3	010003
60	001E	0		STA	LENGTH	0F2048
65	0021	0		JSUB	WRREC	4B2029
70	0024	0		J	@RETADR	3E203F
92	0000	1		USE	CDATA	
95	0000	1	RETADR	RESW	1	
100	0003	1	LENGTH	RESW	1	
103	0000	2		USE	CBLKS	
105	0000	2	BUFFER	RESB	4096	
106	1000	2	BUFEND	EQU	*	
107	1000		MAXLEN	EQU	BUFEND-BUFFER	

Department of Electrical Engineering, Feng-Chia University

SUBROUTINE TO READ RECORD INTO BUFFER

123	0027	0		USE		
125	0027	0	RDREC	CLEAR	X	B410
130	0029	0		CLEAR	A	B400
132	002B	0		CLEAR	S	B440
133	002D	0		+LDT	#MAXLEN	75101000
135	0031	0	RLOOP	TD	INPUT	E32038
140	0034	0		JEQ	RLOOP	332FFA
145	0037	0		RD	INPUT	DB2032
150	003A	0		COMPR	A,S	A004
155	003C	0		JEQ	EXIT	332008
160	003F	0		STCH	BUFFER,X	57A02F
165	0042	0		TIXR	T	B850
170	0044	0		JLT	RLOOP	3B2FEA
175	0047	0	EXIT	STX	LENGTH	13201F
180	004A	0		RSUB		4F0000
183	0006	1		USE	CDATA	
185	0006	1	INPUT	BYTE	X'F1'	F1

SUBROUTINE TO WRITE RECORD FROM BUFFER

208	004D	0		USE		
210	004D	0	WRREC	CLEAR	X	B410
212	004F	0		LDT	LENGTH	772017
215	0052	0	WLOOP	TD	=X'05'	E3201B
220	0055	0		JEQ	WLOOP	332FFA
225	0058	0		LDCH	BUFFER,X	53A016
230	005B	0		WD	=X'05'	DF2012
235	005E	0		TIXR	T	B850
240	0060	0		JLT	WLOOP	3B2FEF
245	0063	0		RSUB		4F0000
252	0007	1		USE	CDATA	
253				LTORG		
	0007	1	*	=C'EOF'		454F46
	000A	1	*	=X'05'		05
255				END	FIRST	

The implementation of Program Blocks (1/2)

Pass 1

- A separate location counter for each program block.
- The current value of this location counter is saved when switching to another block, and the saved value is restored when resuming a previous block.
- Each label in the program is assigned an address that is relative to the start of the block that contains it.
- The latest value of the location counter for each block indicates the length of that block.
- At the end of Pass 1 the assembler constructs a table that contains the starting addresses and lengths for all blocks.

(see next page)

The implementation of Program Blocks (2/2)

Block name	Block number	Address	Length
(default)	0	0000	0066
CDATA	1	0066	000B
CBLKS	2	0071	1000

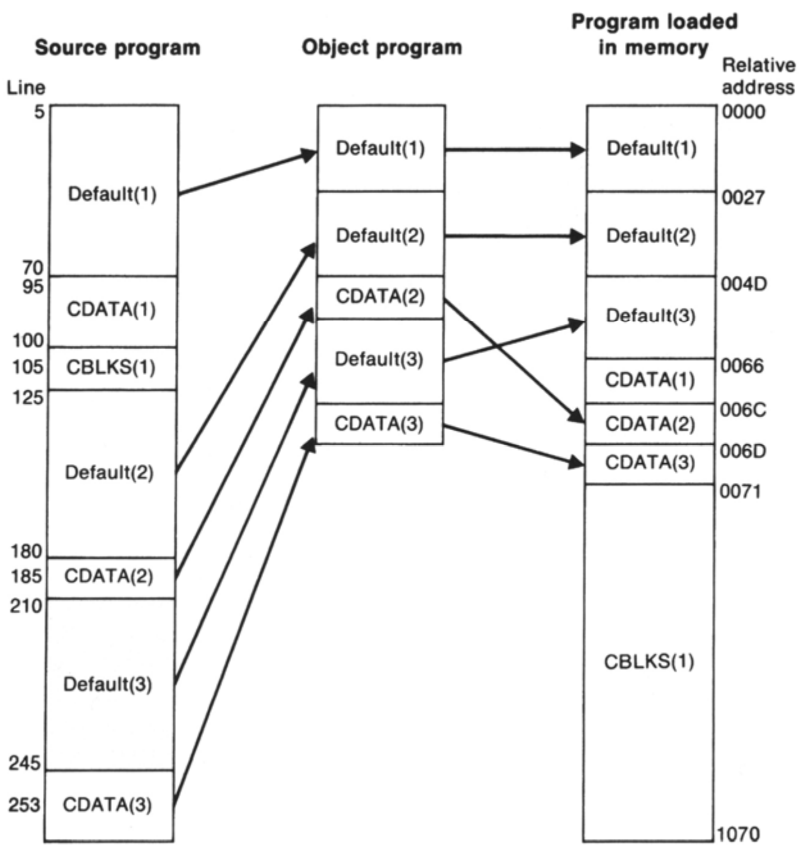
Pass 2

- The address for each symbol relative to the start of the object program is easily found from the information in SYMTAB.
- The assembler simply adds the location of the symbol, relative to the start of its block, to the assigned block starting address.

Object Program
(Corresponding to Figure 2.11)
[Figure 2.13]

```
H^COPY  ^000000^001071
T^000000^1E^172063^4B2021^032060^290000^332006^4B203B^3F2FEE^032055^0F2056^010003
T^00001E^09^0F2048^4B2029^3E203F                                        Default(1)
T^000027^1D^B410^B400^B440^75101000^E32038^332FFA^DB2032^A004^332008^57A02F^B850
T^000044^09^3B2FEA^13201F^4F0000                                        Default(2)
T^00006C^01^F1                                                          CDATA(2)
T^00004D^19^B410^772017^E3201B^332FFA^53A016^DF2012^B850^3B2FEF^4F0000  Default(3)
T^00006D^04^454F46^05                                                   CDATA(3)
E^000000
```

Department of Electrical Engineering, Feng-Chia University



Department of Electrical Engineering, Feng-Chia University

Outline

Assembler Directives

- <symbol> CSECT
  - The CSECT assembler directive signals the start of a new control section named <symbol>
- EXTDEF < symbol_1, symbol_2, … symbol_n >
  - The EXTDEF (external definition) statement in a control section names symbols, called external symbols, that are defined in this control section and may be used by other sections.
  - Control section names do not need to be named in an EXTDEF statement because they are automatically considered to be external symbols.
- EXTREF < symbol_1, symbol_2, … symbol_n >
  - The EXTREF (external reference) statement names symbols that are used in this control section and are defined elsewhere.

Department of Electrical Engineering, Feng-Chia University

Control Section : COPY

[Figure 2.16]

Line	Loc	Source statement			Object code
5	0000	COPY	START	0	
6			EXTDEF	BUFFER,BUFEND,LENGTH	
7			EXTREF	RDREC,WRREC	
10	0000	FIRST	STL	RETADR	172027
15	0003	CLOOP	+JSUB	RDREC	4B100000
20	0007		LDA	LENGTH	032023
25	000A		COMP	#0	290000
30	000D		JEQ	ENDFIL	332007
35	0010		+JSUB	WRREC	4B100000
40	0014		J	CLOOP	3F2FEC
45	0017	ENDFIL	LDA	=C'EOF'	032016
50	001A		STA	BUFFER	0F2016
55	001D		LDA	#3	010003
60	0020		STA	LENGTH	0F200A
65	0023		+JSUB	WRREC	4B100000
70	0027		J	@RETADR	3E2000
95	002A	RETADR	RESW	1	
100	002D	LENGTH	RESW	1	
103			LTORG		
	0030	*	=C'EOF'		454F46
105	0033	BUFFER	RESB	4096	
106	1033	BUFEND	EQU	*	
107	1000	MAXLEN	EQU	BUFEND-BUFFER	

Department of Electrical Engineering, Feng-Chia University

Control Section : RDREC

[Figure 2.16]

109	0000	RDREC	CSECT		
110		.			
115		.	SUBROUTINE TO READ RECORD INTO BUFFER		
120		.			
122			EXTREF	BUFFER,LENGTH,BUFEND	
125	0000		CLEAR	X	B410
130	0002		CLEAR	A	B400
132	0004		CLEAR	S	B440
133	0006		LDT	MAXLEN	77201F
135	0009	RLOOP	TD	INPUT	E3201B
140	000C		JEQ	RLOOP	332FFA
145	000F		RD	INPUT	DB2015
150	0012		COMPR	A,S	A004
155	0014		JEQ	EXIT	332009
160	0017		+STCH	BUFFER,X	57900000
165	001B		TIXR	T	B850
170	001D		JLT	RLOOP	3B2FE9
175	0020	EXIT	+STX	LENGTH	13100000
180	0024		RSUB		4F0000
185	0027	INPUT	BYTE	X'F1'	F1
190	0028	MAXLEN	WORD	BUFEND-BUFFER	000000

Control Section : WRREC

[Figure 2.16]

```
0000    WRREC    CSECT
                 .
                 .        SUBROUTINE TO WRITE RECORD FROM BUFFER
                 .
                          EXTREF    LENGTH,BUFFER
0000                      CLEAR     X                  B410
0002                     +LDT       LENGTH             77100000
0006    WLOOP             TD       =X'05'              E32012
0009                      JEQ       WLOOP              332FFA
000C                     +LDCH      BUFFER,X           53900000
0010                      WD       =X'05'              DF2008
0013                      TIXR      T                  B850
0015                      JLT       WLOOP              3B2FEE
0018                      RSUB                         4F0000
                          END       FIRST
001B    *                =X'05'                        05
```


Object Program Format

	Column	Contents
Define Record	1	D
	2-7	Name of external symbol defined in this control section
	8-13	Relative address of symbol within this control section (HEX)
	14-73	Repeat information in Col. 2-13 for other external symbols
Refer Record	1	R
	2-7	Name of external symbol referred to in this control section
	8-73	Names of other external reference symbols
Mod. Record	1	M
	2-7	Starting address of the field to be modified, relative to the beginning of the program (HEX)
	8-9	Length of the field to be modified, in half-bytes (HEX)
	10	Modification flag (+ or -)
	11-16	External symbol whose value is to be added to or subtracted from the indicated field

```
H^COPY  ^000000^001033
D^BUFFER^000033^BUFEND^001033^LENGTH^00002D
R^RDREC ^WRREC
T^000000^1D^172027^4B100000^032023^290000^332007^4B100000^3F2FEC^032016^0F2016
T^00001D^0D^010003^0F200A^4B100000^3E2000
T^000030^03^454F46
M^000004^05^+RDREC
M^000011^05^+WRREC
M^000024^05^+WRREC
E^000000
```

```
H^RDREC ^000000^00002B
R^BUFFER^LENGTH^BUFEND
T^000000^1D^B410^B400^B440^77201F^E3201B^332FFA^DB2015^A004^332009^57900000^B850
T^00001D^0E^3B2FE9^13100000^4F0000^F1^000000
M^000018^05^+BUFFER
M^000021^05^+LENGTH
M^000028^06^+BUFEND
M^000028^06^-BUFFER
E
```

```
H^WRREC ^000000^00001C
R^LENGTH^BUFFER
T^000000^1C^B410^77100000^E32012^332FFA^53900000^DF2008^B850^3B2FEE^4F0000^05
M^000003^05^+LENGTH
M^00000D^05^+BUFFER
E
```


Outline

Department of Electrical Engineering, Feng-Chia University

Outline

Department of Electrical Engineering, Feng-Chia University

One-Pass Assemblers

- The main problem in trying to assemble a program in one pass involves forward references.
- Eliminate forward references
  - Data items are defined before they are referenced.
  - But, forward references to labels on instructions cannot be eliminated as easily.
  - Prohibit forward references to data items.
- There are two main types of one-pass assembler.
  - Load-and-Go : Produces object code directly in memory for immediate execution
  - Object Program Output : Produces the usual kind of object program for late execution.

Sample program for a one-pass assembler

[Figure 2.18]

Line	Loc	Source statement			Object code
0	1000	COPY	START	1000	
1	1000	EOF	BYTE	C'EOF'	454F46
2	1003	THREE	WORD	3	000003
3	1006	ZERO	WORD	0	000000
4	1009	RETADR	RESW	1	
5	100C	LENGTH	RESW	1	
6	100F	BUFFER	RESB	4096	
9		.			
10	200F	FIRST	STL	RETADR	141009
15	2012	CLOOP	JSUB	RDREC	48203D
20	2015		LDA	LENGTH	00100C
25	2018		COMP	ZERO	281006
30	201B		JEQ	ENDFIL	302024
35	201E		JSUB	WRREC	482062
40	2021		J	CLOOP	302012
45	2024	ENDFIL	LDA	EOF	001000
50	2027		STA	BUFFER	0C100F
55	202A		LDA	THREE	001003
60	202D		STA	LENGTH	0C100C
65	2030		JSUB	WRREC	482062
70	2033		LDL	RETADR	081009
75	2036		RSUB		4C0000

SUBROUTINE TO READ RECORD INTO BUFFER

121	2039	INPUT	BYTE	X'F1'	F1
122	203A	MAXLEN	WORD	4096	001000
124		.			
125	203D	RDREC	LDX	ZERO	041006
130	2040		LDA	ZERO	001006
135	2043	RLOOP	TD	INPUT	E02039
140	2046		JEQ	RLOOP	302043
145	2049		RD	INPUT	D82039
150	204C		COMP	ZERO	281006
155	204F		JEQ	EXIT	30205B
160	2052		STCH	BUFFER,X	54900F
165	2055		TIX	MAXLEN	2C203A
170	2058		JLT	RLOOP	382043
175	205B	EXIT	STX	LENGTH	10100C
180	205E		RSUB		4C0000

SUBROUTINE TO WRITE RECORD FROM BUFFER

206	2061	OUTPUT	BYTE	X'05'	05
207		.			
210	2062	WRREC	LDX	ZERO	041006
215	2065	WLOOP	TD	OUTPUT	E02061
220	2068		JEQ	WLOOP	302065
225	206B		LDCH	BUFFER,X	50900F
230	206E		WD	OUTPUT	DC2061
235	2071		TIX	LENGTH	2C100C
240	2074		JLT	WLOOP	382065
245	2077		RSUB		4C0000
255			END	FIRST	

Department of Electrical Engineering, Feng-Chia University

Load-and-Go Assemblers (1/2)

- This kind of load-and-go assembler is useful in a system that is oriented toward program development and testing.
- If an instruction operand is a symbol that has not yet been defined, the operand address is omitted when the instruction is assembled.
  - The symbol used as an operand is entered into the symbol table.
  - This entry is flagged to indicate that the symbol is undefined.
  - The address of the operand field of the instruction that refers to the undefined symbol is added to a list of forward references associated with the symbol table entry.

(Cont.)

Department of Electrical Engineering, Feng-Chia University

Load-and-Go Assemblers (2/2)

- When the definition for a symbol is encountered, the forward reference list for that symbol is scanned, and the proper address is inserted into any instructions previously generated.
- At the end of the program, all symbols must be defined without any * in SYMTAB.
- For a load-and-go assembler, the actual address must be known at assembly time.

Object code in memory and symbol table entries for program in Fig. 2.18 after scanning line 40

Memory address	Contents			
1000	454F4600	00030000	00xxxxxx	xxxxxxxx
1010	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx
•				
•				
•				
2000	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxx14
2010	100948--	--00100C	28100630	-----48--
2020	--3C2012			
•				
•				
•				

Symbol	Value
LENGTH	100C
RDREC	* → 2013 | 0
THREE	1003
ZERO	1006
WRREC	* → 201F | 0
EOF	1000
ENDFIL	* → 201C | 0
RETADR	1009
BUFFER	100F
CLOOP	2012
FIRST	200F

Object code in memory and symbol table entries for program in Fig. 2.18 after scanning line 160

Memory address	Contents			
1000	454F4600	00030000	00xxxxxx	xxxxxxxx
1010	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx
•				
•				
•				
2000	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxx14
2010	10094820	3D00100C	28100630	202448--
2020	--3C2012	0010000C	100F0010	030C100C
2030	48-----08	10094C00	00F10010	00041006
2040	001006E0	20393020	43D82039	28100630
2050	-----5490	0F		
•				
•				
•				

Symbol	Value
LENGTH	100C
RDREC	203D
THREE	1003
ZERO	1006
WRREC	* → 201F → 2031 | 0
EOF	1000
ENDFIL	2024
RETADR	1009
BUFFER	100F
CLOOP	2012
FIRST	200F
MAXLEN	203A
INPUT	2039
EXIT	* → 2050 | 0
RLOOP	2043

Department of Electrical Engineering, Feng-Chia University

Object Program Output Assemblers

- One-pass assemblers that produce object programs as output are often used on systems where external working-storage devices are not available.
- The assembler generate another **Text record** with the correct operand address.
- When the program is loaded, this address will be inserted into the instruction by the action of the loader.
- The object program records must be kept in their original order when they are presented to the loader.

Department of Electrical Engineering, Feng-Chia University

Object program from one-pass assembler for program in Fig. 2.18

```
H^COPY  ^001000^00107A
T^001000^09^454F46^000003^000000
T^00200F^15^141009^480000^00100C^281006^300000^480000^3C2012
T^00201C^02^2024
T^002024^19^001000^0C100F^001003^0C100C^480000^081009^4C0000^F1^001000
T^002013^02^203D
T^00203D^1E^041006^001006^E02039^302043^D82039^281006^300000^54900F^2C203A^382043
T^002050^02^205B
T^00205B^07^10100C^4C0000^05
T^00201F^02^2062
T^002031^02^2062
T^002062^18^041006^E02061^302065^50900F^DC2061^2C100C^382065^4C0000
E^00200F
```

Outline

Multi-Pass Assemblers

- In our discussion of the EQU assembler directive, we required that any symbol used on the RHS be defined previously in the source program.
- Consider, for example, the sequence

ALPHA	EQU	BETA
BETA	EQU	DELTA
DELTA	RESW	1

  - Two-pass assemblers ✘

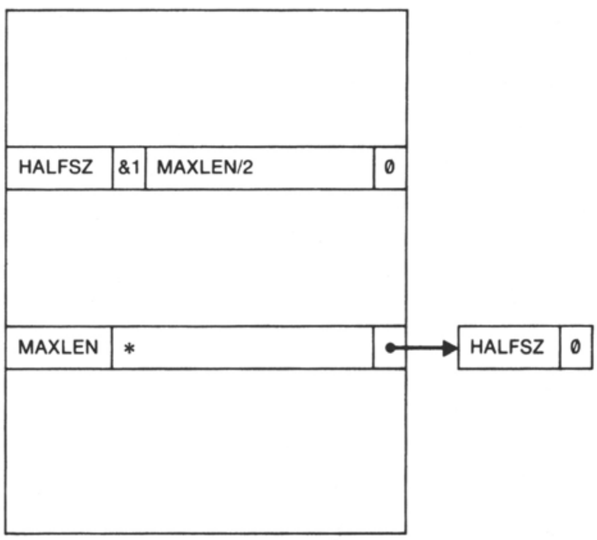
Department of Electrical Engineering, Feng-Chia University

Multi-Pass Assemblers

- The general solution is a multi-pass assembler that can make as many passes as are needed to process the definitions of symbols.
- It is not necessary for such an assembler to make more than two passes over the entire program.
- The method we describe involves storing those symbol definitions that involve forward references in the symbol table.
  - This table also indicates which symbols are dependent on the values of others, to facilitate symbol evaluation.

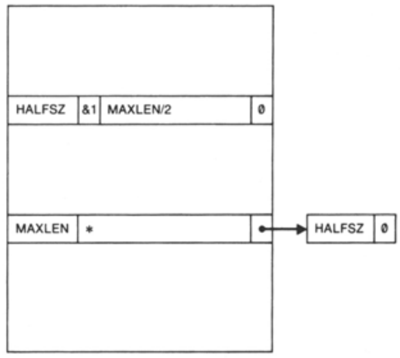
Department of Electrical Engineering, Feng-Chia University

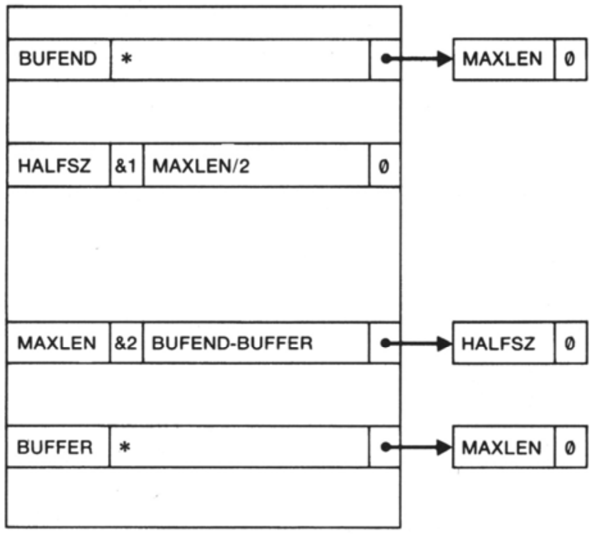
```
1   HALFSZ   EQU    MAXLEN/2
2   MAXLEN   EQU    BUFEND-BUFFER
3   PREVBT   EQU    BUFFER-1
                    .
                    .
                    .
4   BUFFER   RESB   4096
5   BUFEND   EQU    *
```



Department of Electrical Engineering, Feng-Chia University

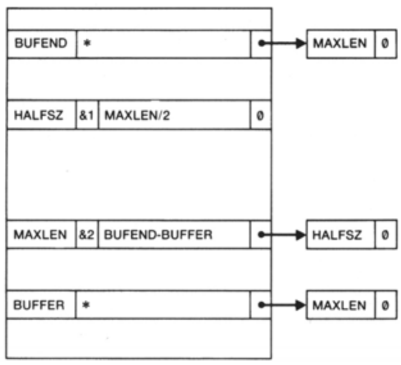
```
1    HALFSZ    EQU     MAXLEN/2
2    MAXLEN    EQU     BUFEND-BUFFER
3    PREVBT    EQU     BUFFER-1
                       .
                       .
                       .
4    BUFFER    RESB    4096
5    BUFEND    EQU     *
```

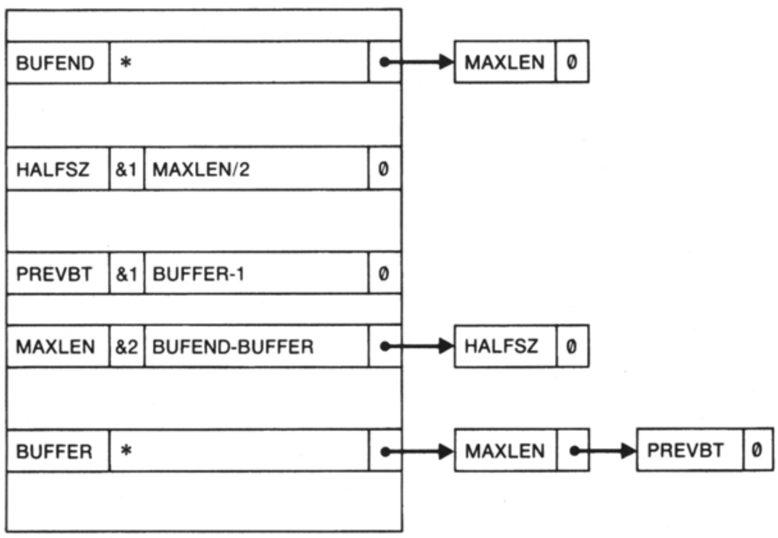




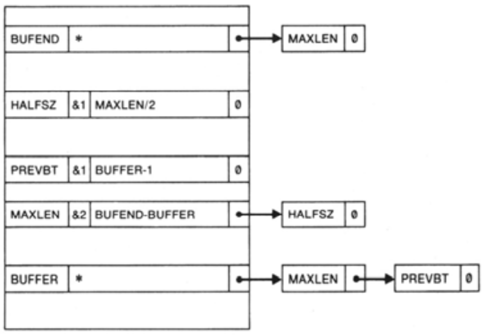
Department of Electrical Engineering, Feng-Chia University

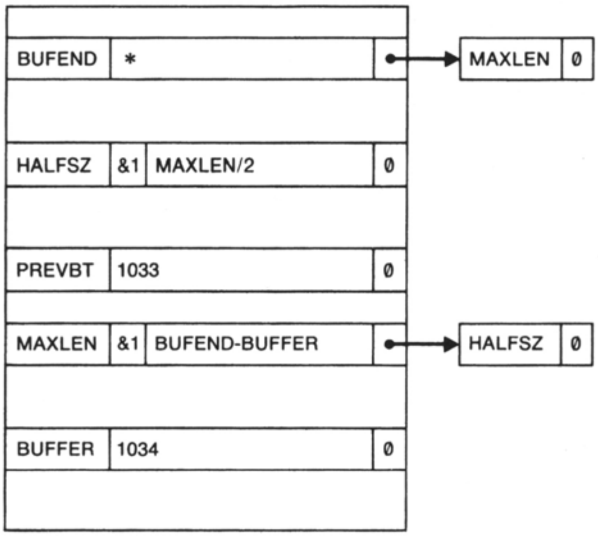
```
1    HALFSZ    EQU     MAXLEN/2
2    MAXLEN    EQU     BUFEND-BUFFER
3    PREVBT    EQU     BUFFER-1
                       .
                       .
                       .
4    BUFFER    RESB    4096
5    BUFEND    EQU     *
```





```
1    HALFSZ    EQU     MAXLEN/2
2    MAXLEN    EQU     BUFEND-BUFFER
3    PREVBT    EQU     BUFFER-1
                       .
                       .
                       .
4    BUFFER    RESB    4096
5    BUFEND    EQU     *
```






```
1    HALFSZ    EQU     MAXLEN/2
2    MAXLEN    EQU     BUFEND-BUFFER
3    PREVBT    EQU     BUFFER-1
                       .
                       .
                       .
4    BUFFER    RESB    4096
5    BUFEND    EQU     *
```