S J P N Trust's

HIRASUGAR INSTITUTE OF TECHNOLOGY, NIDASOSHI.

Inculcating Values, Promoting Prosperity

Approved by AICTE, Recognized by Govt. of Karnataka and Permanently Affiliated to VTU Belagavi.

Accredited at 'A' Grade by NAAC

Programmes Accredited by NBA: CSE, ECE, EEE & ME

Subject: System Software and Compilers (18CS61)

Module 1: Introduction to System Software

Dr. Mahesh G. Huddar

Dept. of Computer Science and Engineering

System Software Definition

- System software consists of a variety of programs that support the operation of a computer.
- Examples for system software are
  - Operating System,
  - Compiler,
  - Assembler,
  - Macro Processor,
  - Loader or Linker,
  - Debugger,
  - Text Editor

System Software vs Application Software

System software	Application software
Collection of programs that help the user to interact with hardware components efficiently.	Collection of programs written for a specific application such as banking, browsing, MS-OFFICE etc.
System software control and manage the hardware and hence system software directly interact with hardware	Application software uses the service of the system software to interact with hardware components. So, application software will not interact with hardware directly.
To write system software the programmer needs to understand the architecture and hardware details and hence system software are machine dependent	To write the application software the programmer need not worry about the architecture and hardware details and hence application software are machine independent.
Programmer should be more familiar with architecture, instruction formats, addressing modes and so on.	Programmer should be more familiar with programming languages, data structures and clear knowledge of the problem domain.
Development of system software is complex task	Development of application software is relatively easier.
Examples: compiler, assembler, operating system etc	Examples: ticket reservation, banking software, MS-WORD etc.

SIC Machine Architecture

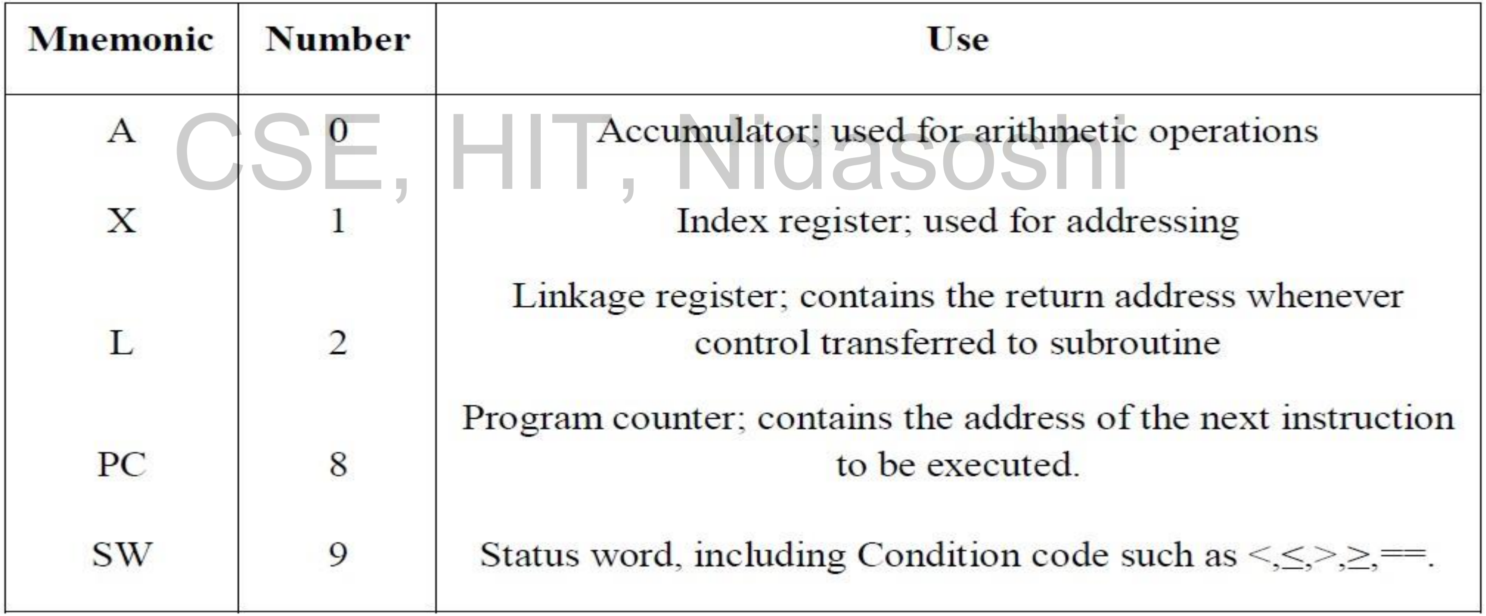
- The SIC machine architecture depends on the following features:
    - Memory
    - Registers
    - Data Formats
    - Instruction Formats
    - Addressing Modes
    - Instruction Set
    - Input and Output

SIC Machine Architecture

- Memory
  - Memory consists of 8-bit bytes
  - Any 3 consecutive bytes form a word (24 bits)
  - Total of 32768 (2^{15}) bytes in the computer memory

SIC Machine Architecture

- Registers
  - Five 24-bits registers. Their mnemonic, number and use are given in the following table.



Mnemonic	Number	Use
A	0	Accumulator; used for arithmetic operations
X	1	Index register; used for addressing
L	2	Linkage register; contains the return address whenever control transferred to subroutine
PC	8	Program counter; contains the address of the next instruction to be executed.
SW	9	Status word, including Condition code such as <,≤,>,≥,==.

SIC Machine Architecture

- Data Formats
  - Integers are stored as 24-bit binary number
  - 2’s complement representation for negative values
  - Characters are stored using 8-bit ASCII codes
  - No floating-point hardware on the standard version of SIC

SIC Machine Architecture

- Instruction Formats
  - All machine instructions on the standard version of SIC have the 24-bit format as shown below

Opcode(8)	x	Address (15)

- Addressing Modes

  - There are two addressing modes available, which are as shown in the below table. Parentheses are used to indicate the contents of a register or a memory location.

Mode	Indication	Target address calculation
Direct	x = 0	TA = address
Indexed	x = 1	TA = address + (x)

SIC Machine Architecture

- Instruction Set
  - Load and store registers - LDA, LDX, STA, STX, etc.
  - Integer arithmetic operations - ADD, SUB, MUL, DIV
    - All arithmetic operations involve register A and a word in memory, with the result being left in A
  - COMP – Comparison instruction
  - Conditional jump instructions - JLT, JEQ, JGT
  - Subroutine linkage - JSUB, RSUB
  - I/O (transferring 1 byte at a time to/from the rightmost 8 bits of register A)
    - Test Device instruction (TD)
    - Read Data (RD)
    - Write Data (WD)

SIC/XE Machine Architecture

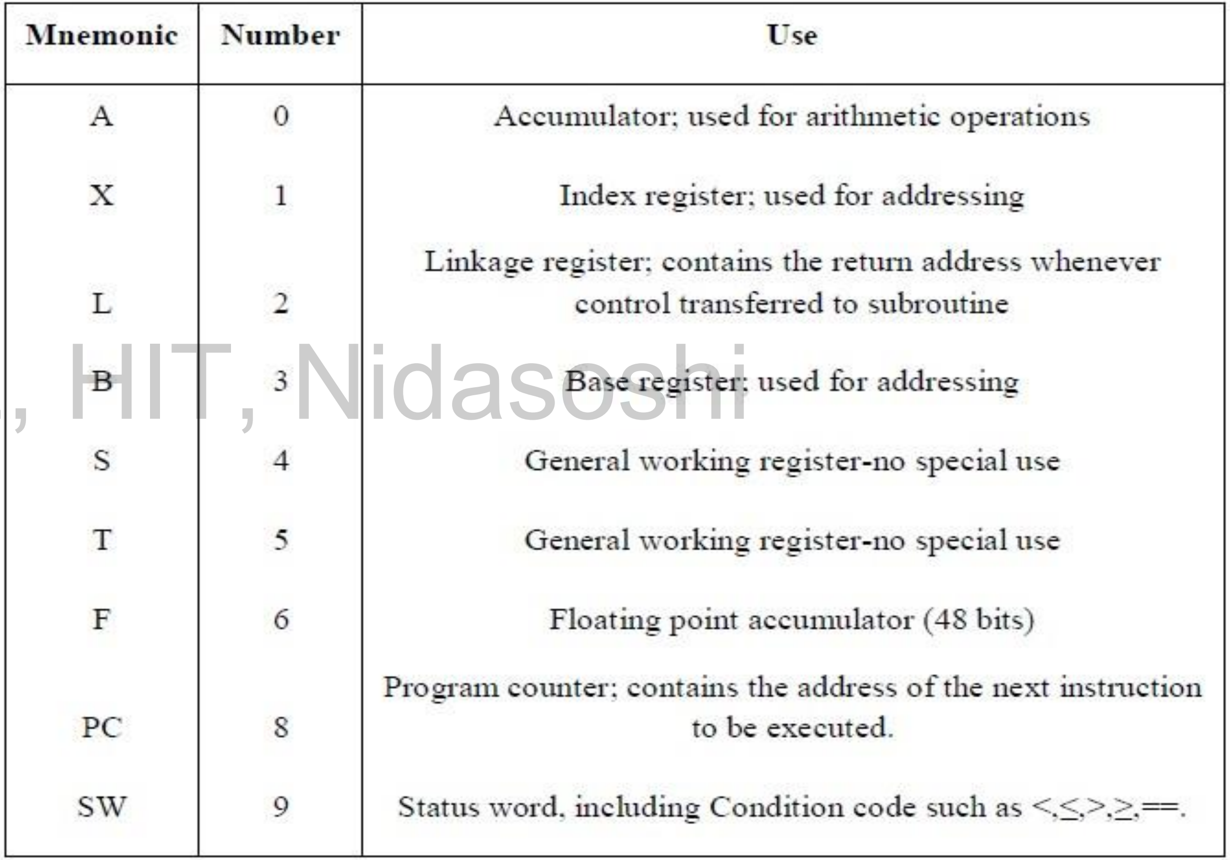
- The SIC machine architecture depends on the following features:
  - Memory
  - Registers
  - Data Formats
  - Instruction Formats
  - Addressing Modes
  - Instruction Set
  - Input and Output

SIC/XE Machine Architecture

- Memory
  - Memory consists of 8-bit bytes
  - Any 3 consecutive bytes form a word (24 bits)
  - Total of 1 Mb (2^{20}) bytes in the computer memory

SIC/XE Machine Architecture

- Registers
  - There are nine registers; each register is 24 bits in length except floating point register.
  - Their mnemonic, number and uses are shown in the following table.



Mnemonic	Number	Use
A	0	Accumulator; used for arithmetic operations
X	1	Index register; used for addressing
L	2	Linkage register; contains the return address whenever control transferred to subroutine
B	3	Base register; used for addressing
S	4	General working register-no special use
T	5	General working register-no special use
F	6	Floating point accumulator (48 bits)
PC	8	Program counter; contains the address of the next instruction to be executed.
SW	9	Status word, including Condition code such as <,≤,>,≥,==.

SIC/XE Machine Architecture

- Data Formats
  - Integers are stored as 24-bit binary number
  - 2's complement representation for negative values
  - Characters are stored using 8-bit ASCII codes
  - Support 48 bit floating-point numbers

1	11	36
s	exponent	fraction

  - There is a 48-bit floating-point data type, $F*2^{(e-1024)}$

SIC/XE Machine Architecture

- Instruction Formats
  - Format 1 (1 byte –

8
opcode

Example: RSUB

  - Format 1 (2 byte) -

8	4	4
op	r1	r2

Example: ADDR S, T

  - Format 3 (4 byte) -

6	1	1	1	1	1	1	12
op	n	i	x	b	p	e	Displacement

    - Example: LDA #3
  - Format 4 (6 byte) -

6	1	1	1	1	1	1	20
op	n	i	x	b	p	e	Address

    - Example: +JSUB RDREC

SIC/XE Machine Architecture

- Addressing Modes and Flag Bits
  - 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)
  - Immediate (n=0, i=1, x=0)
  - Indirect (n=1, i=0, x=0)
  - Indexing (both n & i = 0 or 1, x=1)
  - Extended (e=1 for format 4, e=0 for format 3)

SIC/XE Machine Architecture

- Base Relative Addressing Mode

	n	i	x	b	p	e	
opcode	1	1		1	0		disp

n=1, i=1, b=1, p=0, TA = (B) + disp (0≤disp ≤4095)

- Program-Counter Relative Addressing Mode

	n	i	x	b	p	e	
opcode	1	1		0	1		disp

n=1, i=1, b=0, p=1, TA= (PC) + disp (-2048≤disp ≤2047)

SIC/XE Machine Architecture

- Direct Addressing Mode

	n	i	x	b	p	e	
opcode	1	1		0	0		disp

n=1, i=1, b=0, p=0, TA = disp (0≤disp ≤4095)

	n	i	x	b	p	e	
opcode	1	1	1	0	0		disp

n=1, i=1, b=0, p=0, TA=(X)+disp (with index addressing mode)

SIC/XE Machine Architecture

- Immediate Addressing Mode

	n	i	x	b	p	e	
opcode	0	1	0				disp

n=0, i=1, x=0, operand = disp

- Indirect Addressing Mode

	n	i	x	b	p	e	
opcode	1	0	0				disp

n=1, i=0, x=0, TA= (disp)

SIC/XE Machine Architecture

- Simple Addressing Mode

	n	i	x	b	p	e	
opcode	0	0					disp

i=0, n=0, TA= bpe + disp (SIC standard)

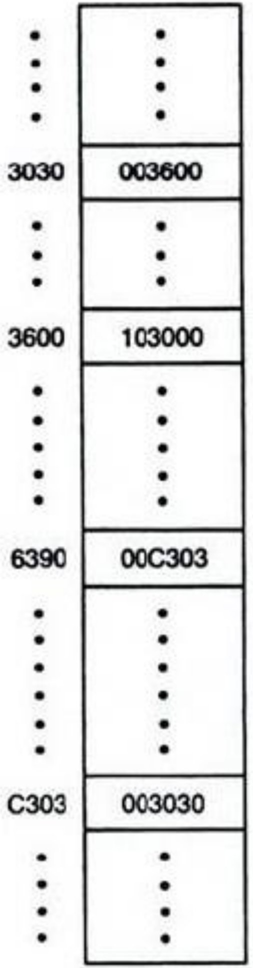
	n	i	x	b	p	e	
opcode	1	1					disp

i=1, n=1, TA = disp (SIC/XE standard)

SIC/XE Machine Architecture

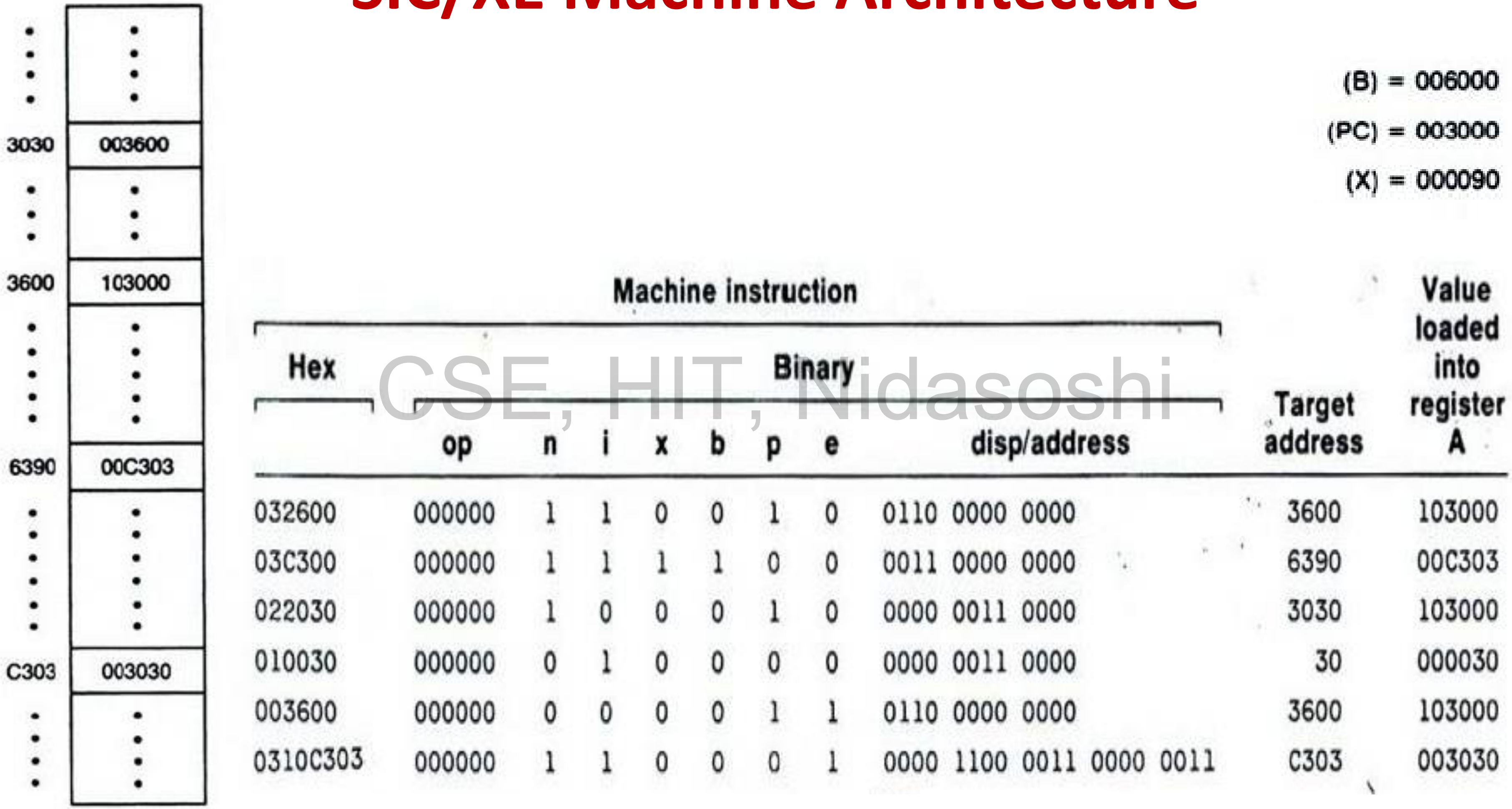
How to convert Hexacode or Object code to Target address

- Calculate the Target address of the following machine instructions.
- Given, (X)=000690, (B)=006030, (PC)=003060



3030	003600
3600	103000
6390	00C303
C303	003030

SIC/XE Machine Architecture



(B) = 006000
(PC) = 003000
(X) = 000090

Machine instruction										
Hex	Binary									
	op	n	i	x	b	p	e	disp/address	Target address	Value loaded into register A
032600	000000	1	1	0	0	1	0	0110 0000 0000	3600	103000
03C300	000000	1	1	1	1	0	0	0011 0000 0000	6390	00C303
022030	000000	1	0	0	0	1	0	0000 0011 0000	3030	103000
010030	000000	0	1	0	0	0	0	0000 0011 0000	30	000030
003600	000000	0	0	0	0	1	1	0110 0000 0000	3600	103000
0310C303	000000	1	1	0	0	0	1	0000 1100 0011 0000 0011	C303	003030

SIC/XE Machine Architecture

- Instruction Set
  - Load and store registers - LDA, LDX, STA, STX, LDB, STB etc.
  - Integer arithmetic operations - ADD, SUB, MUL, DIV
  - Floating-point arithmetic operations: ADDF, SUBF, MULF, DIVF
  - COMP – Comparison instruction
  - Conditional jump instructions - JLT, JEQ, JGT
  - Subroutine linkage - JSUB, RSUB
  - Register move instruction: RMO
  - Register-to-register arithmetic operations: ADDR, SUBR, MULR, DIVR
  - Supervisor call instruction: SVC

SIC/XE Machine Architecture

- I/O (transferring 1 byte at a time to/from the rightmost 8 bits of register A)
  - Test Device instruction (TD)
  - Read Data (RD)
  - Write Data (WD)

CSE, HIT, Nidasoshi

Definition of Assembler

- An assembler is a kind of translator that accepts the input in assembly language program and produces its machine language equivalent.
  - Ex: MASM, TASM

Basic Assembler Functions

1. Convert mnemonic operations code their equivalent machine language.
   - Ex: STL→14, JSUB→48
2. Convert symbolic operands to their equivalent machine address.
   - Ex: Cloop →100
3. Build machine instruction in the proper format (format 3 or format 4).
4. Convert the data constant to internal machine representation.
   - Ex: EF→4546
5. Write the object program and the assembly listing.

Assembler Directives

- These are pseudo instructions,
- They provide definition to the assembler itself.
- They are not translated into machine operation code.
- In addition to the mnemonic machine instruction, we have used the following assembler directives.
- **START, END, BYTE, WORD, RESB, RESW**

Assembler Directives

- **START:** Specify name and starting address for the program.
- **END:** Indicate the end of the source and 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:** Reserves the indicated number of bytes for a data area.
- **RESW:** Reserves the indicated number of words for a data area.

Assembler Algorithms and Data Structures

- Our simpler assembler uses 2 major internal data structures.
  - **OPTAB (Operation Table)**
  - **SYMTAB (Symbol Table)**
  - **LOCCTR (Location Counter)**
- Assembler Algorithms:
  - **PASS - 1 Assembler Algorithm**
  - **PASS - 2 Assembler Algorithm**

Data Structures – LOCCTR Location Counter

- A Location Counter (LOCCTR) is used to be a variable and help in the assignment of addresses.
- LOCCTR initialized to be beginning address specified in the START statement.
- Whenever a label in the source program is read, the current value of LOCCTR gives the address to be associated with that label.
- After each source statement is processed , the length of the assembled instruction or data area to be generated is added to LOCCTR.
- There is certain information (such as location counter values and error flags for statements) that can or should be communicated between the two passes.
- For this reason, Pass 1 usually writes an inter-mediate file that contains each source statement together with its assigned address, error indicators, etc.
- This file is used as the input to Pass 2.

Data Structures - OPTAB (Operation Table)

- It is also one of the internal Data structure.
- It is used to look up mnemonic operation code and translate them to their machine language equivalent.
- In more complex assembler, this table also contains information about instruction format and length.
- During pass 1, OPTAB is used to look up and validate operation codes in the source program.
- During pass 2, it is used to translate the operation codes to machine language.
- For SIC/XE machine, that has instruction of different format, to find the instruction length for incrementing LOCCTR.
- OPTAB is usually organized as a hash table, with mnemonic operation code as the key.
- In most cases, OPTAB is a static table – that is, entries are not normally added to or deleted from it.

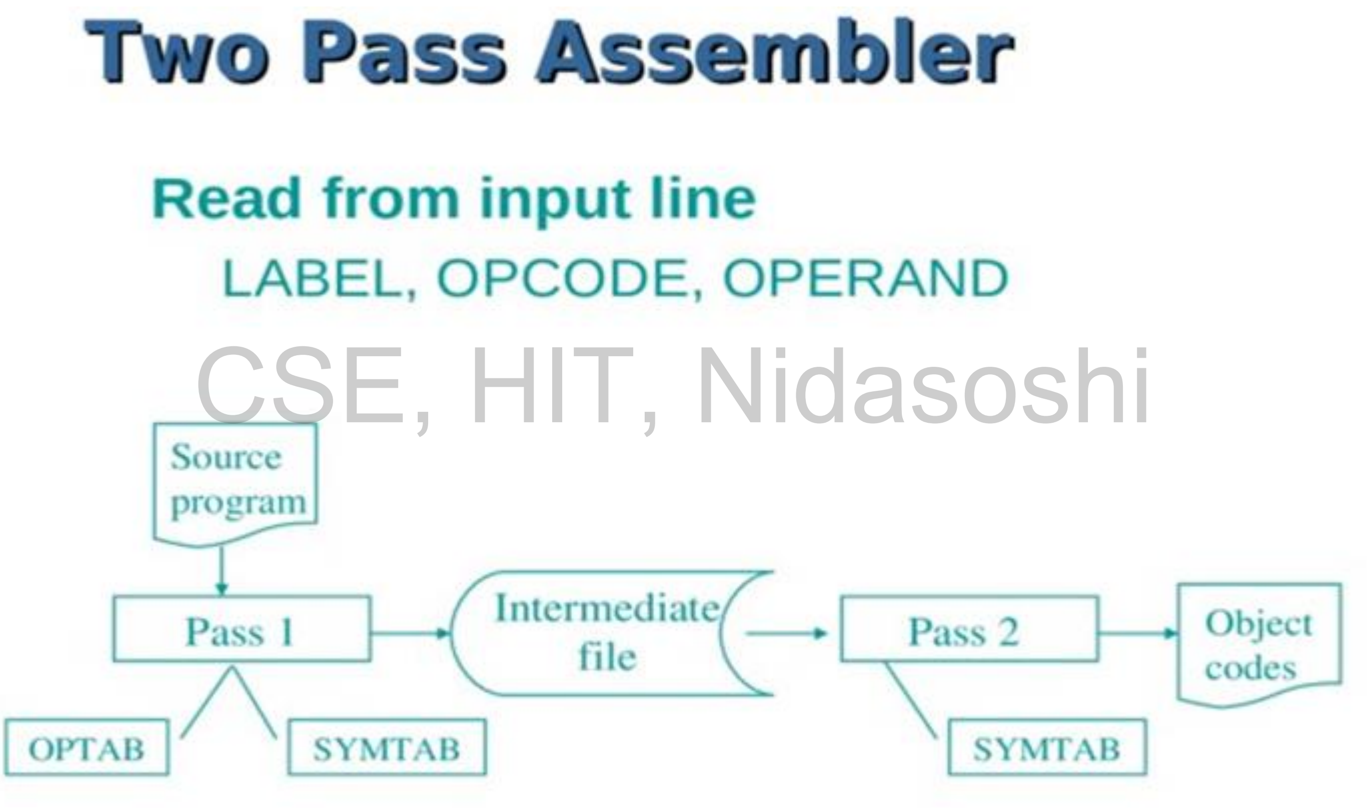
Data Structures - SYMTAB (Symbol Table)

- It is also internal data structures in assembler.
- SYMTAB is used to store values assigned to labels.
- SYMTAB includes the name and value (address) for each label in the source program, together with flags to indicate error condition (e.g., a symbol defined in two different places).
- This table also contain other information about data area.
- During Pass 1, labels are entered into SYMTAB as they are encountered in the source program, along with their assigned addresses (from LOCCTR).
- During Pass 2, symbols used as operands are looked up in SYMTAB to obtain the addresses to be inserted in the assembled instruction.
- SYMTAB is usually organized as a hash table for efficiency of insertion and retrieval.

Functions of the two passes assembler

- Pass 1 (define symbol)
  - Assign addresses to all statements (generate LOC).
  - Save the values (address) assigned to all labels for Pass 2.
  - Perform some processing of assembler directives.
- Pass 2
  - Assemble instructions.
  - Generate data values defined by BYTE, WORD.
  - Perform processing of assembler directives not done during
  - Pass 1.
  - Write the object program and the assembly listing .

Functions of the two passes assembler



Two Pass Assembler

Read from input line

LABEL, OPCODE, OPERAND

Object Program

- The object program (OP) will be loaded into memory for execution.

- Three types of records

  - **Header Record:** program name, starting address, length.
  - **Text Record:** starting address, length, object code.
  - **End Record:** address of first executable instruction.

Records Formats

Header record:

Col. 1	H
Col. 2–7	Program name
Col. 8–13	Starting address of object program (hexadecimal)
Col. 14–19	Length of object program in bytes (hexadecimal)

Records Formats

Text record:

Col. 1	T
Col. 2–7	Starting address for object code in this record(hexadecimal)
Col. 8–9	Length of object code in this record in bytes (hexadecimal)
Col. 10–69	Object code, represented in hexadecimal (2 columns per byte of object code)

End record:

Col. 1	E
Col. 2–7	Address of first executable instruction in object program (hexadecimal)

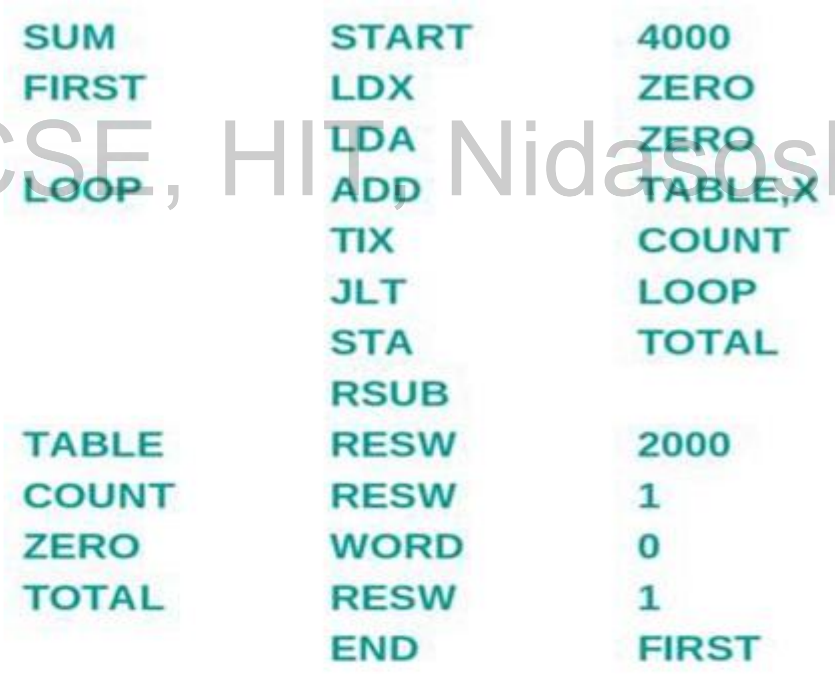
2.2 Machine-Dependent Assembler Features

- Problems on both SIC and SIC/XE machine
- Theory on instruction format and addressing modes , object program .
- SIC/XE

- PC-relative/Base-relative addressing - Ex. op m
- Indirect addressing - Ex. Op @m
- Immediate addressing - Ex. Op #c
- Extended format - Ex. +op m
- Index addressing - Ex. Op m, x
- Register-to-Register instructions - Ex. COMPR s, t
- Larger Memory → multi-programming (program allocation)

Generate Object Program

- Generate the complete object program for the following assembly language program. Assume standard SIC machine and the following machine codes in hexa and also indicate the content of symbol at the end.
- LDA=00, LDX=04, STA=0C, ADD=18, TIX=2C, JLT=38, RSUB=4C



```
SUM      START   4000
FIRST    LDX     ZERO
         LDA     ZERO
LOOP     ADD     TABLE,X
         TIX     COUNT
         JLT     LOOP
         STA     TOTAL
         RSUB
TABLE    RESW    2000
COUNT    RESW    1
ZERO     WORD    0
TOTAL    RESW    1
         END     FIRST
```

Generate Object Program

Line no	LOCATION COUNTER	LABEL	OPCODE	Operand	Object Code
1		SUM	START	4000(H)	-
2	4000	FIRST	LDX	ZERO	045788
3	4003		LDA	ZERO	005788
4	4006	LOOP	ADD	TABLE, X	18CO15
5	4009		TIX	COUNT	2C5785
6	400C		JLT	LOOP	384006
7	400F		STA	TOTAL	0C578B
8	4012		RSUB		4C0000
9	4015	TABLE	RESW	2000	-
10	5785	COUNT	RESW	1	-
11	5788	ZERO	WORD	0	000000
12	578B	TOTAL	RESW	1	-
13	578E		END	FIRST	-

Generate Object Program

- Instruction format -3bytes(24 bits)

Opcode(8 bit)	X(1)	Disp(12)

- SIC add 3 byte to location counter
- RESW→convert decimal to hexa decimal
- LOC= LOC+3*#[operand]

  =4015+3*2000(d) = 4015+3*6000 = 4015+1770 = 5785

- WORD→ add 3 byte to location counter
- RESB→ Convert decimal to hexadecimal
- LOC=LOC+#[operand]
- BYTE→ count the number of character in operand field and add that number to
- Location counter depends on type .
- Length of program=END-START

Generate Object Program

SYMBOL	Address
FIRST	4000
LOOP	4006
TABLE	4015
COUNT	5785
ZERO	5788
TOTAL	578B

Generate Object Program

- Using records write the object program

- **Header Record →only one**
- **Text Record →any number depends on program length**
- **End Record → only one**

- H^SUM_ _ _^004000^00178E
- T^004000^15^045788^005788^18C015^2C5785^384006^0C578B^4C0000
- T^005788^3^000000
- E^004000

Generate Object Program

- Generate the complete object program for the following assembly language program

line	Label	Opcode	Operand
1	SUM	START	3000(H)
2	FIRST	LDX	ZERO
3		LDA	THREE
4	LOOP	ADD	TABLE,X
5		TIX	COUNT
6		JLT	LOOP
7		STA	TOTAL
8		RSUB	
9	THREE	WORD	3
10	TABLE	RESW	100
11	COUNT	RESW	20
12	ZERO	WORD	0
13	TOTAL	RESW	1
14		END	FIRST

Generate Object Program

- Generate the complete object program for the following assembly language program

line	Loc	Label	Opcode	Operand	Object code
1		SUM	START	3000(H)	-
2	3000	FIRST	LDX	ZERO	043180
3	3003		LDA	THREE	003015
4	3006	LOOP	ADD	TABLE,X	18B018
5	3009		TIX	COUNT	2C3144
6	300C		JLT	LOOP	383006
7	300F		STA	TOTAL	0C3183
8	3012		RSUB		4C0000
9	3015	THREE	WORD	3	000003
10	3018	TABLE	RESW	100	-
11	3144	COUNT	RESW	20	-
12	3180	ZERO	WORD	0	000000
13	3183	TOTAL	RESW	1	-
14	3186		END	FIRST	-

Generate Object Program

- H^SUM- - -^003000^000186
- T^003000^18^043180^003015^18B018^2C3144^38 3006^0C3183^4C0000^000003
- T^003180^03^000000
- E^003000

PASS 1 and PASS 2 Assemblers

CSE, HIT, Nidasoshi

Generate Object Program

- Object program for text book problem using SIC
- **Data transfer (RD, WD)**
  - A buffer is used to store record
  - Buffering is necessary for different I/O rates
  - The end of each record is marked with a null character (00_{16})
  - Buffer length is 4096 Bytes
  - The end of the file is indicated by a zero-length record
- **Subroutines (JSUB, RSUB)**
  - RDREC, WRREC
  - Save link (L) register first before nested jump

Generate Object Program

- **Below Figure 2.2 shows the generated object code for each statement.**
  - Loc gives the machine address in Hex.
  - Assume the program starting at address 1000.
- **Translation functions**
  - Translate STL to 14.
  - Translate RETADR to 1033.
  - Build the machine instructions in the proper format (,X).
  - Translate EOF to 454F46.
  - Write the object program and assembly listing.

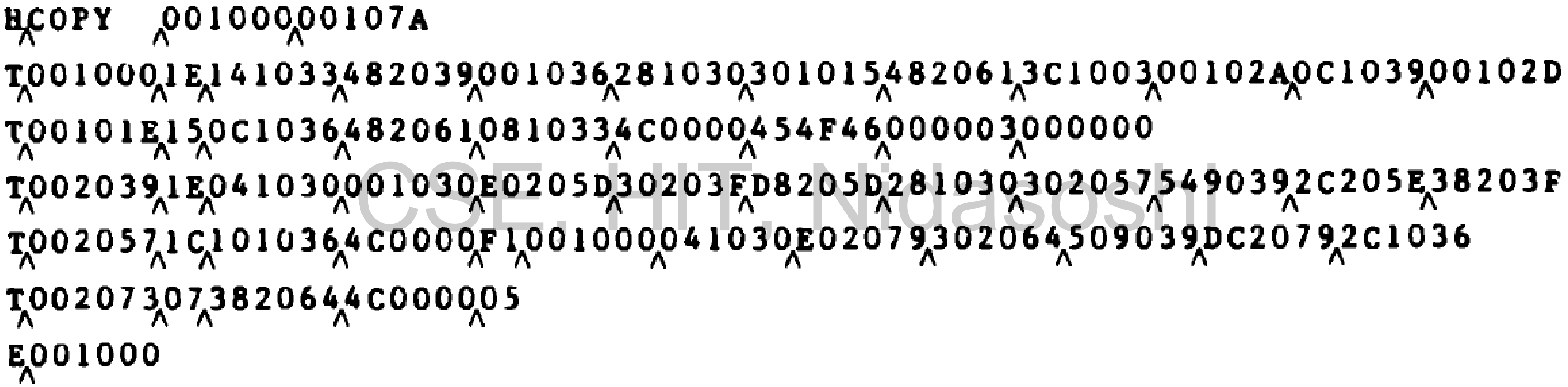
Line	Loc	Source statement			Object code
5	1000	COPY	START	1000	
10	1000	FIRST	STL	RETADR	141033
15	1003	CLOOP	JSUB	RDREC	482039
20	1006		LDA	LENGTH	001036
25	1009		COMP	ZERO	281030
30	100C		JEQ	ENDFIL	301015
35	100F		JSUB	WRREC	482061
40	1012		J	CLOOP	3C1003
45	1015	ENDFIL	LDA	EOF	00102A
50	1018		STA	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	

```
Line    Loc        Source statement                          Object code

110              .
115              .       SUBROUTINE TO READ RECORD INTO BUFFER
120              .
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
195              .
```

```
Line   Loc    Source statement                          Object code

200           .        SUBROUTINE TO WRITE RECORD FROM BUFFER
205           .
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
```

Figure 2.2 Program from Fig. 2.1 with object code.

Object program



```
HCOPY ^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
```

- A forward reference
  - 10 1000 FIRST STL RETADR 141033
  - A reference to a label (RETADR) that is defined later in the program
  - Most assemblers make two passes over the source program
- **Most assemblers make two passes over source program.**
  - Pass 1 scans the source for label definitions and assigns address (Loc).
  - Pass 2 performs most of the actual translation.

Problems on SIC/XE machine

- Register translation
  - register name (A, X, L, B, S, T, F, PC, SW) and their values (0, 1, 2, 3, 4, 5, 6, 8, 9)
  - preloaded in SYMTAB
- Address translation
  - Most register-memory instructions use program counter relative or base relative addressing
  - Format 3: 12-bit disp (address) field
    - Base-relative: 0~4095
    - PC-relative: -2048~2047
  - Format 4: 20-bit address field (absolute addressing)

- The START statement
  - Specifies a beginning address of 0.
- Register-register instructions
  - CLEAR & TIXR, COMPR
- Register-memory instructions are using
  - Program-counter (PC) relative addressing
  - The program counter is advanced *after* each instruction is fetched and *before* it is executed.
  - PC will contain the address of the *next* instruction.

10 0000 FIRST STL RETADR 17202D

TA - (PC) = disp = 30 - 3 = 2D

Object program on SIC/XE machine

LINE	LOC	LABEL	OPCODE	OPERAND	OBJECT CODE
1		WRREC	START	105D	
2			CLEAR	X	
3			LDT	LENGTH	
4		WLOOP	TD	OUTPUT	
5			JEQ	WLOOP	
6			LDCH	BUFFER,X	
7			WD	OUTPUT	
8			TIXR	T	
9			JLT	WLOOP	
10			RSUB		
11		OUTPUT	BYTE	X’05’	
12		BUFFER	RESB	400	
13		LENGTH	RESB	2	
14			END	WRREC	

LINE	LOC	LABEL	OPCODE	OPERAND	OBJECT CODE
1		WRREC	START	105D	-
2	105D		CLEAR	X	B410
3	105F		LDT	LENGTH	7721A5
4	1062	WLOOP	TD	OUTPUT	E32011
5	1065		JEQ	WLOOP	332FFA
6	1068		LDCH	BUFFER,X	53A00C
7	106B		WD	OUTPUT	DF2008
8	106E		TIXR	T	B850
9	1070		JLT	WLOOP	3B2FEF
10	1073		RSUB		4F0000
11	1076	OUTPUT	BYTE	X’05’	05
12	1077	BUFFER	RESB	400	-
13	1207	LENGTH	RESB	2	-
14	1209		END	WRREC	-

Object code calculation

1. Check the instruction format
2. If format 3 , check program counter relative address and base relative address for displacement calculation.
3. Format 1 and format 2 not required address.
4. Remember the n and i bit . # (n=0, i=1), @(n=1, i=0)
5. Remember the mnemonic number of register
6. EX: CLEAR X b410 (clear-B4)(X=1) (r2-absent)

Object Program

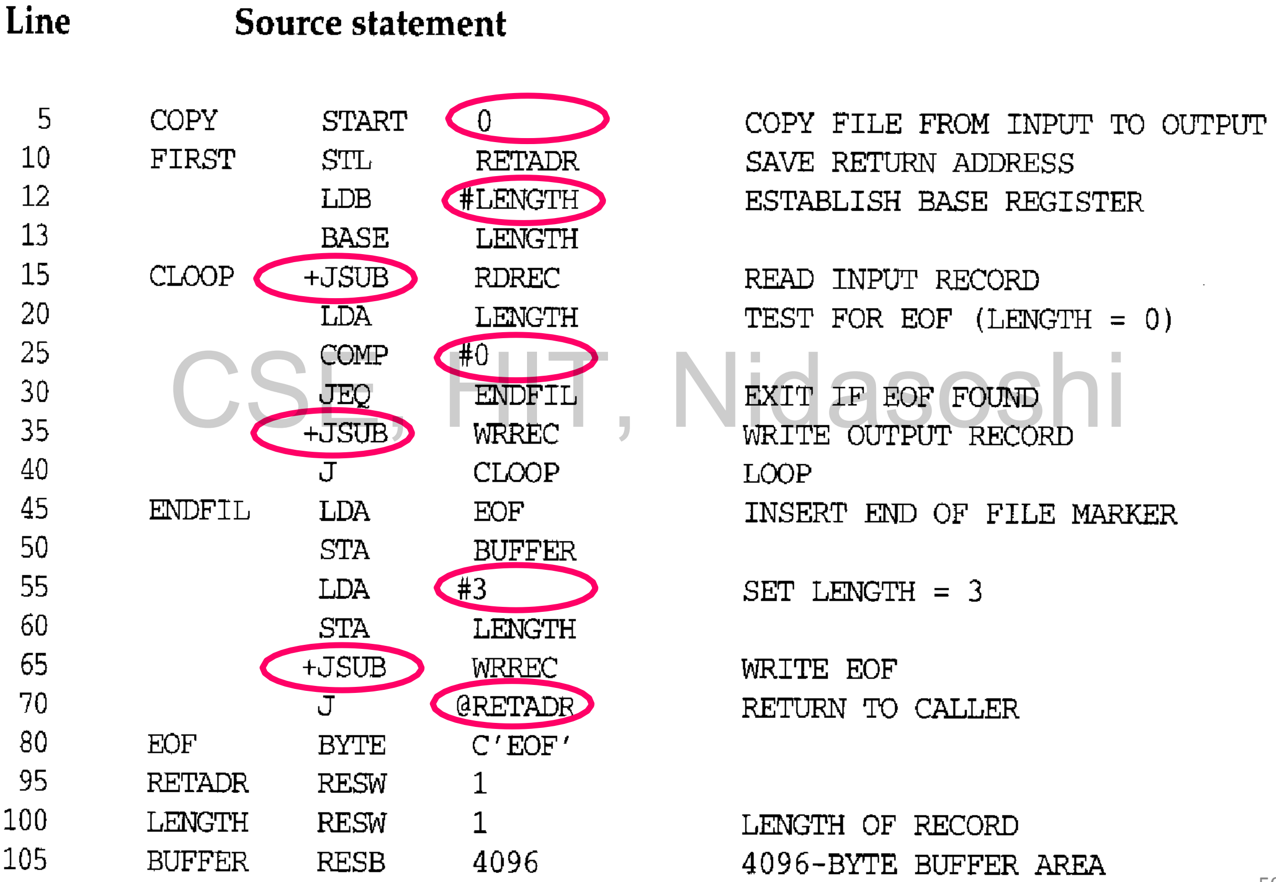
- H^WRREC_^00105D^0001AC
- T^00105D^1A^00B410^7721a5^E32011^332FFA^53A00C^DF2008^00B850^3B2FEF^4F000^ 05
- E^00105D

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

LINE	LOC	LABEL	OPCODE	OPERAND	OBJECT CODE
1		SUM	START	0	
2	0000	FIRST	LDX	#0	050000
3	0003		LDA	#0	010000
4	0006		+LDB	#TABLE2	69101790
5			BASE	TABLE2	-
6	000A	LOOP	ADD	TABLE,X	1BA013
7	000D		ADD	TABLE2,X	1BC000
8	0010		TIX	COUNT	2F200A
9	0013		JLT	LOOP	3B2FF4
10	0016		+STA	TOTAL	0F102F00
11	001A		RSUB		4F0000
12	001D	COUNT	RESW	1	-
13	0020	TABLE	RESW	2000	-
14	1790	TABLE2	RESW	2000	-
15	2F00	TOTAL	RESW	1	-
16	2F03		END	FIRST	-

OBJECT PROGRAM

```
H^SUM- - - ^000000^002F03

T^000000^1D^050000^010000^69101790^1BA013^1BC000^2F200A^3B2FF4^0F
102F00^4F0000

E^000000
```



```
Line          Source statement

  5    COPY     START   0          COPY FILE FROM INPUT TO OUTPUT
 10    FIRST    STL     RETADR     SAVE RETURN ADDRESS
 12             LDB     #LENGTH    ESTABLISH BASE REGISTER
 13             BASE    LENGTH
 15    CLOOP   +JSUB    RDREC      READ INPUT RECORD
 20             LDA     LENGTH     TEST FOR EOF (LENGTH = 0)
 25             COMP    #0
 30             JEQ     ENDFIL     EXIT IF EOF FOUND
 35            +JSUB    WRREC      WRITE OUTPUT RECORD
 40             J       CLOOP      LOOP
 45    ENDFIL   LDA     EOF        INSERT END OF FILE MARKER
 50             STA     BUFFER
 55             LDA     #3         SET LENGTH = 3
 60             STA     LENGTH
 65            +JSUB    WRREC      WRITE EOF
 70             J       @RETADR    RETURN TO CALLER
 80    EOF      BYTE    C'EOF'
 95    RETADR   RESW    1
100    LENGTH   RESW    1          LENGTH OF RECORD
105    BUFFER   RESB    4096       4096-BYTE BUFFER AREA
```

```
110      .
115      .         SUBROUTINE TO READ RECORD INTO BUFFER
120      .
125      RDREC     CLEAR     X                CLEAR LOOP COUNTER
130                CLEAR     A                CLEAR A TO ZERO
132                CLEAR     S                CLEAR S TO ZERO
133               +LDT       #4096
135      RLOOP     TD        INPUT            TEST INPUT DEVICE
140                JEQ       RLOOP            LOOP UNTIL READY
145                RD        INPUT            READ CHARACTER INTO REGISTER A
150                COMPR     A,S              TEST FOR END OF RECORD (X'00')
155                JEQ       EXIT             EXIT LOOP IF EOR
160                STCH      BUFFER,X         STORE CHARACTER IN BUFFER
165                TIXR      T                LOOP UNLESS MAX LENGTH
170                JLT       RLOOP              HAS BEEN REACHED
175      EXIT      STX       LENGTH           SAVE RECORD LENGTH
180                RSUB                       RETURN TO CALLER
185      INPUT     BYTE      X'F1'            CODE FOR INPUT DEVICE
```

```
195     .
200     .          SUBROUTINE TO WRITE RECORD FROM BUFFER
205     .
210     WRREC      CLEAR     X                CLEAR LOOP COUNTER
212                LDT       LENGTH
215     WLOOP      TD        OUTPUT           TEST OUTPUT DEVICE
220                JEQ       WLOOP            LOOP UNTIL READY
225                LDCH      BUFFER,X         GET CHARACTER FROM BUFFER
230                WD        OUTPUT           WRITE CHARACTER
235                TIXR      T                LOOP UNTIL ALL CHARACTERS
240                JLT       WLOOP              HAVE BEEN WRITTEN
245                RSUB                       RETURN TO CALLER
250     OUTPUT     BYTE      X'05'            CODE FOR OUTPUT DEVICE
255                END       FIRST
```

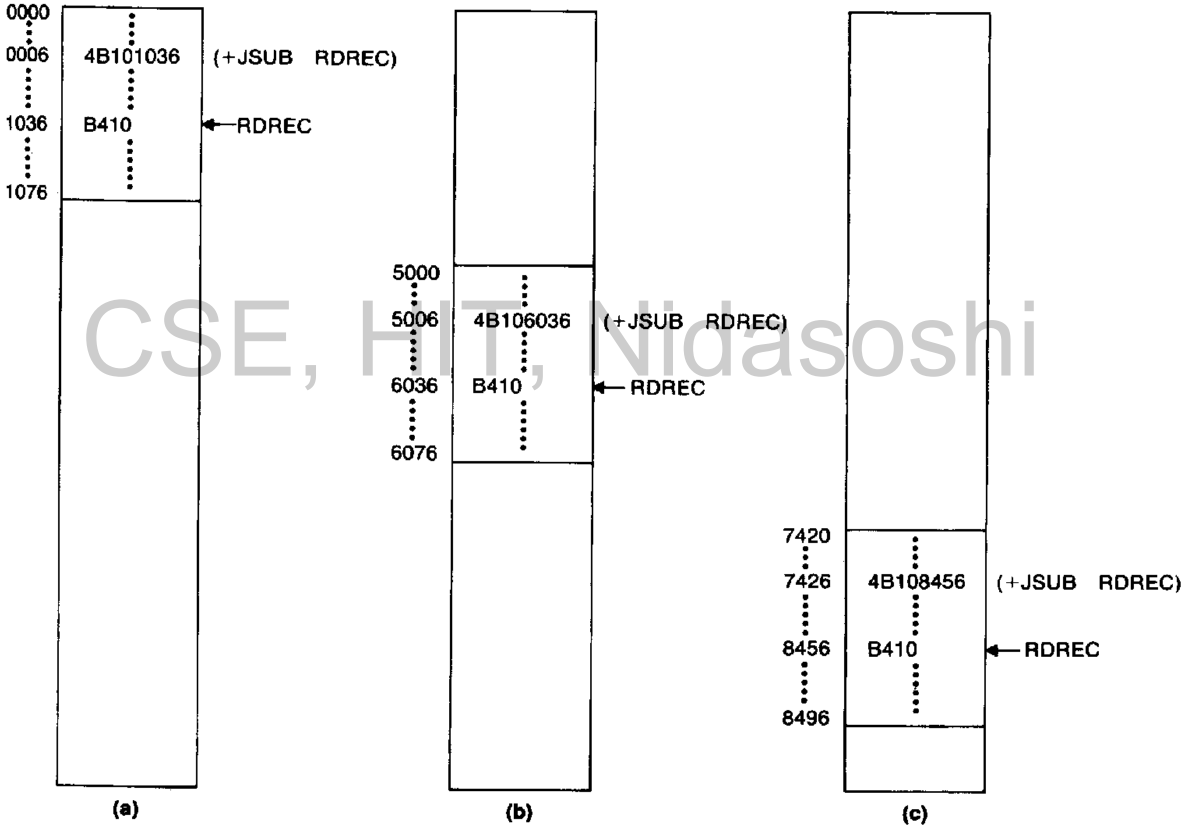
Figure 2.5 Example of a SIC/XE program.

Program Relocation (VVIMP)

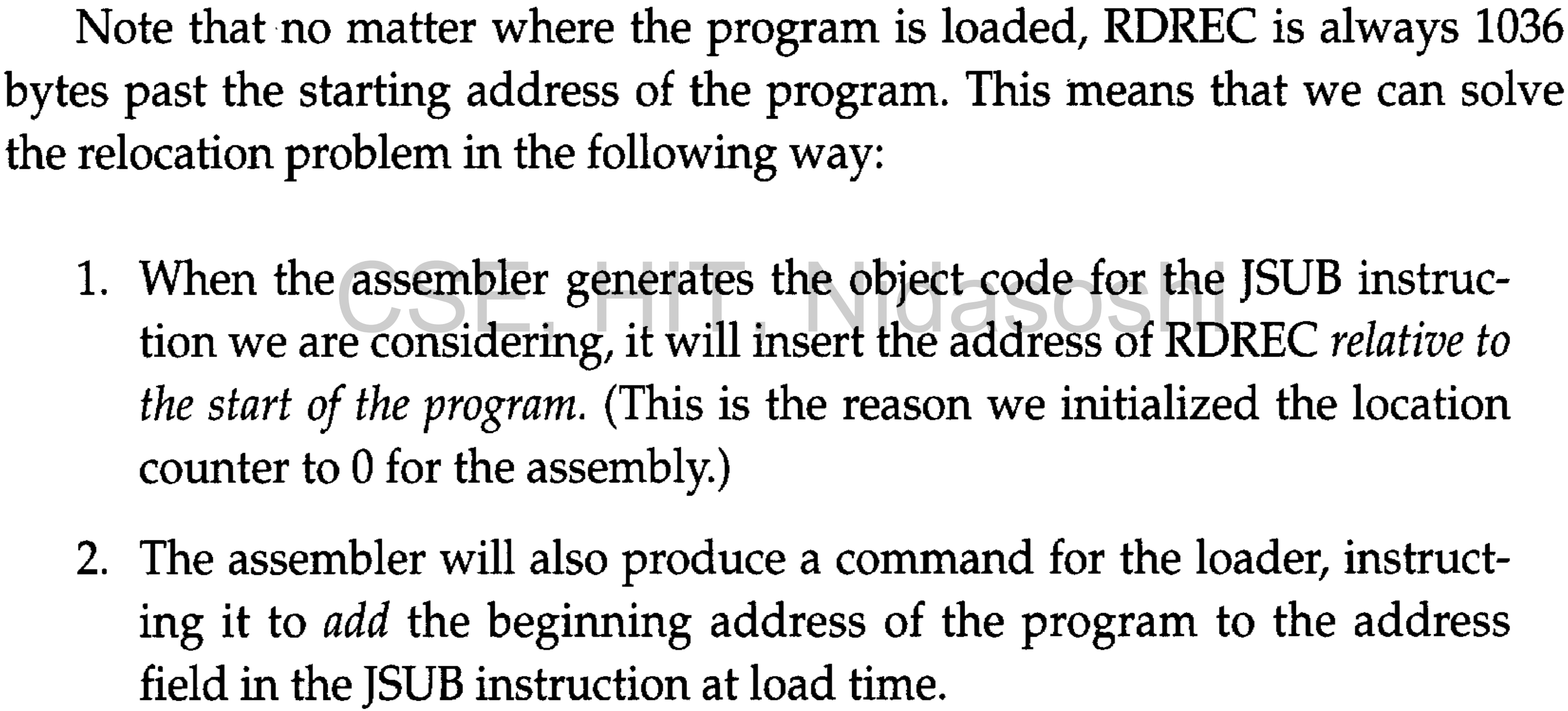
- In a typical multiprogramming environment where multiple programs can run simultaneously , there is no guarantee that program will get loaded at a particular memory location.
- Rather programs are time sharing not only memory but other resources including **CPU**.
- Because of the above mentioned scenario, there is no way that assembler can not prepare object program with reference to final location.
- Rather, to load a program into memory whenever there is a room for it , the actual starting address of the program is not known until load time, only loader knows this.

- The assembler does not know the actual location where the program will be loaded it cannot make the necessary changes in the addressing used by the program.
- But the assembler needs to save information of address sensitive locations in the object program.
- So that loader can take proper decision when program needs to be loaded or assembler can identify for the loader those parts of the object program that need modification.
- An object program that contains the information necessary to perform this kind of modification is called a **“Relocatable Program”.**
- Loader that allow for program relocation are called relocating loader or relative loader.

Program Relocation



Solution of Program Relocation

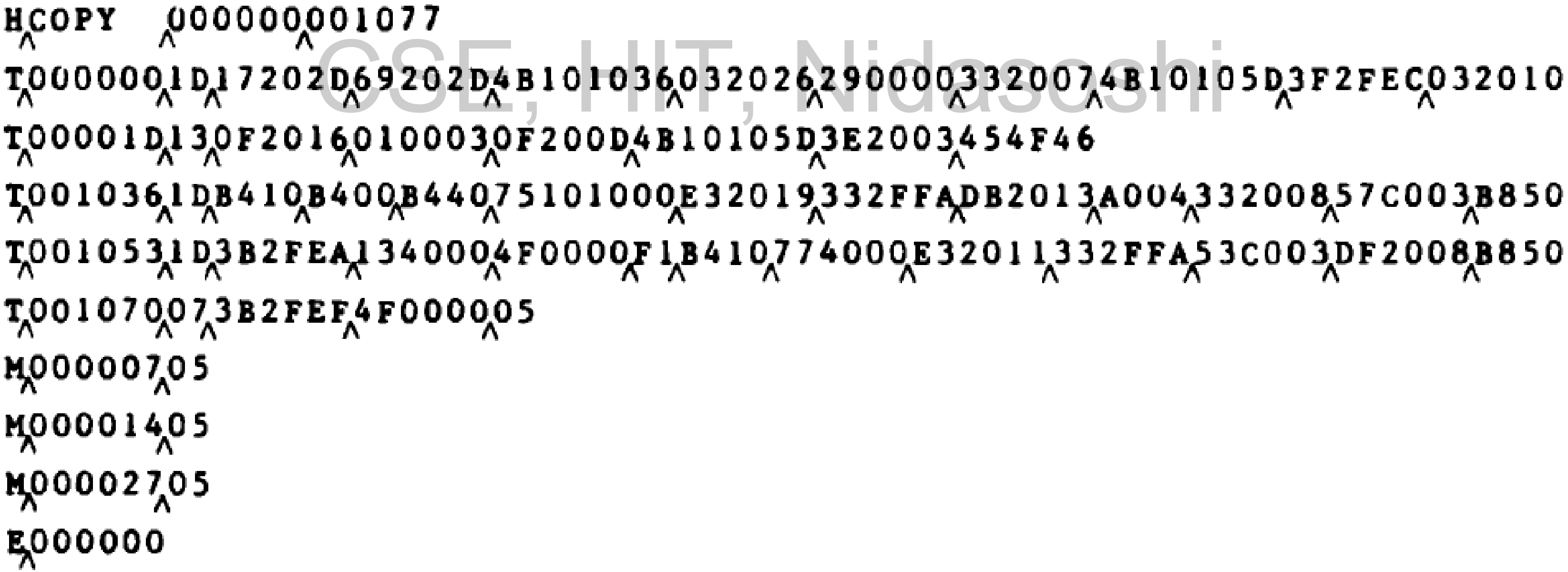


Note that no matter where the program is loaded, RDREC is always 1036 bytes past the starting address of the program. This means that we can solve the relocation problem in the following way:

1. When the assembler generates the object code for the JSUB instruction we are considering, it will insert the address of RDREC *relative to the start of the program.* (This is the reason we initialized the location counter to 0 for the assembly.)
2. The assembler will also produce a command for the loader, instructing it to *add* the beginning address of the program to the address field in the JSUB instruction at load time.

Modification record (direct addressing)

- 1 M
- 2-7 Starting location of the address field to be modified, relative to the beginning of the program.
- 8-9 Length of the address field to be modified, in *half bytes*.



```
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
```

2.3.1 Literals

- 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.

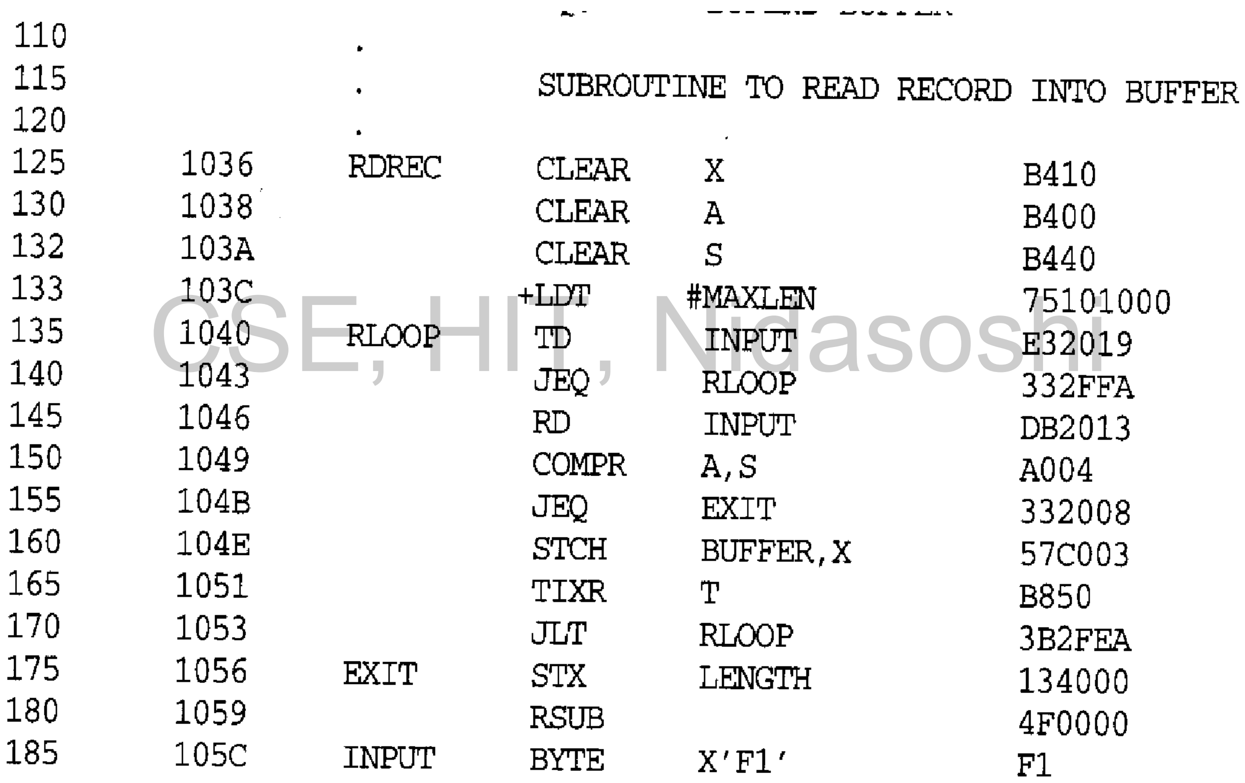
2.3.1 Literals

- The difference between literal and immediate
  - Immediate addressing, the operand value is assembled as part of the machine instruction, no memory reference.
  - With a literal, the assembler generates the specified value as a constant at some other memory location. The *address* of this generated constant is used as the TA for the machine instruction, using PC-relative or base- relative addressing with memory reference.

- **Literal pools**
  - It is delcared at the end of the program (Fig. 2.10).
  - Assembler directive LTORG, it creates a literal pool that contains all of the literal operands used since the previous LTORG.

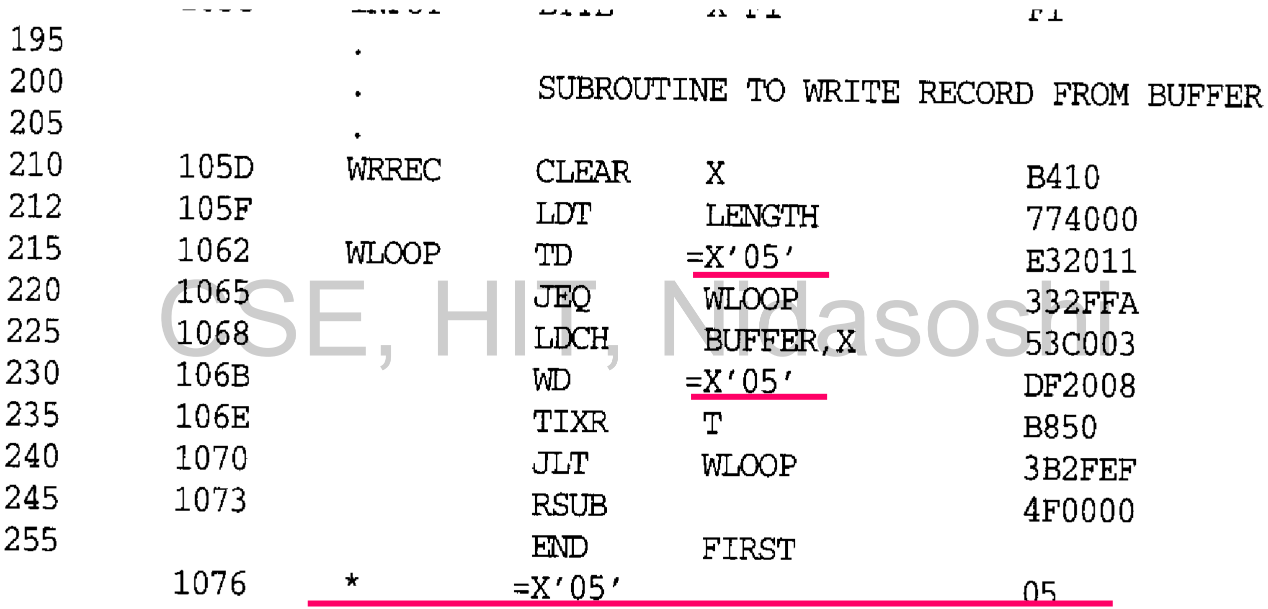
Line	Loc	Source statement			Object code
5	0000	COPY	START	0	
10	0000	FIRST	STL	RETADR	17202D
13	0003		LDB	#LENGTH	69202D
14			BASE	LENGTH	
15	0006	CLOOP	+JSUB	RDREC	4B101036
20	000A		LDA	LENGTH	032026
25	000D		COMP	#0	290000
30	0010		JEQ	ENDFIL	332007
35	0013		+JSUB	WRREC	4B10105D
40	0017		J	CLOOP	3F2FEC
45	001A	ENDFIL	LDA	=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	
100	0033	LENGTH	RESW	1	
105	0036	BUFFER	RESB	4096	
106	1036	BUFEND	EQU	*	
107	1000	MAXLEN	EQU	BUFEND-BUFFER	

RDREC



```
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
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
```

WRREC



```
195                 .
200                 .        SUBROUTINE TO WRITE RECORD FROM BUFFER
205                 .
210     105D    WRREC    CLEAR    X                  B410
212     105F             LDT      LENGTH             774000
215     1062    WLOOP    TD       =X'05'             E32011
220     1065             JEQ      WLOOP              332FFA
225     1068             LDCH     BUFFER,X           53C003
230     106B             WD       =X'05'             DF2008
235     106E             TIXR     T                  B850
240     1070             JLT      WLOOP              3B2FEF
245     1073             RSUB                        4F0000
255                      END      FIRST
        1076    *       =X'05'                       05
```

Figure 2.10 Program from Fig. 2.9 with object code.

Literals-Continue

- When to use LTORG
  - ❖The literal operand would be placed too far away from the instruction referencing.
  - ❖Cannot use PC-relative addressing or Base-relative addressing to generate Object Program.

- Most assemblers recognize duplicate literals.
  - ❖By comparison of the character strings defining them.
  - ❖ =C’EOF’ and =X’454F46’

Literals-Continue

- **Literal table (LITTAB)**
  - Contains the literal name (=C’EOF’), the operand value (454F46) and length (3), and the address (002D).
  - Organized as a hash table.
  - Pass 1, the assembler searches LITTAB for the specified literal name.
  - Pass 1 encounters a LTORG statement or the end of the program, the assembler makes a scan of the literal table.
  - Pass 2, the operand address for use in generating OC is obtained by searching LITTAB.

2.3.2 Symbol-Defining Statements

- The standard names reflect the usage of the registers.

```
BASE    EQU    R1
COUNT   EQU    R2
INDEX   EQU    R3
```

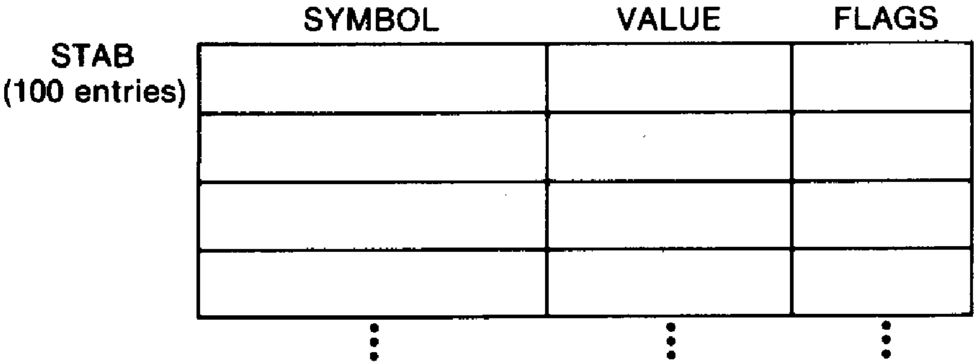
- Assembler directive ORG
  - ❖Use to indirectly assign values to symbols.

```
ORG    value
```

  - ❖The assembler resets its LOCCTR to the specified value.
  - ❖ORG can be useful in label definition.

Symbol-Defining Statements

- The location counter is used to control assignment of storage in the object Program
- In most cases, altering its value would result in an incorrect assembly.
- ORG is used
- SYMBOL is 6-byte, VALUE is 3-byte, and FLAGS is 2-byte.



2.3.3 Expressions

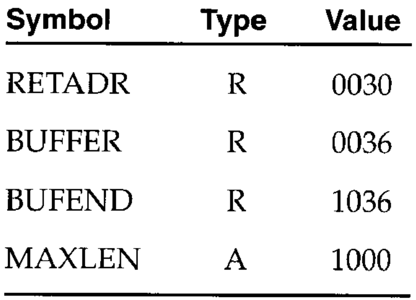
- Allow arithmetic expressions formed
  - ❖ Using the operators +, -, *, /.
  - ❖ Division is usually defined to produce an integer result.
  - ❖ Expression may be constants, user-defined symbols, or special terms.
  - ❖ 106 1036 BUFEND EQU *
  - ❖ Gives BUFEND a value that is the address of the next byte after the buffer area.
- **Absolute expressions or relative expressions**
  - ❖ A relative term or expression represents some value (S+r), S: starting address, r: the relative value.

2.3.3 Expressions

```
107     1000  MAXLEN     EQU     BUFEND-BUFFER
```

- Both BUFEND and BUFFER are relative terms.
- The expression represents absolute value: the *difference* between the two addresses.
- Loc =1000 (Hex)
- The value that is associated with the symbol that appears in the source statement.
- BUFEND+BUFFER, 100-BUFFER, 3*BUFFER represent neither absolute values nor locations.

- Symbol tables entries



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

2.3.4 Program Blocks

- Three blocks, Figure 2.11
  - ❖ Default, CDATA, CBLKS.
- Assembler directive USE
  - ❖ Indicates which portions of the source program blocks.
  - ❖ At the beginning of the program, statements are assumed to be part of the default block.
  - ❖ Lines 92, 103, 123, 183, 208, 252.
- Each program block may contain several separate segments.
  - ❖ The assembler will rearrange these segments to gather together the pieces of each block.

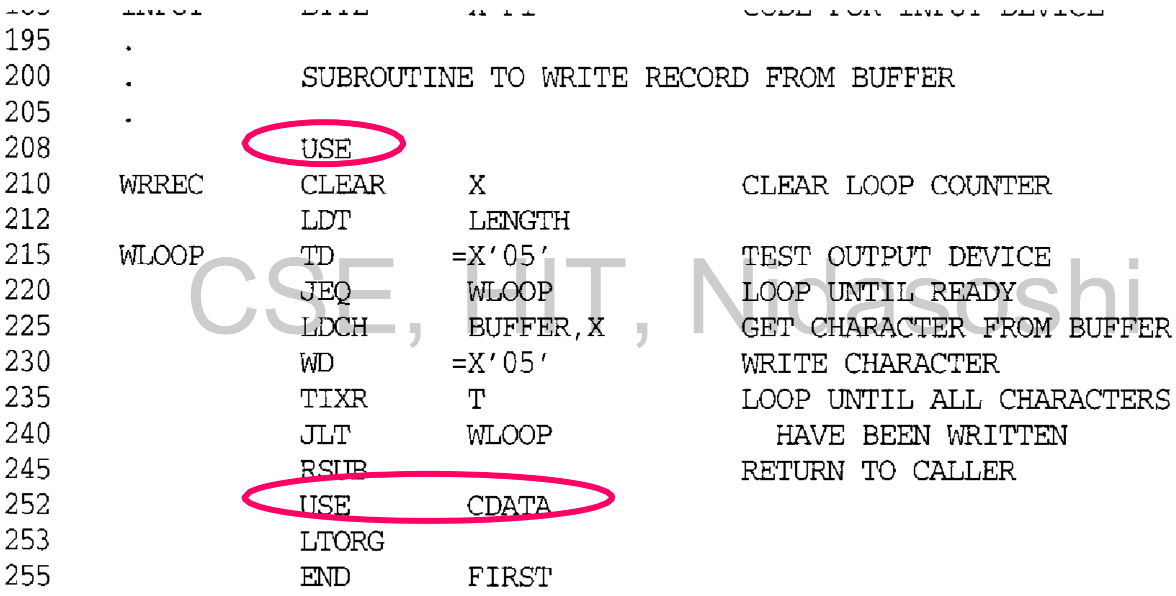
Main

```
Line        Source statement

  5    COPY      START   0               COPY FILE FROM INPUT TO OUTPUT
 10    FIRST     STL     RETADR          SAVE RETURN ADDRESS
 15    CLOOP     JSUB    RDREC           READ INPUT RECORD
 20              LDA     LENGTH          TEST FOR EOF (LENGTH = 0)
 25              COMP   #0
 30              JEQ     ENDFIL          EXIT IF EOF FOUND
 35              JSUB    WRREC           WRITE OUTPUT RECORD
 40              J       CLOOP           LOOP
 45    ENDFIL    LDA    =C'EOF'          INSERT END OF FILE MARKER
 50              STA     BUFFER
 55              LDA    #3               SET LENGTH = 3
 60              STA     LENGTH
 65              JSUB    WRREC           WRITE EOF
 70              J      @RETADR          RETURN TO CALLER
 92              USE     CDATA
 95    RETADR    RESW    1
100    LENGTH    RESW    1               LENGTH OF RECORD
103              USE     CBLKS
105    BUFFER    RESB    4096            4096-BYTE BUFFER AREA
106    BUFEND    EQU     *               FIRST LOCATION AFTER BUFFER
107    MAXLEN    EQU     BUFEND-BUFFER   MAXIMUM RECORD LENGTH
```

RDREC

```
110        .
115        .          SUBROUTINE TO READ RECORD INTO BUFFER
120        .
123                   USE
125        RDREC      CLEAR     X                      CLEAR LOOP COUNTER
130                   CLEAR     A                      CLEAR A TO ZERO
132                   CLEAR     S                      CLEAR S TO ZERO
133                  +LDT      #MAXLEN
135        RLOOP      TD        INPUT                  TEST INPUT DEVICE
140                   JEQ       RLOOP                  LOOP UNTIL READY
145                   RD        INPUT                  READ CHARACTER INTO REGISTER A
150                   COMPR     A,S                    TEST FOR END OF RECORD (X'00')
155                   JEQ       EXIT                   EXIT LOOP IF EOR
160                   STCH      BUFFER,X               STORE CHARACTER IN BUFFER
165                   TIXR      T                      LOOP UNLESS MAX LENGTH
170                   JLT       RLOOP                    HAS BEEN REACHED
175        EXIT       STX       LENGTH                 SAVE RECORD LENGTH
180                   RSUB                             RETURN TO CALLER
183                   USE       CDATA
185        INPUT      BYTE      X'F1'                  CODE FOR INPUT DEVICE
```

WRREC



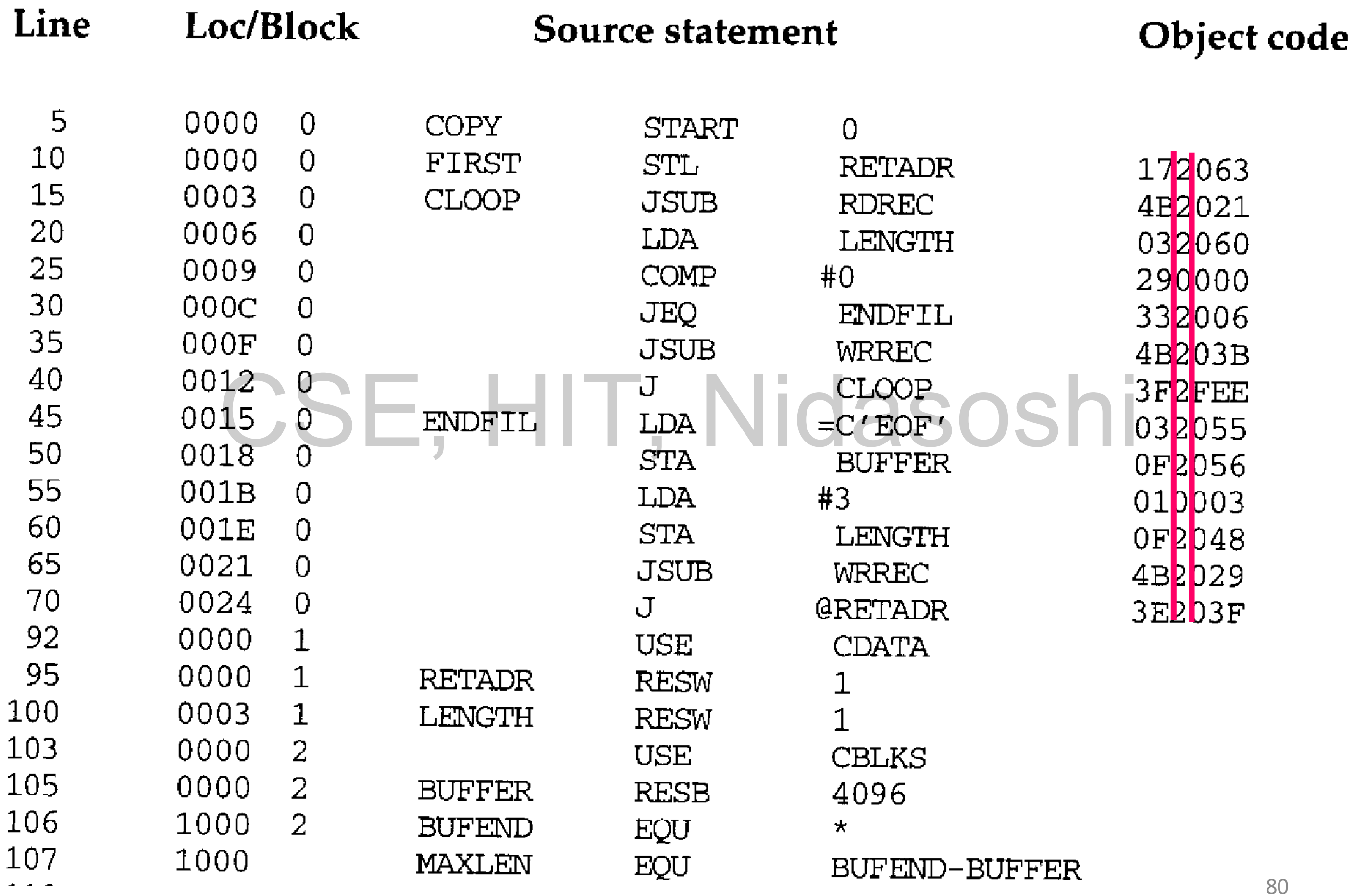
```
        .
        .          SUBROUTINE TO WRITE RECORD FROM BUFFER
        .
                   USE
        WRREC      CLEAR      X               CLEAR LOOP COUNTER
                   LDT        LENGTH
        WLOOP      TD        =X'05'           TEST OUTPUT DEVICE
                   JEQ        WLOOP           LOOP UNTIL READY
                   LDCH       BUFFER,X        GET CHARACTER FROM BUFFER
                   WD        =X'05'           WRITE CHARACTER
                   TIXR       T               LOOP UNTIL ALL CHARACTERS
                   JLT        WLOOP             HAVE BEEN WRITTEN
                   RSUB                       RETURN TO CALLER
                   USE        CDATA
                   LTORG
                   END        FIRST
```

Figure 2.11 Example of a program with multiple program blocks.

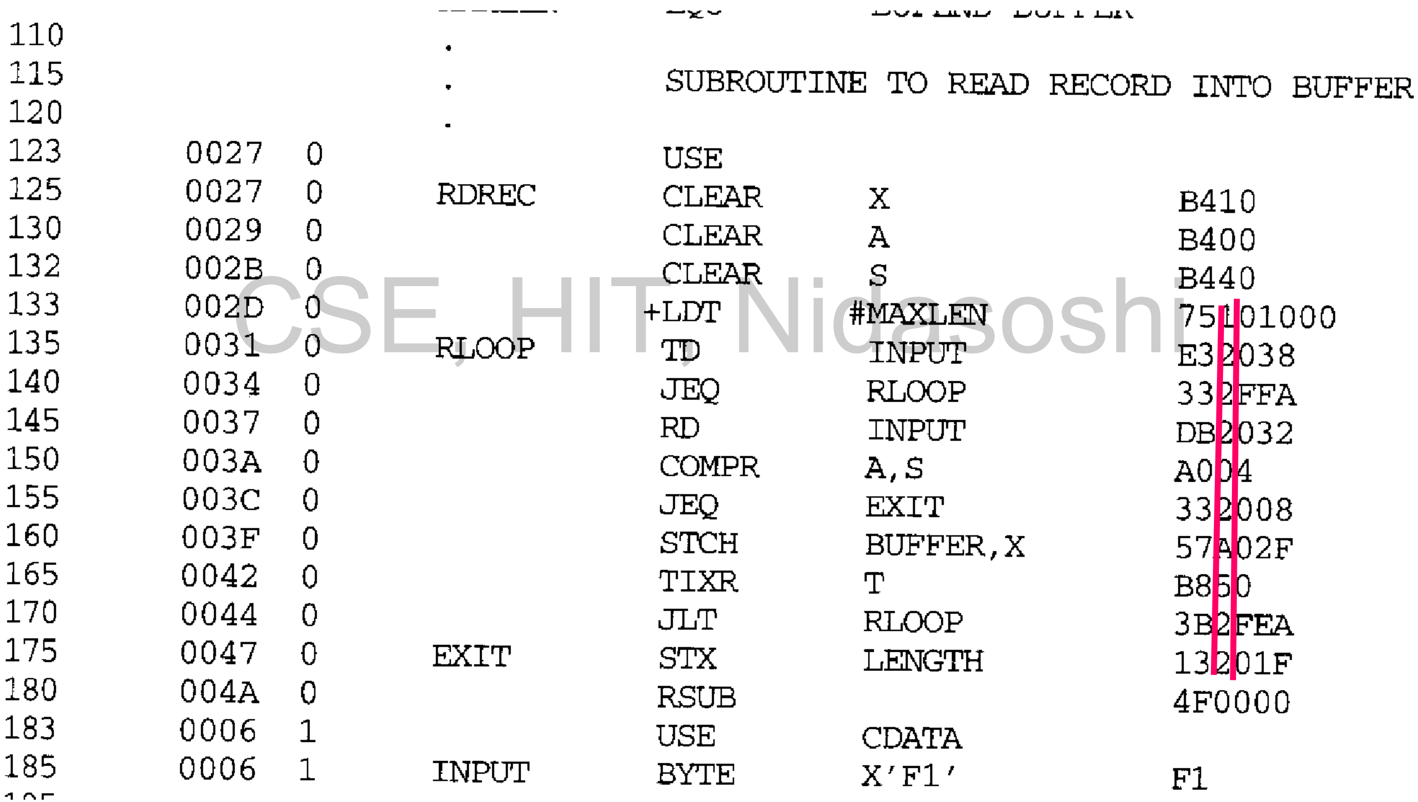
Program Blocks-continue

- **Pass 1, Figure 2.12**
  - ❖ A separate location counter for each program block.
  - ❖ The location counter for a block is initialized to 0 when the block is first begun.
  - ❖ Assign each block a starting address in the object program (location 0).
  - ❖ Labels, block name or block number, relative address
  - ❖ Working table

Block name	Block number	Address	Length
(default)	0	0000	0066 (0~65)
CDATA	1	0066	000B (0~A)
CBLKS	2	0071	1000 (0~0FFF)



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	



```
110                    .
115                    .          SUBROUTINE TO READ RECORD INTO BUFFER
120                    .
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
```

```
195                    .
200                    .           SUBROUTINE TO WRITE RECORD FROM BUFFER
205                    .
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
```

Figure 2.12 Program from Fig. 2.11 with object code.

Program Blocks

- **Pass 2, Figure 2.12**
  - The assembler needs the address for each symbol relative to the start of the object program.
  - Loc shows the relative address and block number.
  - Notice that the value of the symbol MAXLEN (line 70) is shown without a block number.

20 0006 0 LDA LENGTH 032060

0003(CDATA) +0066 =0069 =TA

using program-counter relative addressing

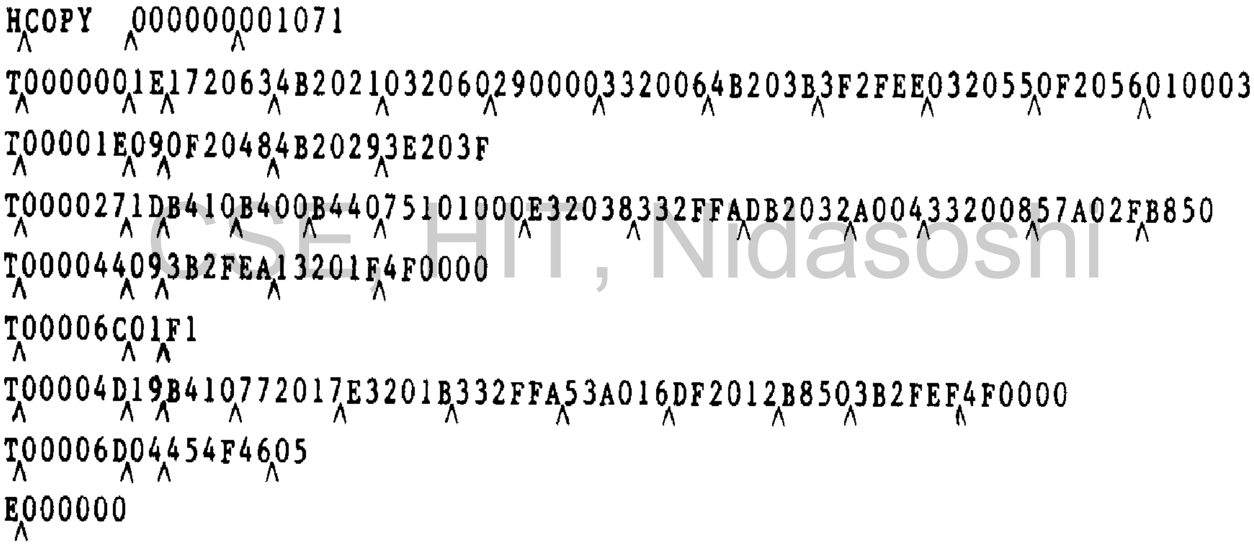
TA - (PC) =0069-0009 =0060 =disp

Program Blocks

- Separation of the program into blocks.
  - ❖ Because the large buffer is moved to the end of the object program.
  - ❖ No longer need extended format, base register, simply a LTORG statement.
  - ❖ No need Modification records.
  - ❖ Improve program readability.

- Figure 2.13
  - ❖ Reflect the starting address of the block as well as the relative location of the code within the block.
- Figure 2.14
  - ❖ Loader simply loads the object code from each record at the dictated.
  - ❖ CDATA(1) & CBLKS(1) are not actually present in Object program.

Program Blocks-object program



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

Figure 2.13 Object program corresponding to Fig. 2.11.

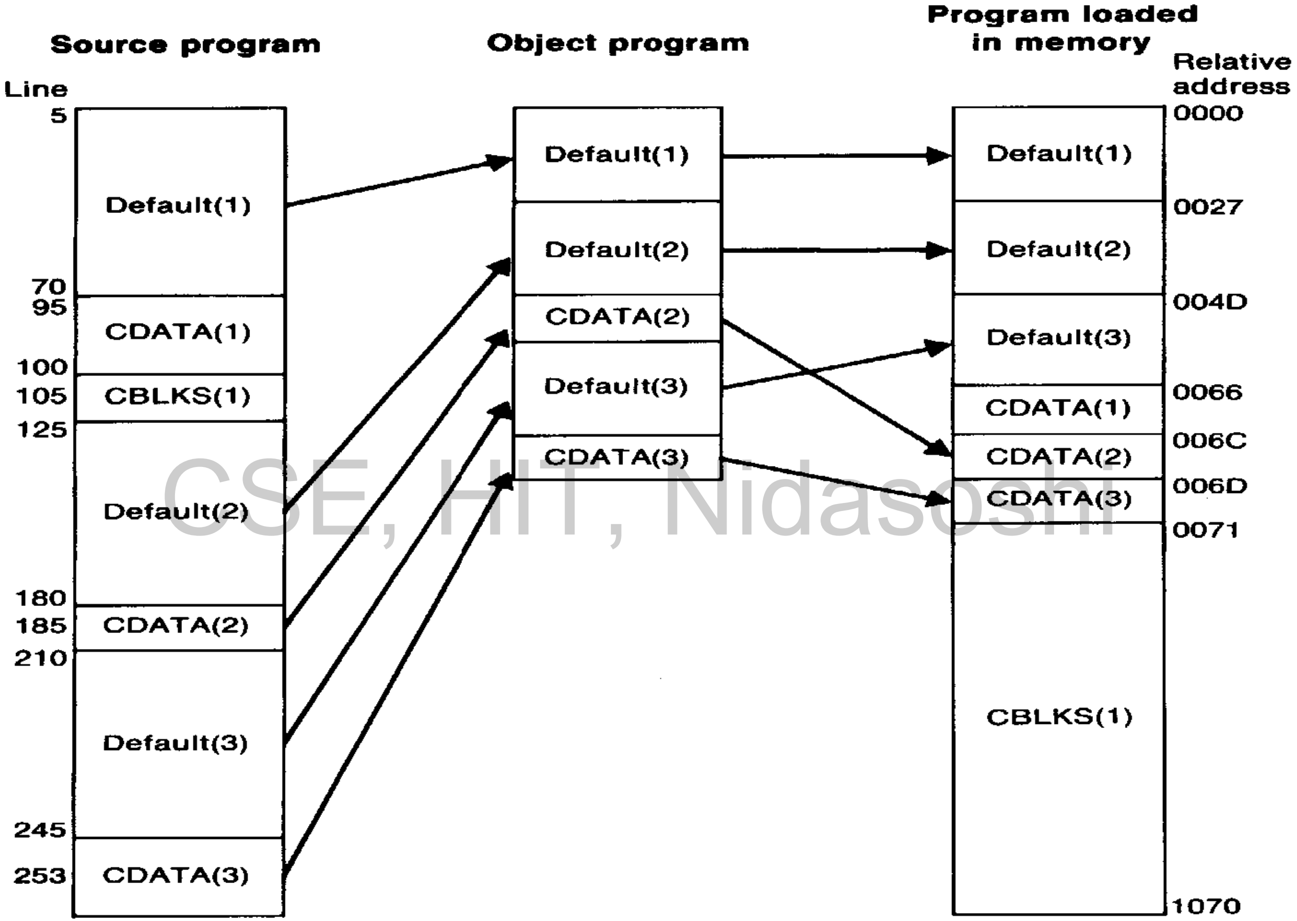


Figure 2.14 Program blocks from Fig. 2.11 traced through the assembly and loading processes.

2.3.5 Control Sections & Program Linking

- **Control section**
  - ❖ Handling of programs that consist of multiple control sections.
  - ❖ A part of the program.
  - ❖ Can be loaded and relocated independently.
  - ❖ Different control sections are most often used for subroutines or other logical subdivisions of a program.
  - ❖ The programmer can assemble, load, and manipulate each of these control sections separately.
  - ❖ Flexibility.
  - ❖ Linking control sections together.

Control Sections & Program Linking

- External references
  - ❖ Instructions in one control section might need to refer to instructions or
  - data located in another section.
- Figure 2.15, multiple control sections.
  - ❖ Three sections, main COPY, RDREC, WRREC.
  - ❖ Assembler directive CSECT.
  - ❖ EXTDEF and EXTREF for external symbols.
  - ❖ The order of symbols is not significant.

```
COPY      START      0
          EXTDEF     BUFFER, BUFEND,
                     LENGTH
          EXTREF     RDREC, WRREC
```

```
Line        Source statement

  5    COPY      START    0              COPY FILE FROM INPUT TO OUTPUT
  6              EXTDEF   BUFFER,BUFEND,LENGTH
  7              EXTREF   RDREC,WRREC
 10    FIRST     STL      RETADR         SAVE RETURN ADDRESS
 15    CLOOP    +JSUB     RDREC          READ INPUT RECORD
 20              LDA      LENGTH         TEST FOR EOF (LENGTH = 0)
 25              COMP    #0
 30              JEQ      ENDFIL         EXIT IF EOF FOUND
 35             +JSUB     WRREC          WRITE OUTPUT RECORD
 40              J        CLOOP          LOOP
 45    ENDFIL    LDA     =C'EOF'         INSERT END OF FILE MARKER
 50              STA      BUFFER
 55              LDA     #3              SET LENGTH = 3
 60              STA      LENGTH
 65             +JSUB     WRREC          WRITE EOF
 70              J       @RETADR         RETURN TO CALLER
 95    RETADR    RESW     1
100    LENGTH    RESW     1              LENGTH OF RECORD
103              LTORG
105    BUFFER    RESB     4096           4096-BYTE BUFFER AREA
106    BUFEND    EQU      *
107    MAXLEN    EQU      BUFEND-BUFFER
```

```
RDREC   CSECT
.
.       SUBROUTINE TO READ RECORD INTO BUFFER
.
        EXTREF  BUFFER,LENGTH,BUFEND
        CLEAR   X                       CLEAR LOOP COUNTER
        CLEAR   A                       CLEAR A TO ZERO
        CLEAR   S                       CLEAR S TO ZERO
        LDT     MAXLEN
RLOOP   TD      INPUT                   TEST INPUT DEVICE
        JEQ     RLOOP                   LOOP UNTIL READY
        RD      INPUT                   READ CHARACTER INTO REGISTER A
        COMPR   A,S                     TEST FOR END OF RECORD (X'00')
        JEQ     EXIT                    EXIT LOOP IF EOR
       +STCH    BUFFER,X                STORE CHARACTER IN BUFFER
        TIXR    T                       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    BUFEND-BUFFER
```

```
WRREC     CSECT
.
.         SUBROUTINE TO WRITE RECORD FROM BUFFER
.
          EXTREF    LENGTH,BUFFER
          CLEAR     X               CLEAR LOOP COUNTER
         +LDT       LENGTH
WLOOP     TD       =X'05'           TEST OUTPUT DEVICE
          JEQ       WLOOP           LOOP UNTIL READY
         +LDCH      BUFFER,X        GET CHARACTER FROM BUFFER
          WD       =X'05'           WRITE CHARACTER
          TIXR      T               LOOP UNTIL ALL CHARACTERS
          JLT       WLOOP             HAVE BEEN WRITTEN
          RSUB                      RETURN TO CALLER
          END       FIRST
```

Figure 2.15 Illustration of control sections and program linking.

Control Sections & Program Linking

- Figure 2.16, the generated object code.

```
15    0003     CLOOP+JSUB  RDREC            4B100000
160  0017              +STCH BUFFER,X       57900000
```

  - RDREC is an external reference.
  - The assembler has no idea where the control section containing RDREC will be loaded, so it cannot assemble the address.
  - The proper address to be inserted at load time.
  - Must use extended format instruction for external reference (M records are needed).

```
190             0028     MAXLEN WORD BUFEND-BUFFER
```

  - An expression involving two external references.

```
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
```

```
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
```

```
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
```

Figure 2.16 Program from Fig. 2.15 with object code.

Control Sections & Program Linking

- The loader will add to this data area with the address of BUFEND and subtract from it the address of BUFFER. (COPY and RDREC)
- Line 190 and 107, in 107, the symbols BUFEND and BUFFER are defined in the same section.
- The assembler must remember in which control section a symbol is defined.
- The assembler allows the same symbol to be used in different control sections, lines 107 and 190.

- Figure 2.17, two new records.
  - Defined record for EXTDEF, relative address.
  - Refer record for EXTREF.

Define record:

Col. 1	D
Col. 2–7	Name of external symbol defined in this control section
Col. 8–13	Relative address of symbol within this control section (hexadecimal)
Col. 14–73	Repeat information in Col. 2–13 for other external symbols

Refer record:

Col. 1	R
Col. 2–7	Name of external symbol referred to in this control section
Col. 8–73	Names of other external reference symbols

The other information needed for program linking is added to the Modification record type. The new format is as follows.

Modification record (revised):

Col. 1	M
Col. 2–7	Starting address of the field to be modified, relative to the beginning of the control section (hexadecimal)
Col. 8–9	Length of the field to be modified, in half-bytes (hexadecimal)
Col. 10	Modification flag (+ or –)
Col. 11–16	External symbol whose value is to be added to or subtracted from the indicated field

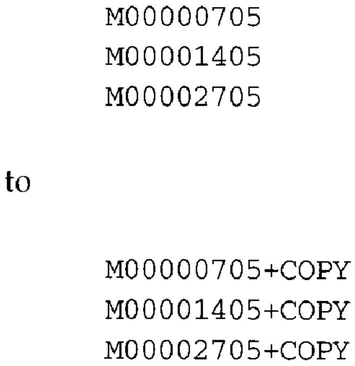
Control Sections & Program Linking

- Modification record
  - M
  - Starting address of the field to be modified, relative to the beginning of the control section (Hex).
  - Length of the field to be modified, in *half-bytes*.
  - Modification flag (+ or -).
  - External symbol.

  M^000004^05+RDREC

  M^000028^06+BUFEND

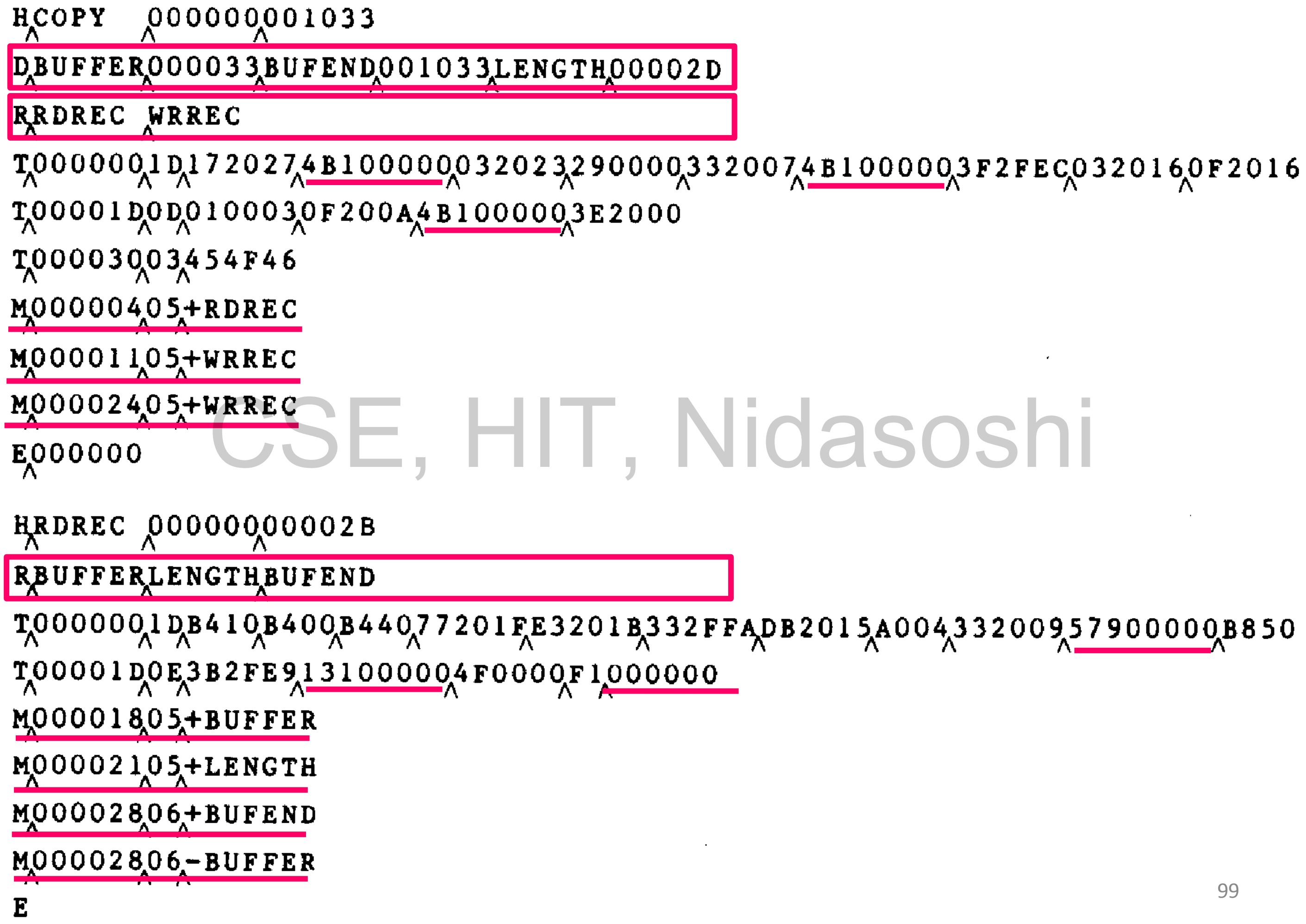
  M^000028^06-BUFFER
- Use Figure 2.8 for program relocation.



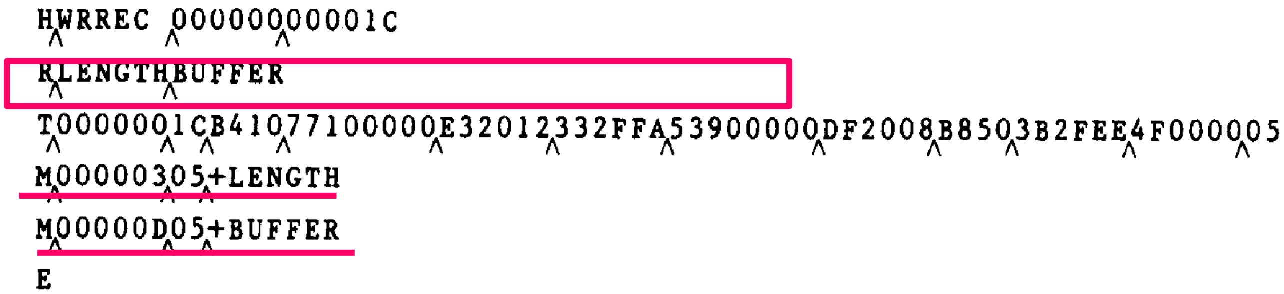
```
M00000705
M00001405
M00002705
```

to

```
M00000705+COPY
M00001405+COPY
M00002705+COPY
```



```
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
```

Figure 2.17 Object program corresponding to Fig. 2.15.

2.4 Assembler Design Options

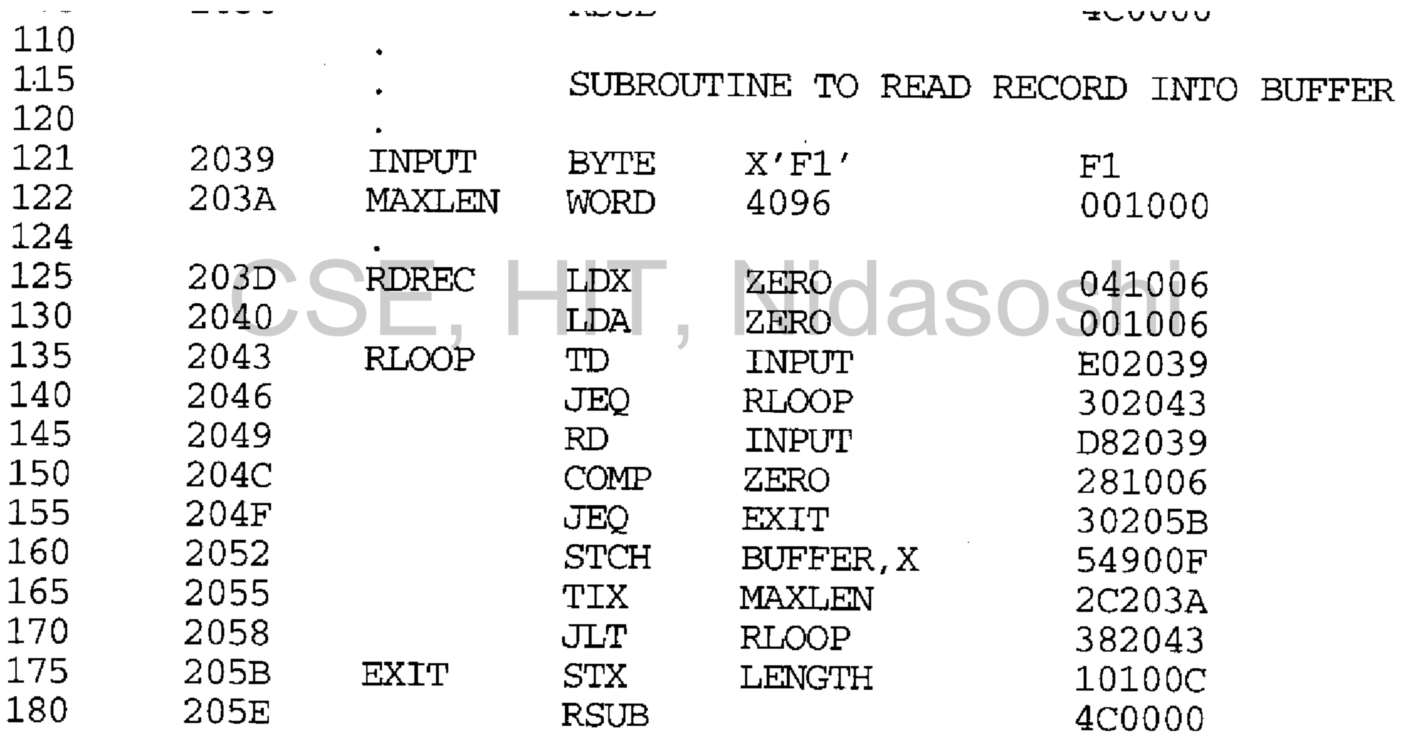
- 2.4.1 Two-Pass Assembler
- Most assemblers
  - ❖Processing the source program into two passes.
  - ❖The internal tables and subroutines that are used only during Pass 1.
  - ❖The SYMTAB, LITTAB, and OPTAB are used by both passes.
- The main problems to assemble a program in one pass involves forward references.

2.4.2 One-Pass Assemblers

- **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 labels.

- **Two types of one-pass assembler. (Fig. 2.18)**
  - ❖One type produces object code directly in memory for immediate execution.
  - ❖The other type produces the usual kind of object program for later execution.

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



```
110               .
115               .         SUBROUTINE TO READ RECORD INTO BUFFER
120               .
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
```

```
195                  .
200                  .          SUBROUTINE TO WRITE RECORD FROM BUFFER
205                  .
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
```

Figure 2.18 Sample program for a one-pass assembler.

One-Pass Assemblers

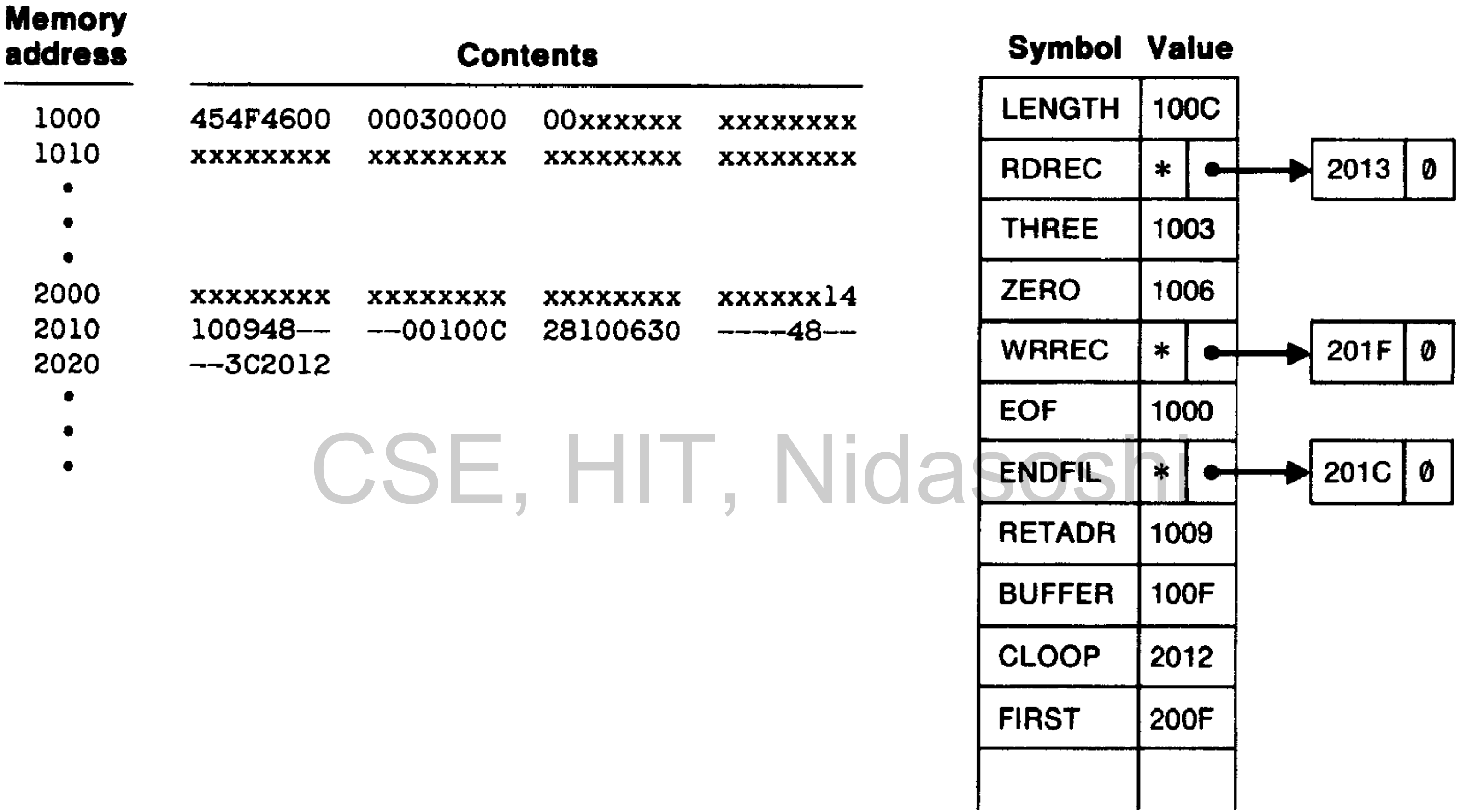
- **Load-and-go one-pass assembler**

  - ❖ The assembler avoids the overhead of writing the object program out and reading it back in.

  - ❖ The object program is produced in memory, the handling of forward references becomes less difficult.

  - ❖ Figure2.19(a), shows the SYMTAB after scanning line 40 of the program in Figure 2.18.

  - ❖ Since RDREC was not yet defined, the instruction was assembled with no value assigned as the operand address (denote by ----).



```
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

Figure 2.19(a) Object code in memory and symbol table entries for the program in Fig. 2.18 after scanning line 40.

One-Pass Assemblers

- **Load-and-go one-pass assembler**
  - ❖RDREC was then entered into SYMTAB as an undefined symbol, the address of the operand field of the instruction (2013) was inserted.
  - ❖Figure 2.19(b), when the symbol ENDFIL was defined (line 45), the assembler placed its value in the SYMTAB entry; it then inserted this value into the instruction operand field (201C).
  - ❖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.

One-Pass Assemblers

- **Another one-pass assembler by generating OP**

- Generate another Text record with correct operand address.
- When the program is loaded, this address will be inserted into the instruction by the action of the loader.
- Figure 2.20, the operand addresses for the instructions on lines 15, 30, and 35 have been generated as 0000.
- When the definition of ENDFIL is encountered on line 45, the third Text record is generated, the value 2024 is to be loaded at location 201C.
- The loader completes forward references.

```
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
```

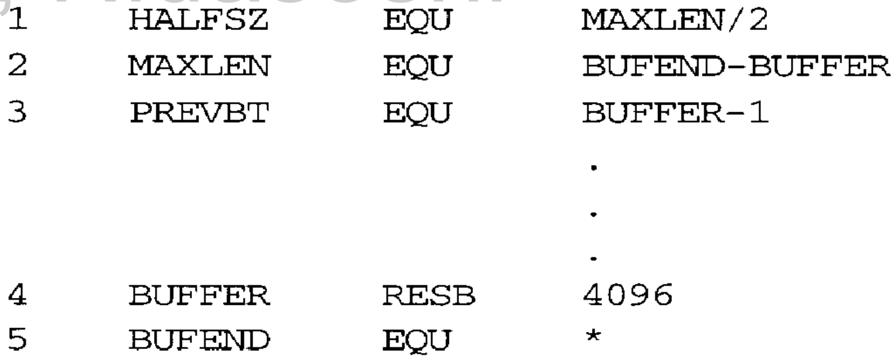
Figure 2.20 Object program from one-pass assembler for program in Fig. 2.18.

2.4.3 Multi-Pass Assemblers

- Use EQU, any symbol used on the RHS be defined previously in the source.

```
ALPHA    EQU     BETA
BETA     EQU     DELTA
DELTA    RESW    1
```

  - **Need 3 passes!**
- Figure 2.21, multi-pass assembler

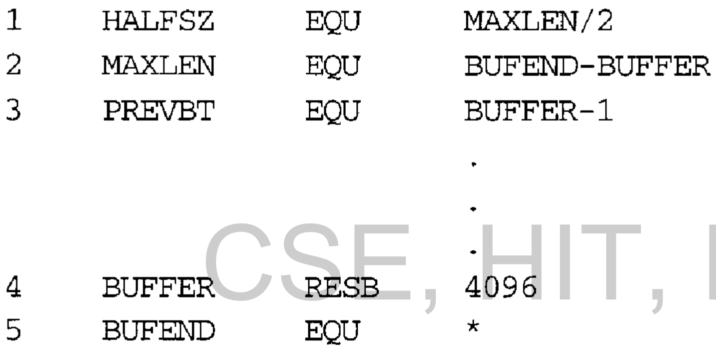


```
1     HALFSZ     EQU     MAXLEN/2
2     MAXLEN     EQU     BUFEND-BUFFER
3     PREVBT     EQU     BUFFER-1
                         .
                         .
                         .
4     BUFFER     RESB    4096
5     BUFEND     EQU     *
```

(a)

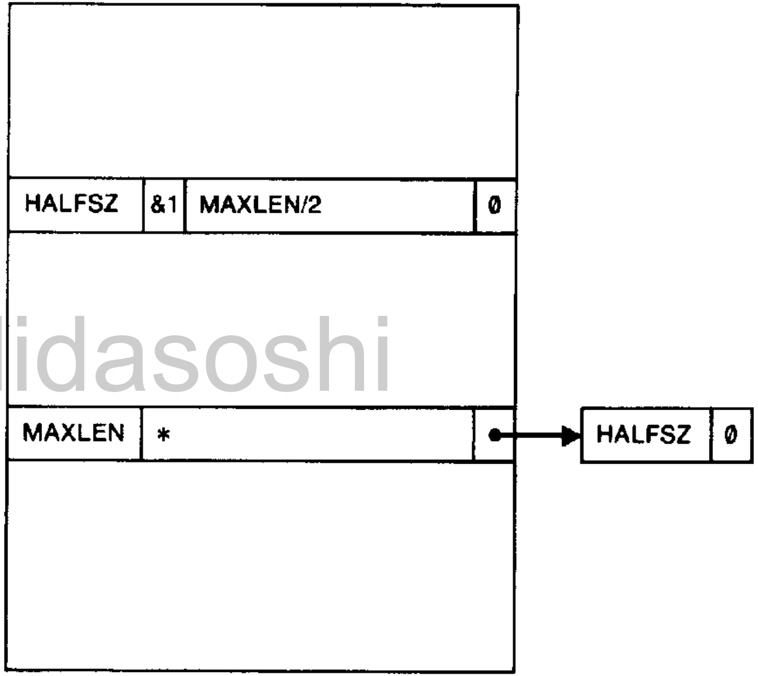
Multi-Pass Assemblers

- **Problem-step1**



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

(a)



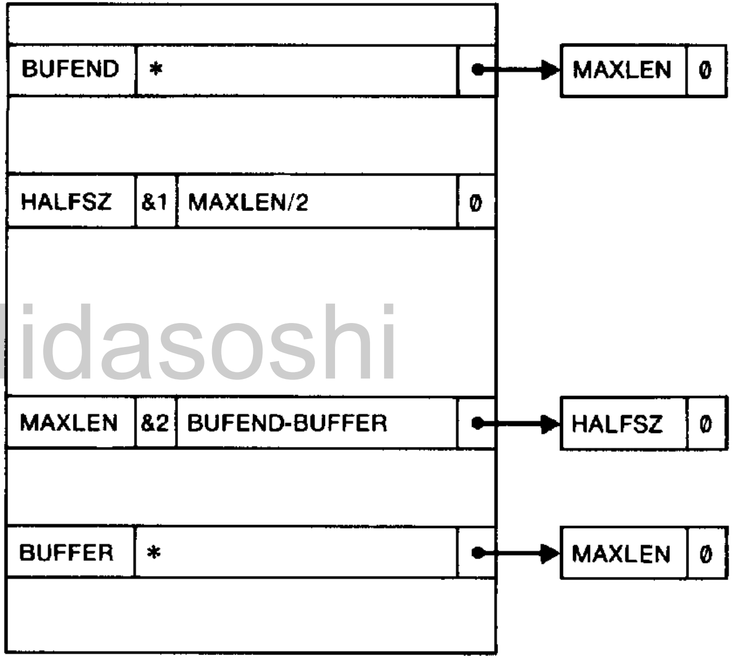
(b)

Multi-Pass Assemblers

- **Problem-step2**

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

(a)



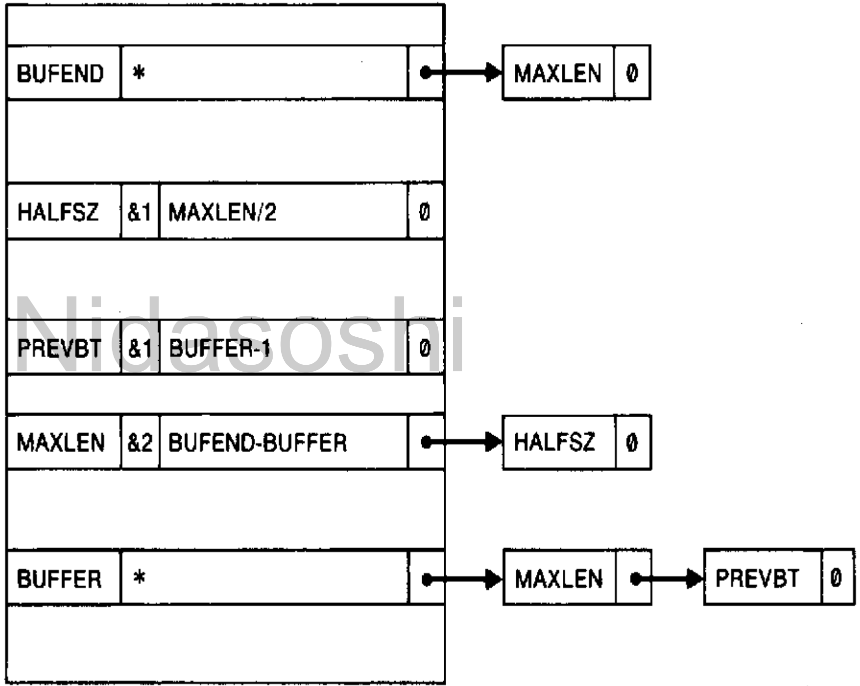
(c)

Multi-Pass Assemblers

- **Problem-step3**

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

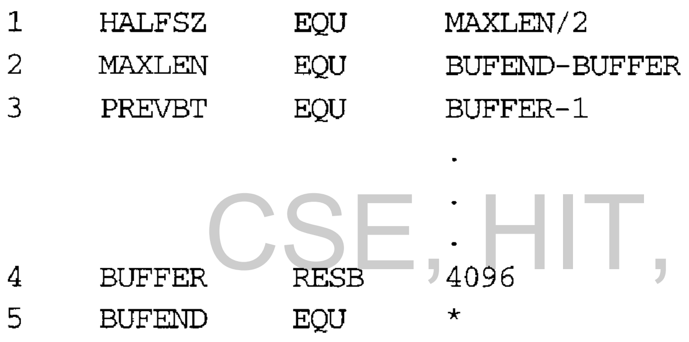
(a)



(d)

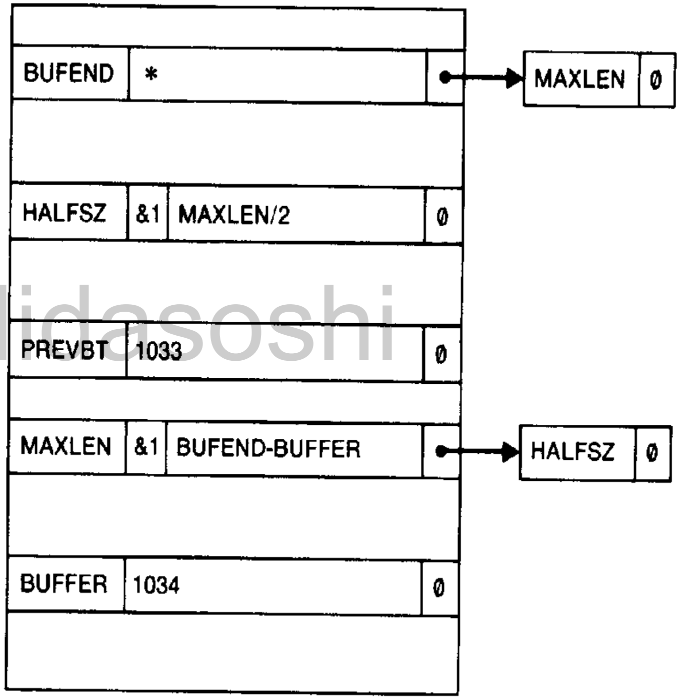
Multi-Pass Assemblers

- **Problem-step4**



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

(a)

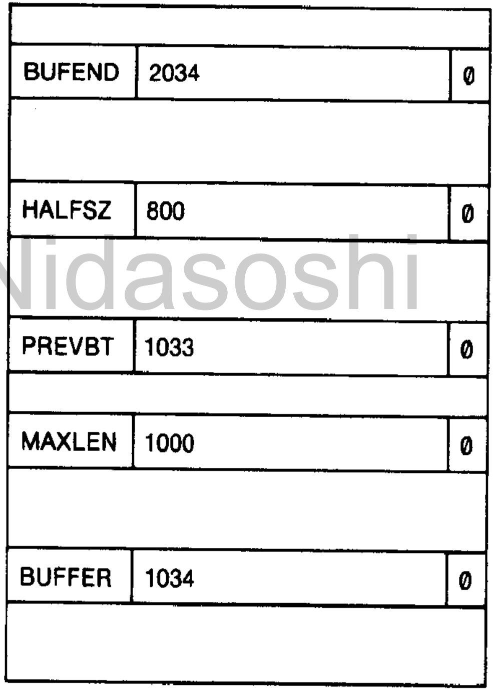


(e)

Multi-Pass Assemblers

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

(a)

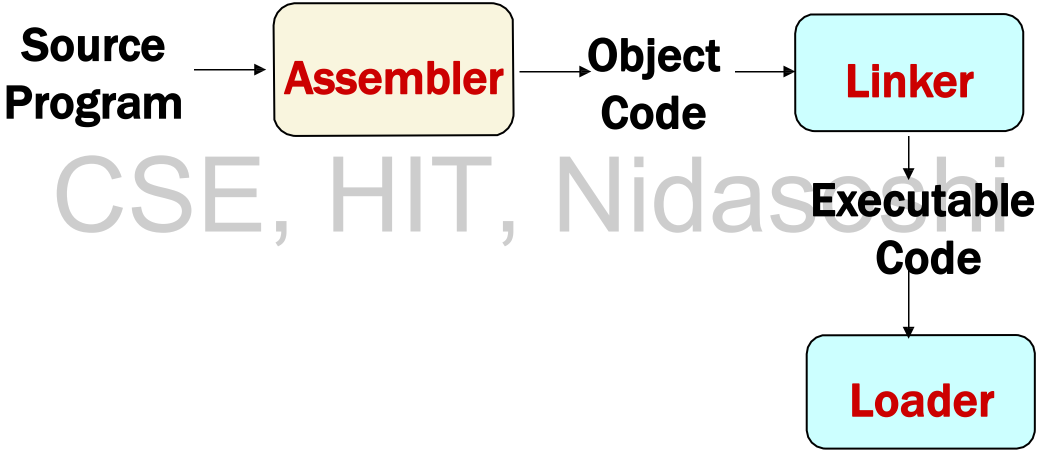


BUFEND	2034	0
HALFSZ	800	0
PREVBT	1033	0
MAXLEN	1000	0
BUFFER	1034	0

(f)

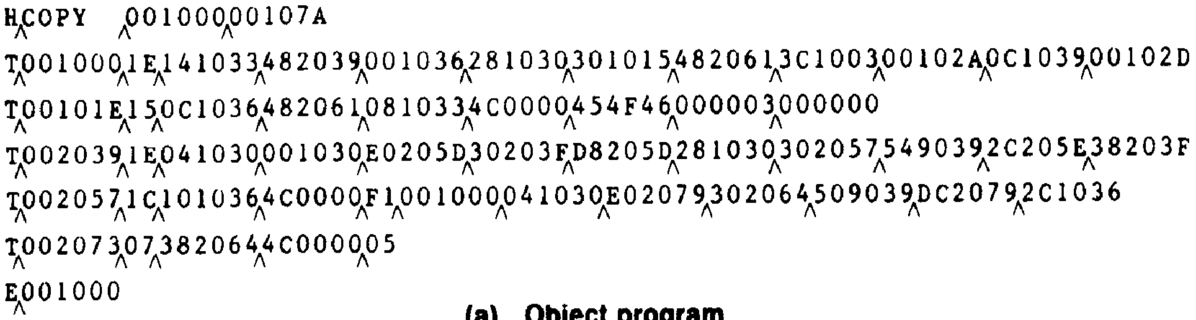
Module-1: Chapter3

Loaders



Design of an Absolute Loader

- Absolute loader, in Figures 3.1 and 3.2.
  - Does not perform linking and program relocation.
  - The contents of memory locations for which there is no Text record are shown as xxxx.
  - Each byte of assembled code is given using its Hex representation in character form.

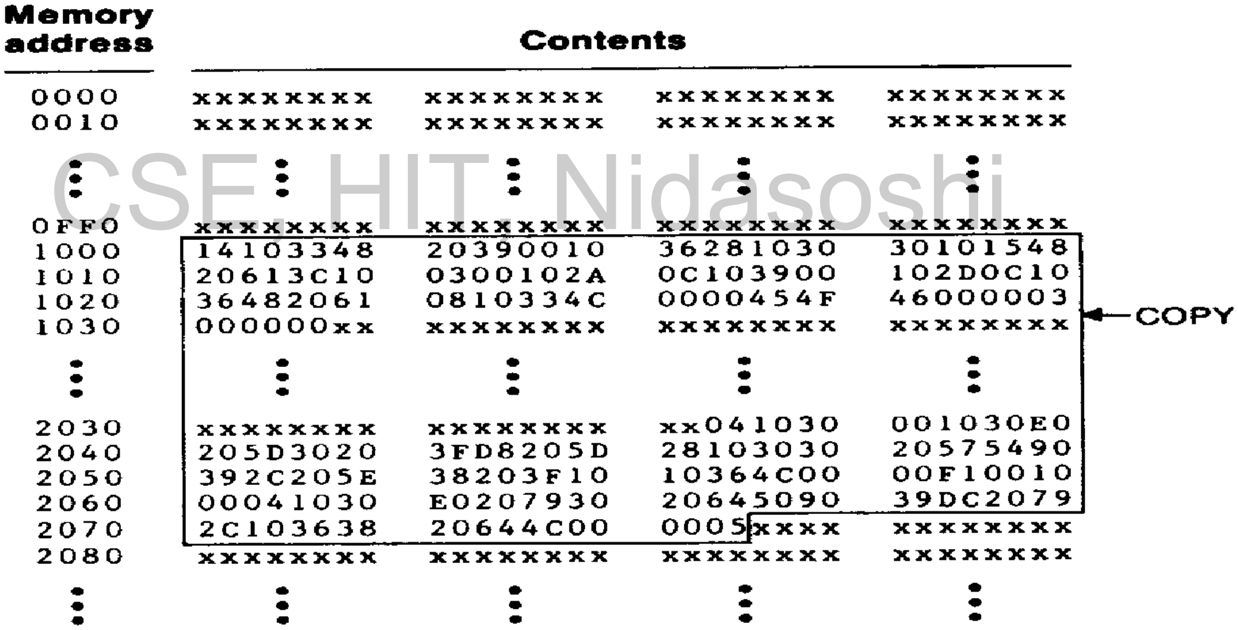


```
H^COPY  ^001000^00107A
T^001000^1E^141033^482039^001036^281030^301015^482061^3C1003^00102A^0C1039^00102D
T^00101E^15^0C1036^482061^081033^4C0000^454F46^000003^000000
T^002039^1E^041030^001030^E0205D^30203F^D8205D^281030^302057^549039^2C205E^38203F
T^002057^1C^101036^4C0000^F1^001000^041030^E02079^302064^509039^DC2079^2C1036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

(a) Object program

```
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
```

(a) Object program

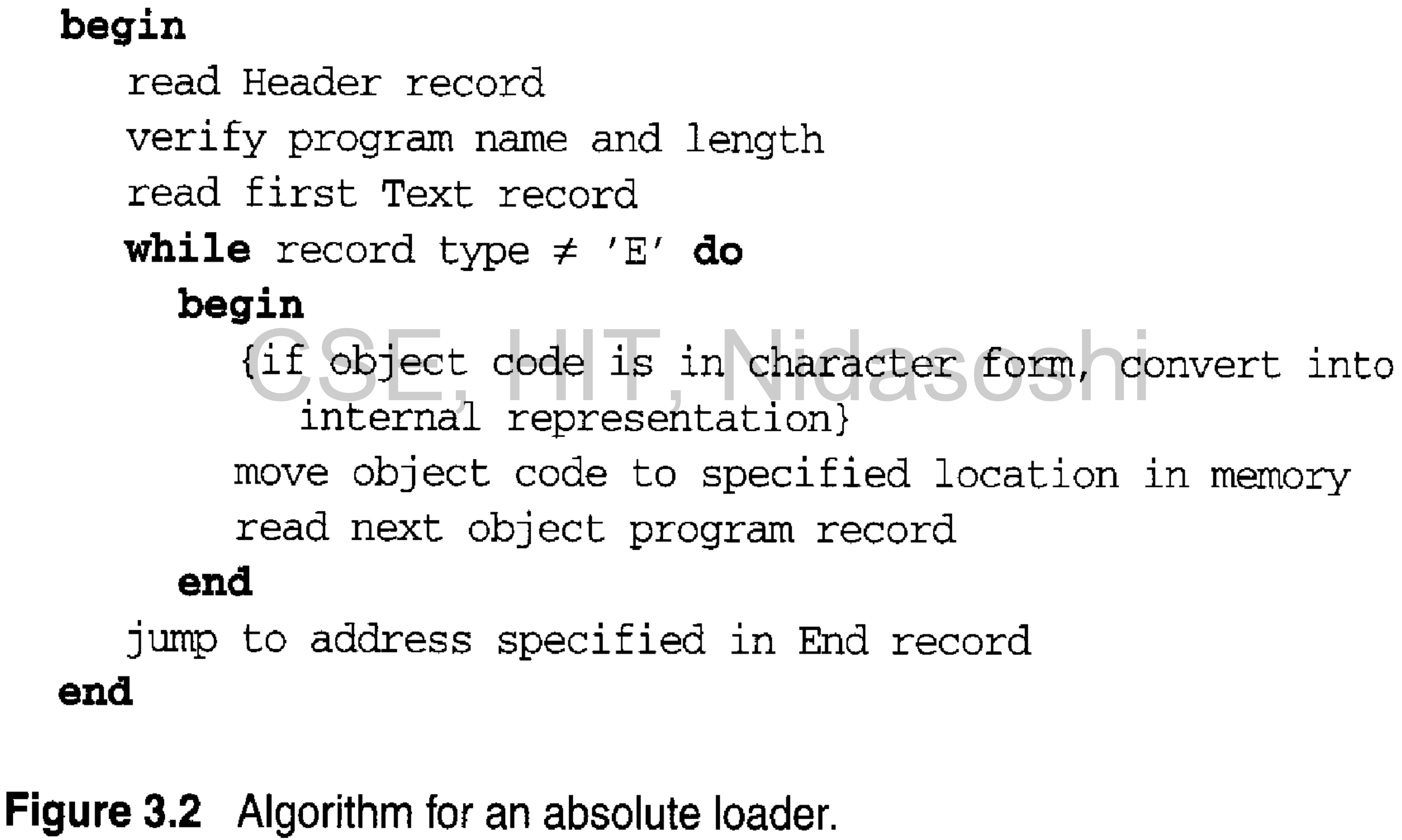


```
Memory
address             Contents

0000      xxxxxxxx  xxxxxxxx  xxxxxxxx  xxxxxxxx
0010      xxxxxxxx  xxxxxxxx  xxxxxxxx  xxxxxxxx
  :          :         :         :         :
0FF0      xxxxxxxx  xxxxxxxx  xxxxxxxx  xxxxxxxx
1000      14103348  20390010  36281030  30101548
1010      20613C10  0300102A  0C103900  102D0C10
1020      36482061  0810334C  0000454F  46000003   ←COPY
1030      000000xx  xxxxxxxx  xxxxxxxx  xxxxxxxx
  :          :         :         :         :
2030      xxxxxxxx  xxxxxxxx  xx041030  001030E0
2040      205D3020  3FD8205D  28103030  20575490
2050      392C205E  38203F10  10364C00  00F10010
2060      00041030  E0207930  20645090  39DC2079
2070      2C103638  20644C00  0005xxxx  xxxxxxxx
2080      xxxxxxxx  xxxxxxxx  xxxxxxxx  xxxxxxxx
  :          :         :         :         :
```

(b) Program loaded in memory

Figure 3.1 Loading of an absolute program.

Algorithm for Absolute loader (IMP)



```
begin
   read Header record
   verify program name and length
   read first Text record
   while record type ≠ 'E' do
      begin
         {if object code is in character form, convert into
            internal representation}
         move object code to specified location in memory
         read next object program record
      end
   jump to address specified in End record
end
```

Figure 3.2 Algorithm for an absolute loader.

A Simple Bootstrap Loader

- **A bootstrap loader, Figure 3.3**.
  - ❖ Loads the first program to be run by the computer--- usually an operating system.
  - ❖ The bootstrap itself begins at address 0 in the memory.
  - ❖ It loads the OS or some other program starting at address 80
  - ❖ Each byte of object code to be loaded is represented on device F1 as two Hex digits (by GETC subroutines).
  - ❖ The ASCII code for the character 0 (Hex 30) is converted to the numeric value 0.
  - ❖ The object code from device F1 is always loaded into consecutive bytes of memory, starting at address 80.

A Simple Bootstrap Loader

- THIS BOOTSTRAP READS OBJECT CODE FROM DEVICE Fl AND ENTERS IT INTO MEMORY STARTING AT ADDRESS 80 (HEXADECIMAL). AFTER ALL OF THE CODE FROM DEVF1 HAS BEEN SEEN ENTERED INTO MEMORY, THE BOOTSTRAP EXECUTES A JUMP TO ADDRESS 80 TO BEGIN EXECUTION OF THE PROGRAM JUST LOADED. REGISTER X CONTAINS THE NEXT ADDRESS TO BE LOADED.

- SUBROUTINE TO READ ONE CHARACTER FROM INPUT DEVICE AND CONVERT IT FROM ASCII CODE TO HEXADECIMAL DIGIT VALUE. THE CONVERTED DIGIT VALUE IS RETURNED IN REGISTER A. WHEN AN END-OF-FILE IS READ, CONTROL IS TRANSFERRED TO THE STARTING ADDRESS (HEX 80).

```
          CLEAR      A        CLEAR REGISTER A TO ZERO
          LDX       #128      INITIALIZE REGISTER X TO HEX 80
LOOP      JSUB       GETC     READ HEX DIGIT FROM PROGRAM BEING LOADED
          RMO        A,S      SAVE IN REGISTER S
          SHIFTL     S,4      MOVE TO HIGH-ORDER 4 BITS OF BYTE
          JSUB       GETC     GET NEXT HEX DIGIT
          ADDR       S,A      COMBINE DIGITS TO FORM ONE BYTE
          STCH       0,X      STORE AT ADDRESS IN REGISTER X
          TIXR       X,X      ADD 1 TO MEMORY ADDRESS BEING LOADED
          J          LOOP     LOOP UNTIL END OF INPUT IS REACHED

GETC      TD         INPUT    TEST INPUT DEVICE
          JEQ        GETC     LOOP UNTIL READY
          RD         INPUT    READ CHARACTER
          COMP      #4        IF CHARACTER IS HEX 04 (END OF FILE),
          JEQ        80           JUMP TO START OF PROGRAM JUST LOADED
          COMP      #48       COMPARE TO HEX 30 (CHARACTER '0')
          JLT        GETC     SKIP CHARACTERS LESS THAN '0'
          SUB       #48       SUBTRACT HEX 30 FROM ASCII CODE
          COMP      #10       IF RESULT IS LESS THAN 10, CONVERSION IS
          JLT        RETURN      COMPLETE. OTHERWISE, SUBTRACT 7 MORE
          SUB       #7           (FOR HEX DIGITS 'A' THROUGH 'F')
RETURN    RSUB                RETURN TO CALLER
INPUT     BYTE       X'F1'    CODE FOR INPUT DEVICE
          END        LOOP
```