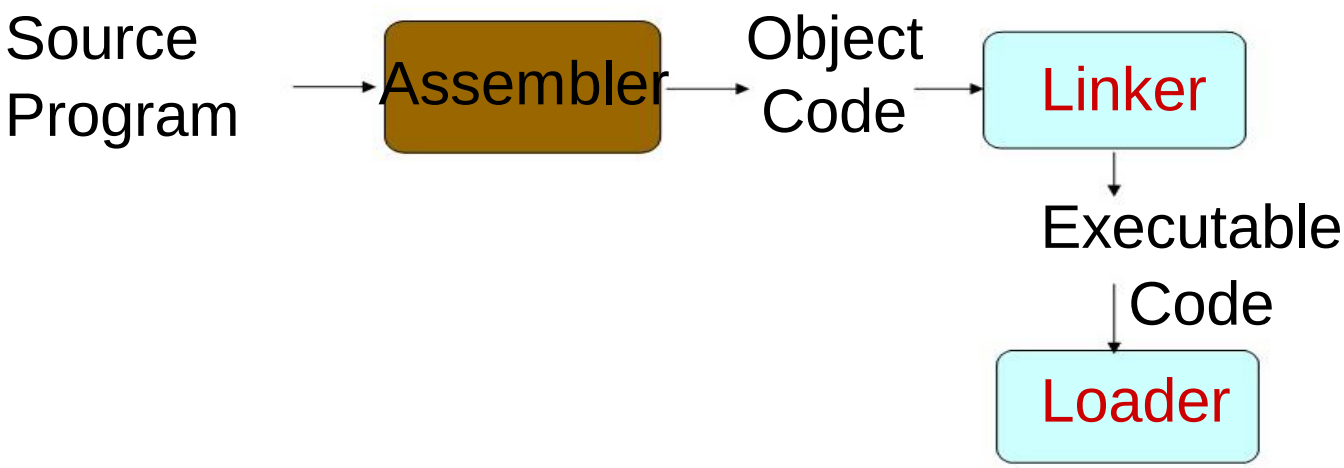
SWE4001 – System Programming
Module 3: Assembler
Lesson 1 of 9: Introduction to Assembler

Assemblers


Source Program
Assembler
Object Code
Linker
Executable Code
Loader

Outline

- Basic Assembler Functions
  - A simple SIC assembler
  - Assembler tables and logic
- Machine-Dependent Assembler Features
  - Instruction formats and addressing modes
  - Program relocation
- Machine-Independent Assembler Features
- Assembler Design Options
  - Two-pass
  - One-pass
  - Multi-pass

2.1 Basic Assembler Functions

- Figure 2.1 shows an assembler language program for SIC.
  - The line numbers are for reference only.
  - Indexing addressing is indicated by adding the modifier “,X”
  - Lines beginning with “.” contain comments only.
  - Reads records from input device (code F1)
  - Copies them to output device (code 05)
  - At the end of the file, writes EOF on the output device, then RSUB to the operating system

Line	Source statement			
5	COPY	START	1000	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	ZERO	
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	THREE	SET LENGTH = 3
60		STA	LENGTH	
65		JSUB	WRREC	WRITE EOF
70		LDL	RETADR	GET RETURN ADDRESS
75		RSUB		RETURN TO CALLER
80	EOF	BYTE	C'EOF'	
85	THREE	WORD	3	
90	ZERO	WORD	0	
95	RETADR	RESW	1	
100	LENGTH	RESW	1	LENGTH OF RECORD
105	BUFFER	RESB	4096	4096-BYTE BUFFER AREA
110				

```
110         .
115         .       SUBROUTINE TO READ RECORD INTO BUFFER
120         .
125     RDREC   LDX     ZERO        CLEAR LOOP COUNTER
130             LDA     ZERO        CLEAR A TO ZERO
135     RLOOP   TD      INPUT       TEST INPUT DEVICE
140             JEQ     RLOOP       LOOP UNTIL READY
145             RD      INPUT       READ CHARACTER INTO REGISTER A
150             COMP    ZERO        TEST FOR END OF RECORD (X'00')
155             JEQ     EXIT        EXIT LOOP IF EOR
160             STCH    BUFFER,X    STORE CHARACTER IN BUFFER
165             TIX     MAXLEN      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
190     MAXLEN  WORD    4096
```

```
200         .       SUBROUTINE TO WRITE RECORD FROM BUFFER
205         .
210         WRREC   LDX     ZERO        CLEAR LOOP COUNTER
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                 TIX     LENGTH      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 2.1 Example of a SIC assembler language program.

2.1 Basic Assembler Functions

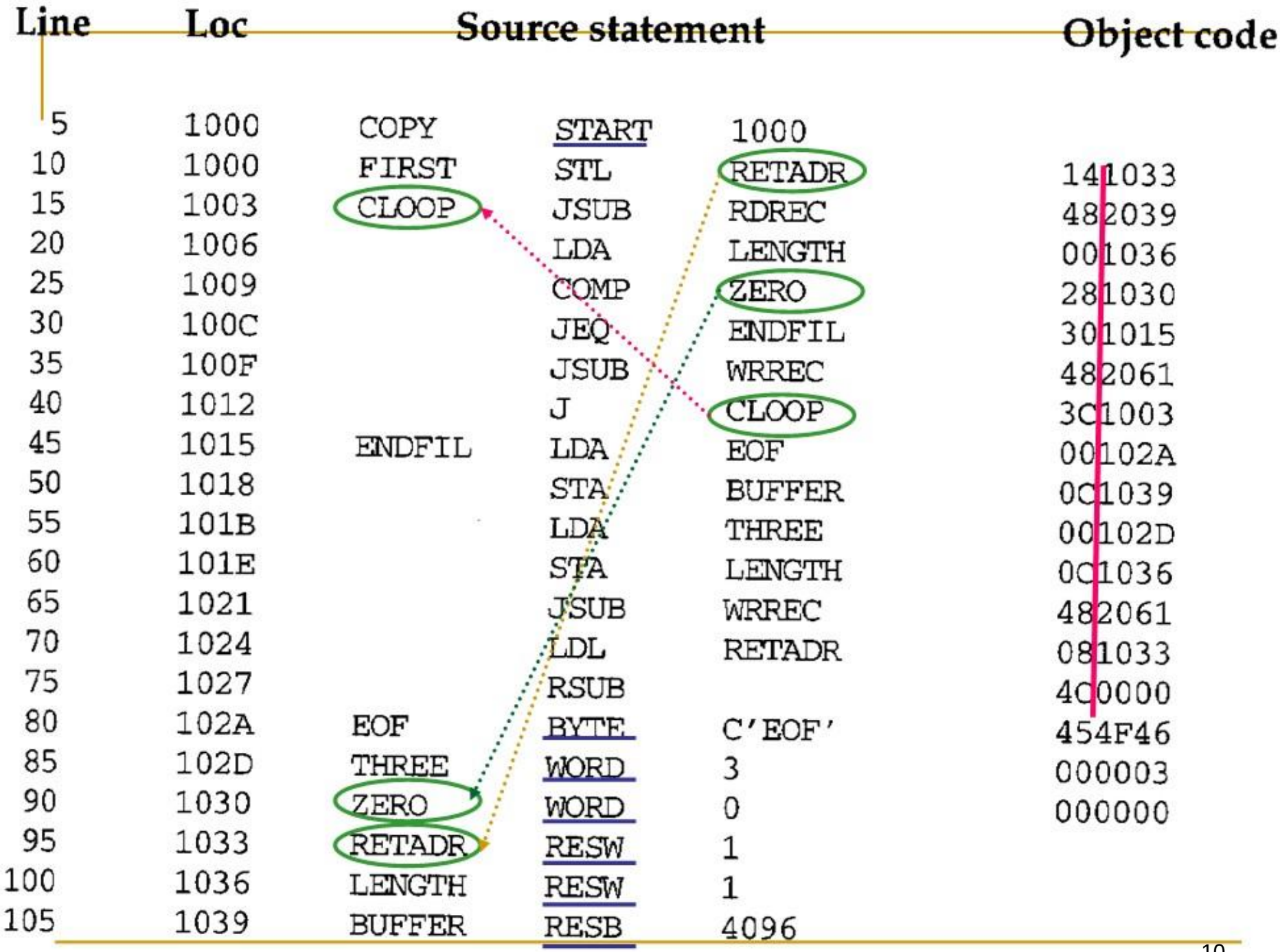
- Assembler *directives* (pseudo-instructions)
  - START, END, BYTE, WORD, RESB, RESW.
  - These statements are not translated into machine instructions.
  - Instead, they provide instructions to the assembler itself.

2.1 Basic Assembler Functions

- 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
  - When the end of file is detected, the program writes EOF on the output device and terminates by RSUB.
- Subroutines (JSUB, RSUB)
  - RDREC, WRREC
  - Save link (L) register first before nested jump

2.1.1 A simple SIC Assembler

- 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.

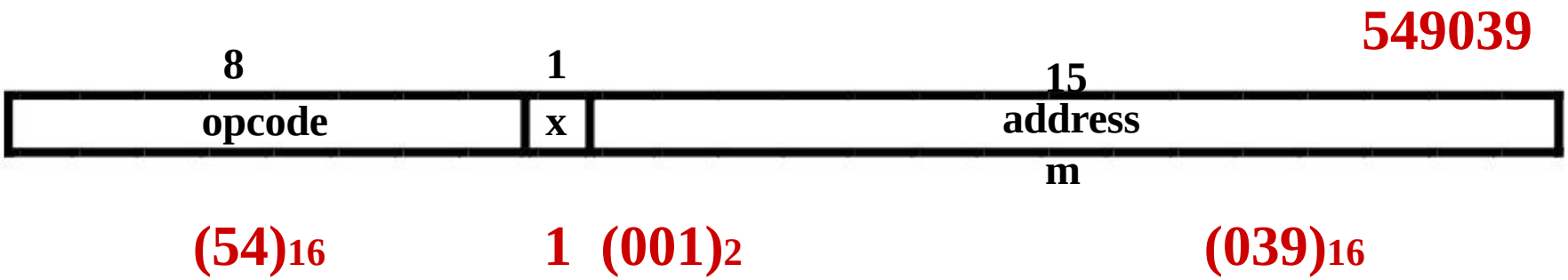
2.1.1 A simple SIC Assembler

- 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 source program.
  - Pass 1 scans the source for label definitions and assigns address (Loc).
  - Pass 2 performs most of the actual translation.

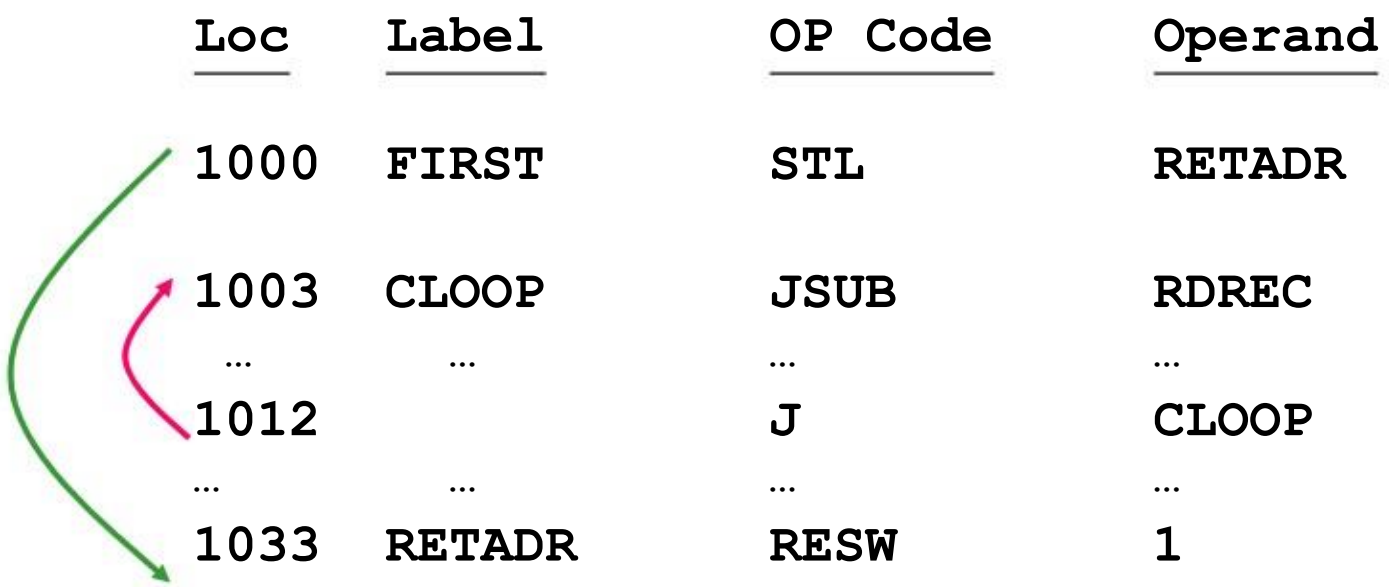
2.1.1 A simple SIC Assembler

- Example of Instruction Assemble
  - Forward reference
  - STCH BUFFER, X



2.1.1 A simple SIC Assembler

- Forward reference
  - Reference to a label that is defined later in the program.



Loc	Label	OP Code	Operand
1000	FIRST	STL	RETADR
1003	CLOOP	JSUB	RDREC
…	…	…	…
1012		J	CLOOP
…	…	…	…
1033	RETADR	RESW	1

2.1.1 A simple SIC Assembler

- The object program (OP) will be loaded into memory for execution.
- Three types of records
  - Header: program name, starting address, length.
  - Text: starting address, length, object code.
  - End: address of first executable instruction.

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)

2.1.1 A simple SIC Assembler

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)

(69-10+1)/10 = 10 instructions

End record:

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

2.1.1 A simple SIC Assembler

- The symbol ^ is used to separate fields.
  - Figure 2.3

    1E(H)=30(D)=16(D)+14(D)

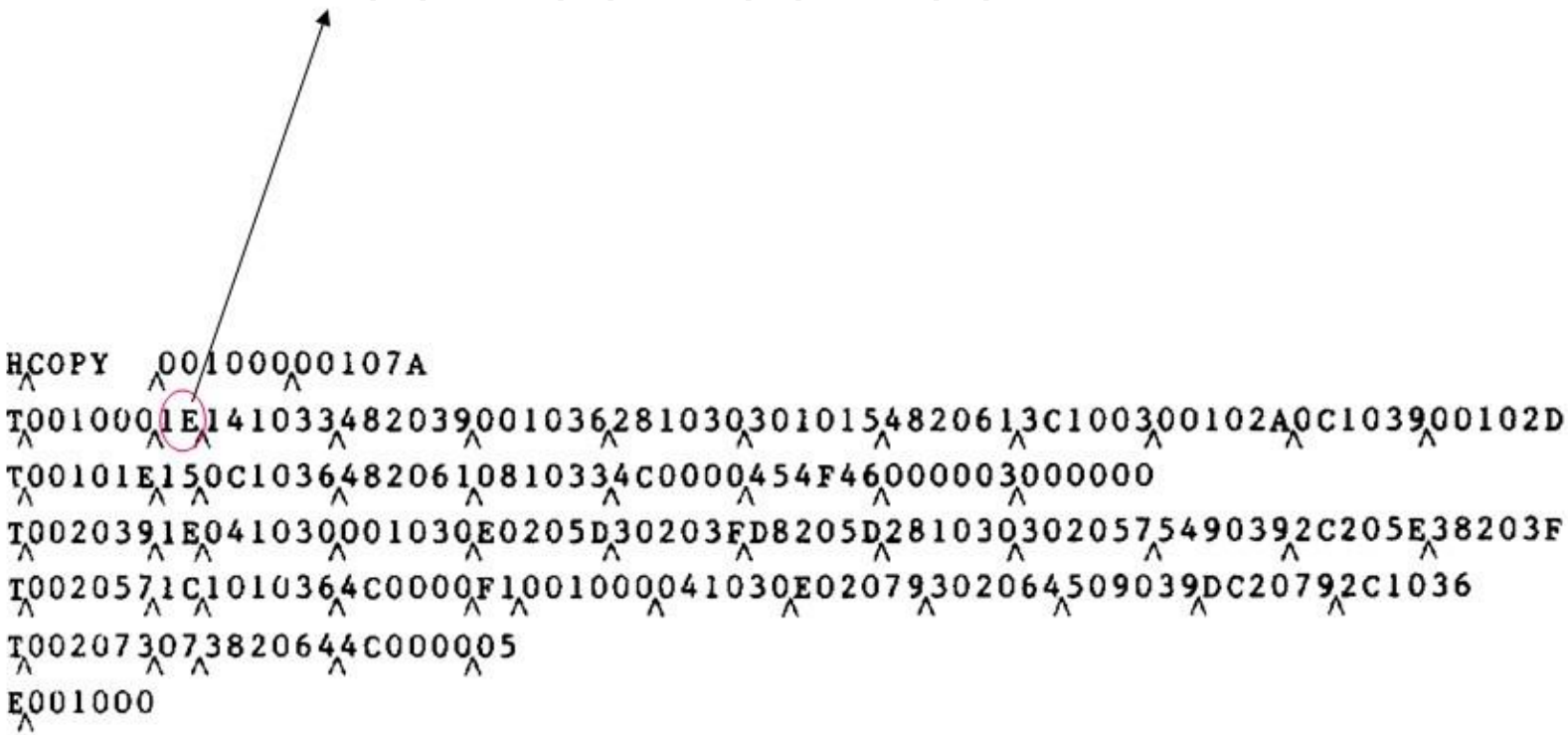


Figure 2.3 Object program corresponding to Fig. 2.2.

2.1.1 A simple SIC Assembler

- Assembler's Functions
  - Convert mnemonic operation codes to their machine language equivalents
    - STL to 14
  - Convert symbolic operands (referred label) to their equivalent machine addresses
    - RETADR to 1033
  - Build the machine instructions in the proper format
  - Convert the data constants to internal machine representations
  - Write the object program and the assembly listing

2.1.1 A simple SIC Assembler

- The functions of the two passes assembler.
- Pass 1 (define symbol)
  - Assign addresses to all statements (generate LOC).
  - Check the correctness of Instruction (check with OP table).
  - Save the values (address) assigned to all labels into SYMBOL table for Pass 2.
  - Perform some processing of assembler directives.

2.1.1 A simple SIC Assembler

- The functions of the two passes assembler.

- Pass 2
  - Assemble instructions (op code from OP table, address from SYMBOL table).
  - Generate data values defined by BYTE, WORD.
  - Perform processing of assembler directives not done during Pass 1.
  - Write the OP (Fig. 2.3) and the assembly listing (Fig. 2.2).

2.1.2 Assembler Tables and Logic

- Our simple assembler uses two internal tables: The OPTAB and SYMTAB.
  - OPTAB is used to look up mnemonic operation codes and translate them to their machine language equivalents.
    - LDA→00, STL→14, …
  - SYMTAB is used to store values (addresses) assigned to labels.
    - COPY→1000, FIRST→1000 …
- Location Counter LOCCTR
  - LOCCTR is a variable for assignment addresses.
  - LOCCTR is initialized to address specified in START.
  - When reach a label, the current value of LOCCTR gives the address to be associated with that label.

2.1.2 Assembler Tables and Logic

- The Operation Code Table (OPTAB)
  - Contain the mnemonic operation & its machine language equivalents (at least).
  - Contain instruction format & length.
  - Pass 1, OPTAB is used to look up and validate operation codes.
  - Pass 2, OPTAB is used to translate the operation codes to machine language.
  - In SIC/XE, assembler search OPTAB in Pass 1 to find the instruction length for incrementing LOCCTR.
  - Organize as a hash table (static table).

2.1.2 Assembler Tables and Logic

- The Symbol Table (SYMTAB)
  - Include the name and value (address) for each label.
  - Include flags to indicate error conditions
  - Contain type, length.
  - Pass 1, labels are entered into SYMTAB, along with assigned addresses (from LOCCTR).
  - Pass 2, symbols used as operands are look up in SYMTAB to obtain the addresses.
  - Organize as a hash table (static table).
  - The entries are rarely deleted from table.

```
COPY   1000
FIRST  1000
CLOOP  1003
ENDFIL 1015
EOF    1024
THREE  102D
ZERO   1030
RETADR 1033
LENGTH 1036
BUFFER 1039
RDREC  2039
```

2.1.2 Assembler Tables and Logic

- Pass 1 usually writes an intermediate file.
  - Contain source statement together with its assigned address, error indicators.
  - This file is used as input to Pass 2.
- Figure 2.4 shows the two passes of assembler.
  - Format with fields LABEL, OPCODE, and OPERAND.
  - Denote numeric value with the prefix #.

    #[OPERAND]

Pass 1

Pass 1:

```
begin
   read first input line
   if OPCODE = 'START' then
      begin
         save #[OPERAND] as starting address
         initialize LOCCTR to starting address
         write line to intermediate file
         read next input line
      end {if START}
   else
      initialize LOCCTR to 0
```

```
   write last line to intermediate file
   save (LOCCTR - starting address) as program length
end {Pass 1}
```

```
while OPCODE ≠ 'END' do
    begin
        if this is not a comment line then
            begin
                if there is a symbol in the LABEL field then
                    begin
                        search SYMTAB for LABEL
                        if found then
                            set error flag (duplicate symbol)
                        else
                            insert (LABEL,LOCCTR) into SYMTAB
                    end {if symbol}
                search OPTAB for OPCODE
                if found then
                    add 3 {instruction length} to LOCCTR
                else if OPCODE = 'WORD' then
                    add 3 to LOCCTR
                else if OPCODE = 'RESW' then
                    add 3 * #[OPERAND] to LOCCTR
                else if OPCODE = 'RESB' then
                    add #[OPERAND] to LOCCTR
                else if OPCODE  = 'BYTE' then
                    begin
                        find length of constant in bytes
                        add length to LOCCTR
                    end {if BYTE}
                else
                    set error flag (invalid operation code)
            end {if not a comment}
        write line to intermediate file
        read next input line
    end {while not END}
```

Pass 2

```
begin
   read first input line {from intermediate file}
   if OPCODE = 'START' then
       begin
           write listing line
           read next input line
       end {if START}
   write Header record to object program
   initialize first Text record
```

```
   write last Text record to object program
   write End record to object program
   write last listing line
end {Pass 2}
```

```
while OPCODE ≠ 'END' do
    begin
        if this is not a comment line then
            begin
                search OPTAB for OPCODE
                if found then
                    begin
                        if there is a symbol in OPERAND field then
                            begin
                                search SYMTAB for OPERAND
                                if found then
                                    store symbol value as operand address
                                else
                                    begin
                                        store 0 as operand address
                                        set error flag (undefined symbol)
                                    end
                            end {if symbol}
                        else
                            store 0 as operand address
                        assemble the object code instruction
                    end {if opcode found}
                else if OPCODE = 'BYTE' or 'WORD' then
                    convert constant to object code
                if object code will not fit into the current Text record then
                    begin
                        write Text record to object program
                        initialize new Text record
                    end
                add object code to Text record
            end {if not comment}
        write listing line
        read next input line
    end {while not END}
```