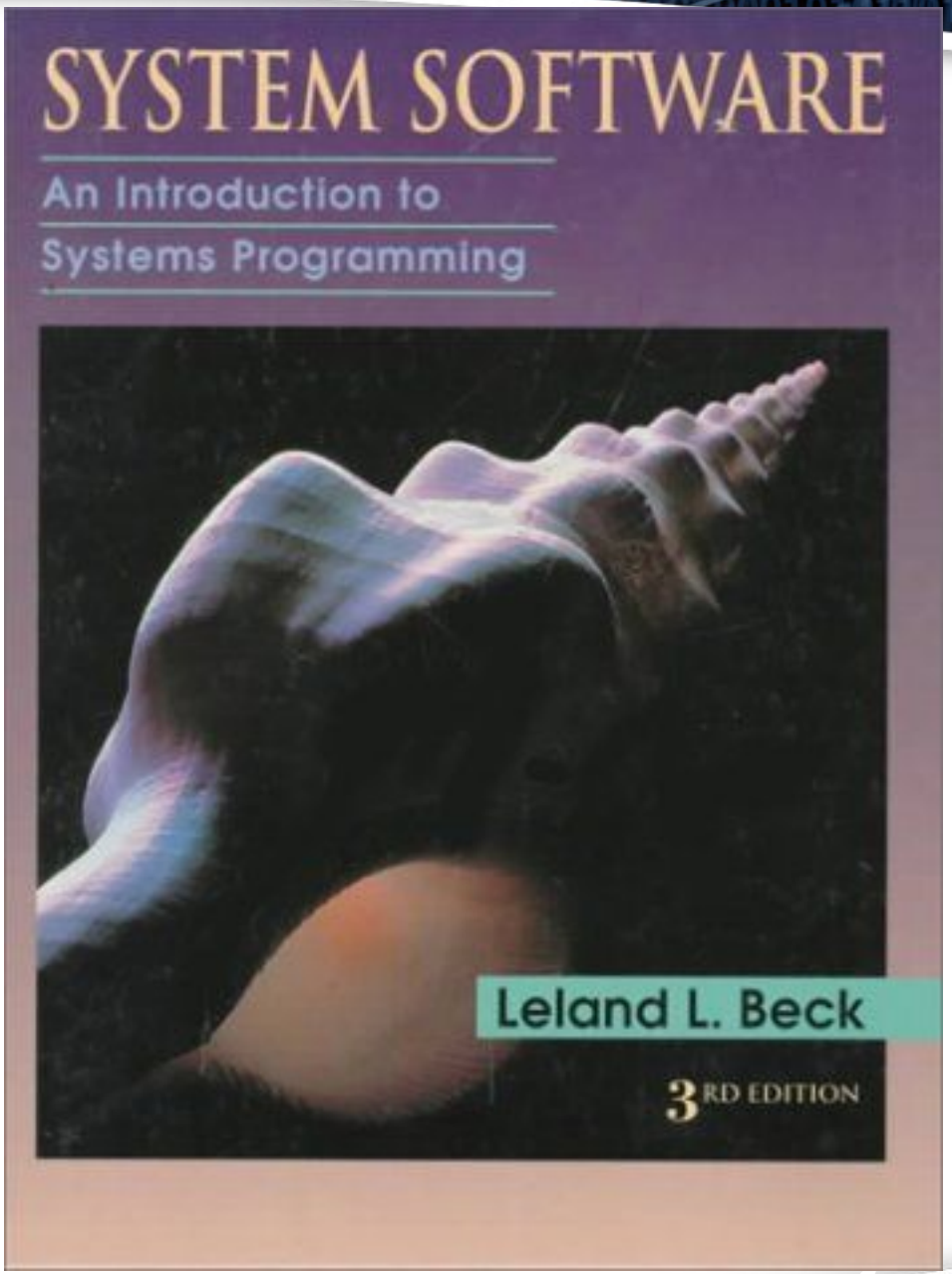
System Programming

Textbook

System software: an introduction to systems programming, Leland L. Beck, 3rd Edition, Addison Wesley, Longman Inc., 1997.



Chapter 1
Background

1.1 Introduction

- What is a System Software ?
  - System Software consists of a variety of programs that support the operation of a computer.
  - The programs implemented in either software and (or) firmware that makes the computer hardware usable.
  - The system software makes it possible for the users to focus on an application or other problem to be solved, without needing to know the details of how the machine works internally.

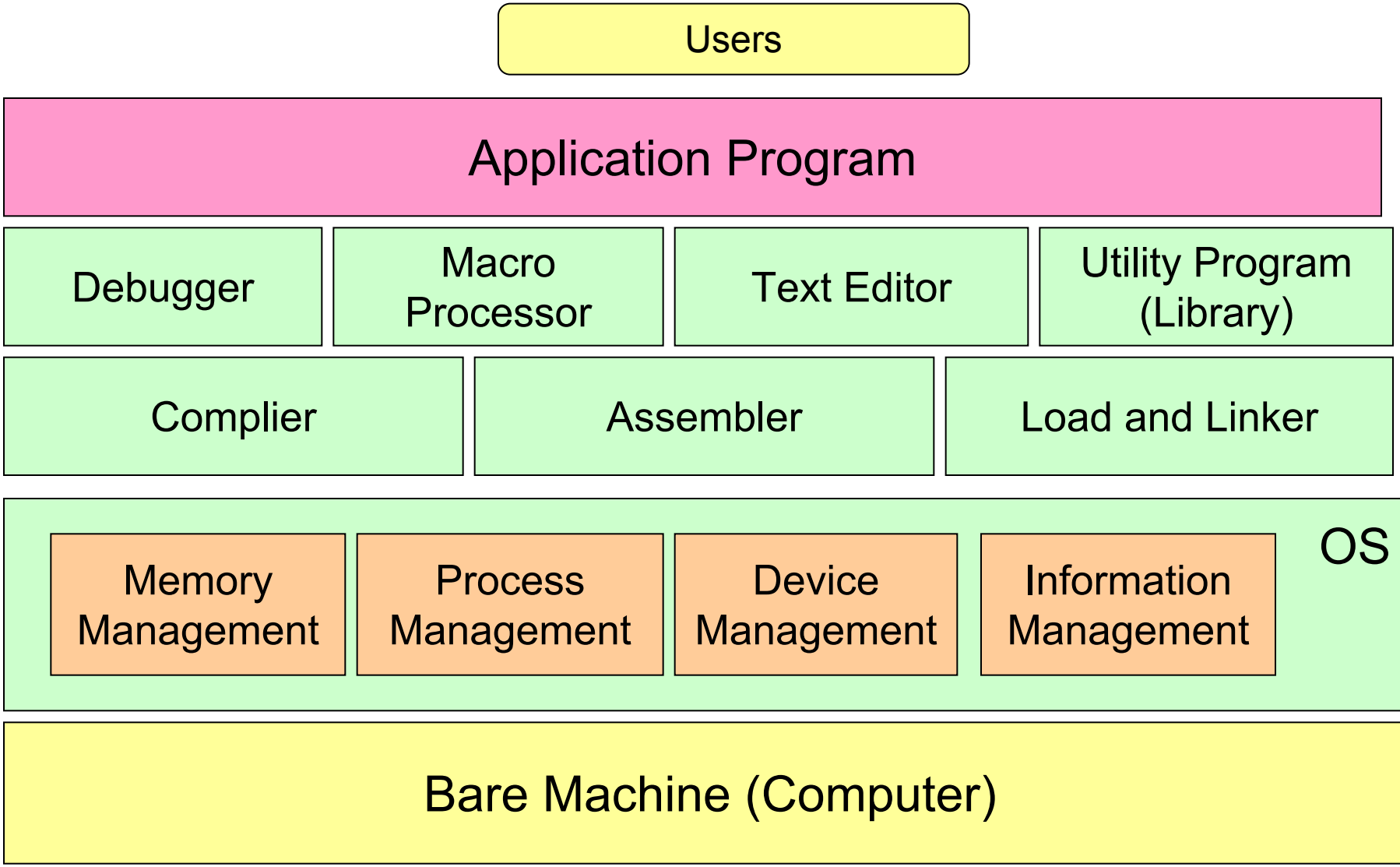
1.2 System Software and Machine Architecture

- System Software vs Application
  - One characteristic in which most system software differs from application software is machine dependency.
  - System programs are intended to support the operation and use of the computer itself, rather than any particular application.
- Examples of system software
  - Text editor, assembler, compiler, loader or linker, debugger, macro processors, operating system, database management systems, software engineering tools, …

1.2 System Software and Machine Architecture

- Text editor
  - To create and modify the program
- Compiler and assembler
  - translate high level programs into machine language
- Loader or linker
  - The resulting machine program was loaded into memory and prepared for execution
- Debugger
  - To help detect errors in the program

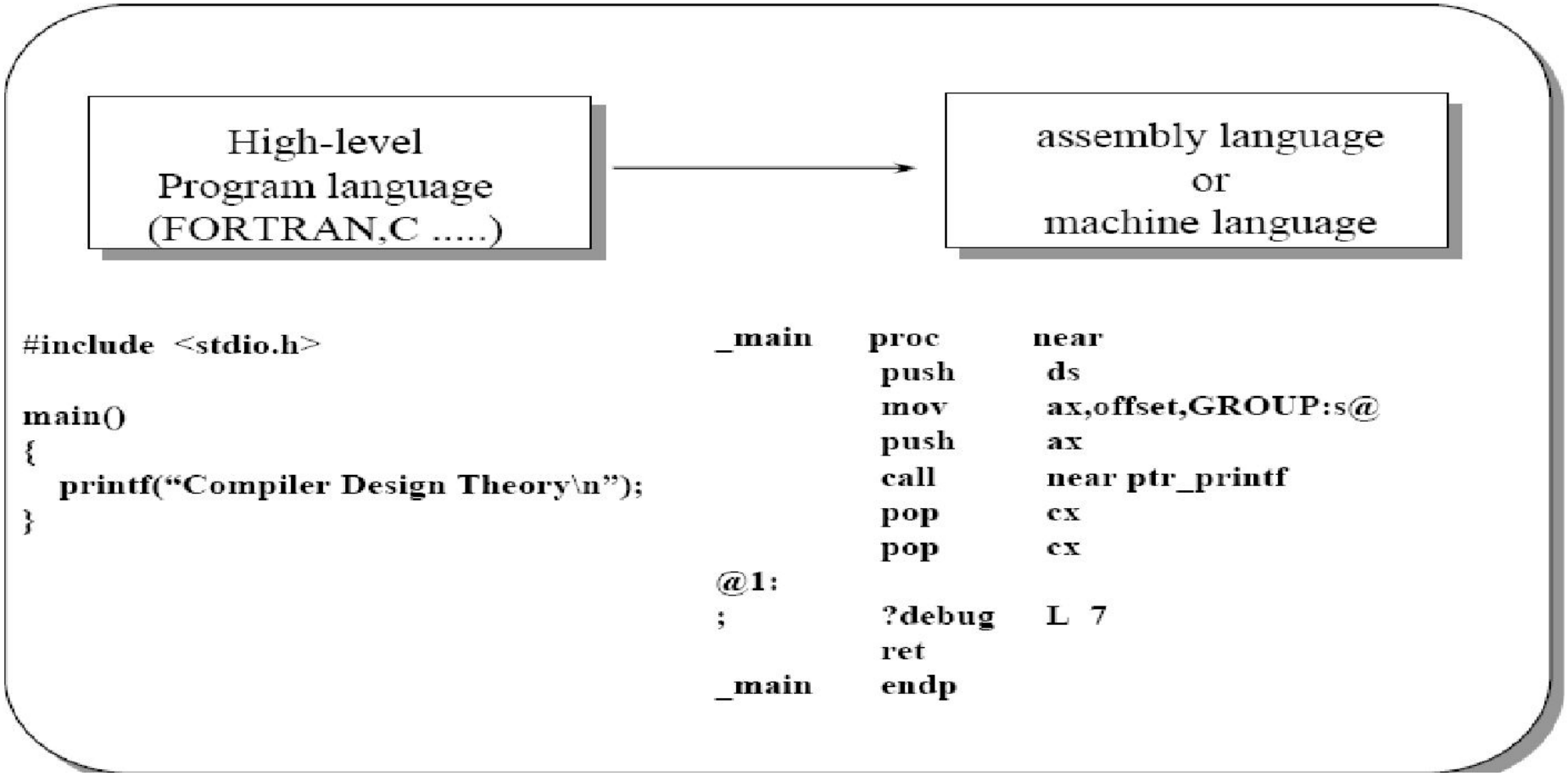
System Software Concept



- **Compiler:** is a program whose task is to accept as input a source program written in a certain high-level language and to produce as output an object code or assembly code.
- **Interpreter:** is a program that ultimately performs the same function as a compiler, but in a different manner. It works by scanning through the source program instruction by instruction. As each instruction is encountered, the interpreter translates it into machine code and executes it directly.
- **Assembler:** is a program that automatically translates the source program written in assembly language and to produce as output an object code written in binary machine code.
- **Linker:** is a program that takes one or more object codes generated by compilers and assembles them into a single executable program.
- **Loader:** (is a routine that) loads an object program into memory of the processor and prepares it for execution

Compiler

- A program whose task is to accept as input a source program written in a certain high-level language and to produce as output an object code which can be run on the computer




```
#include <stdio.h>

main()
{
   printf("Compiler Design Theory\n");
}
```

```
_main   proc    near
        push    ds
        mov     ax,offset,GROUP:s@
        push    ax
        call    near ptr_printf
        pop     cx
        pop     cx
@1:
;       ?debug  L 7
        ret
_main   endp
```

Assembler

- A program that automatically translates the source program written in assembly language and to produce as output an object code written in binary machine code.

Interpreter

- A program that ultimately performs the same function as a compiler, but in a different manner. It works by scanning through the source program instruction by instruction. As each instruction is encountered, the interpreter translates it into m/c code and executes it directly.

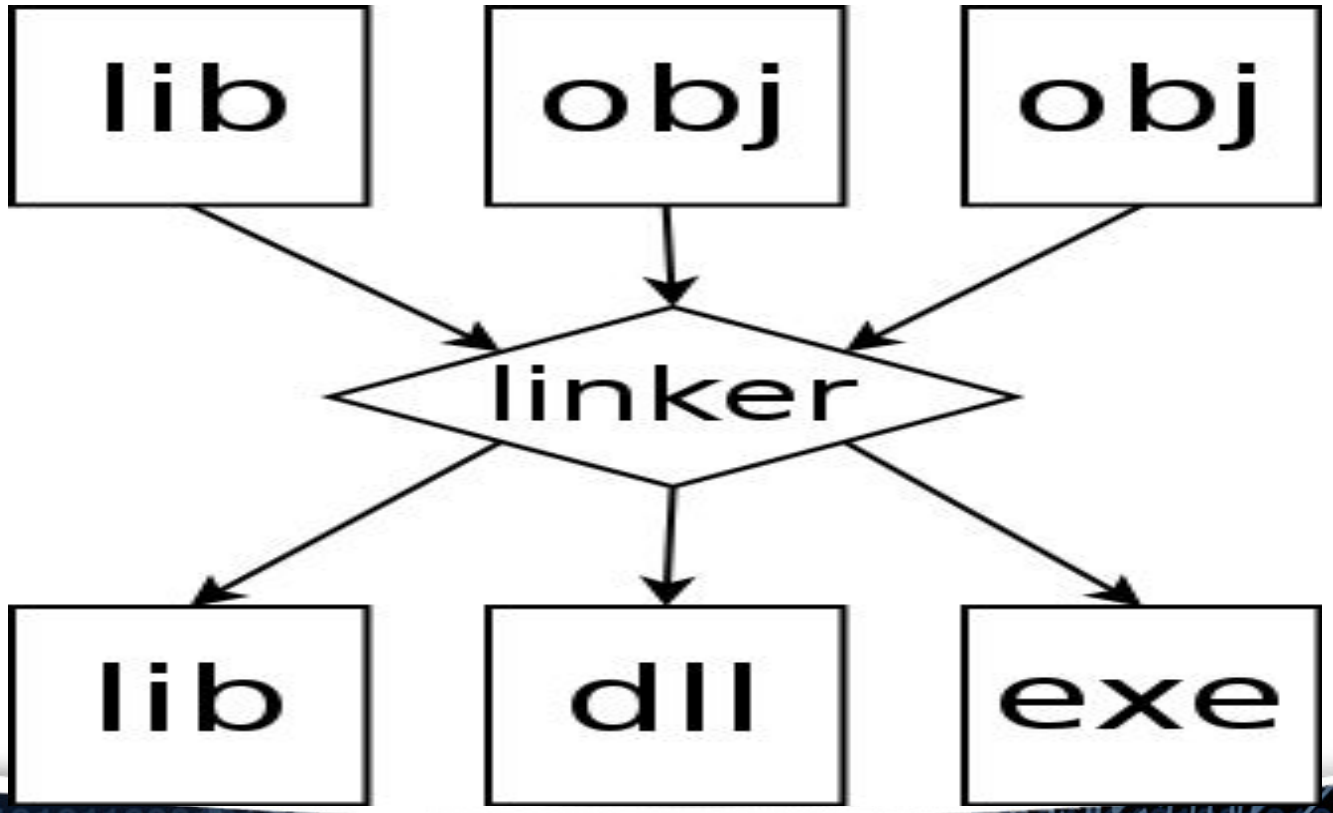
Comparison between Compiler and Interpreter

Compiler	Interpreter
Converts the source code program into m/c code before program execution	Converts each source program line into m/c code as it executes, line by line.
Can be removed from memory after compilation is finished	Must be continuously resident in memory while the program is being executed
Compilation is faster, translation occurs with program execution	Interpretation is slower. There is no concurrent translation and so consumes time
Compilation results in an object code on direct execution	No object code results for that can be executed later
More complicated error checking	Easier error checking

Linker

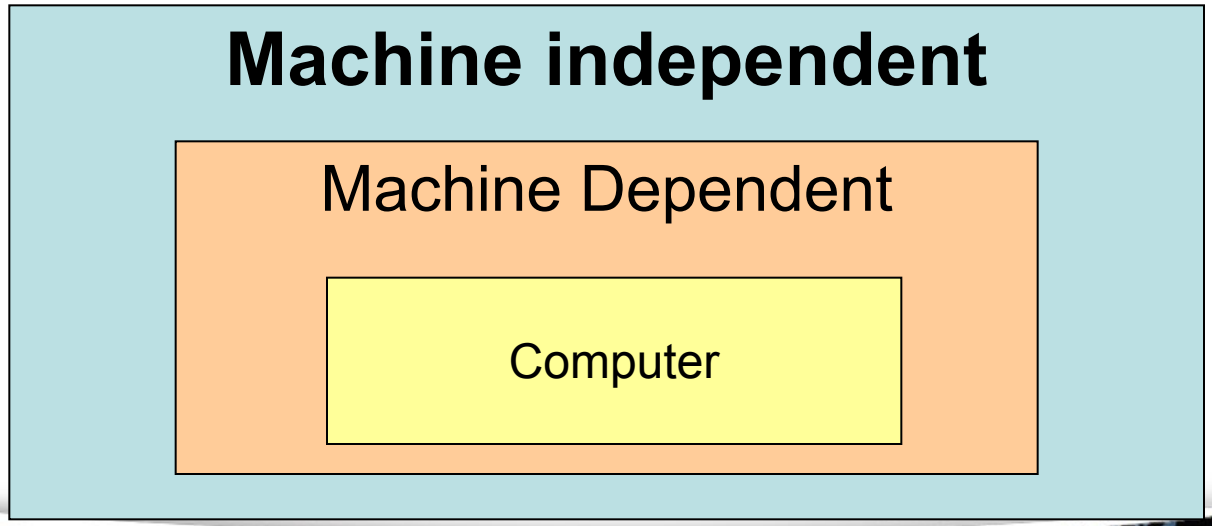
Is a program that takes one or more objects generated by compilers and assembles them into a single executable program.

Figure of the linking process, where object files and static libraries are assembled into a new library or executable.



System Software and Machine Architecture

- Machine dependent
  - Instruction Set, Instruction Format, Addressing Mode, Assembly language …
- Machine independent
  - General design logic/strategy, Two passes assembler…



1.3 The Simplified Instructional Computer

- Like many other products, SIC comes in two versions
  - The standard model
  - An XE version
    - “extra equipments”, “extra expensive”
- SIC (Simplified Instructional Computer)
- SIC/XE (Extra Equipment)

1.3 The Simplified Instructional Computer

- Memory:
  - Memory consists of 8-bit bytes, 3 consecutive bytes form a word (24 bits)
  - There are a total of 32768 bytes (32 KB) in the computer memory.
- Registers:
  - 5 registers, 24 bits in length

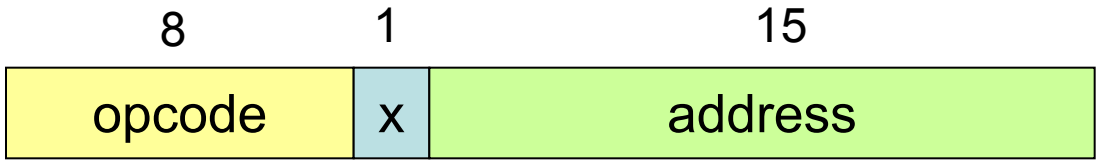
Mnemonic	Number	Special Use
A	0	Accumulator
X	1	Index register
L	2	Linkage register (JSUB)
PC	8	Program counter
SW	9	Status word (Condition Code)

1.3.1 SIC Machine Architecture

- Data Formats
  - Memory consists of 8 bit bytes
  - 3 consecutive bytes form a word.
  - 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

1.3.1 SIC Machine Architecture

- Instruction format
  - 24-bit format
  - The flag bit x is used to indicate addressing mode



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

- Addressing Modes
  - There are two addressing modes available
    - Indicated by x bit in the instruction
    - (X) represents the contents of register X

1.3.1 SIC Machine Architecture

- Instruction set
  - Load and store registers (LDA, LDX, STA, STX, etc.)
  - Integer arithmetic operations (ADD, SUB, MUL, DIV)
  - Compare instruction (COMP)
  - Conditional jump instructions (JLT, JEQ, JGT)
  - JSUB jumps to the subroutine, placing the return address in register L.
  - RSUB returns by jumping to the address contained in register L.

1.3.1 SIC Machine Architecture

- I/O
  - I/O are performed by transferring 1 byte at a time to or from the rightmost 8 bits of register A.
  - Each device is assigned a unique 8-bit code as an operand.
  - Test Device (TD): tests whether the addressed device is ready to send or receive
    - Condition code (flag)is set to indicate the result of this test.
    - < ready = not ready
  - Read Data (RD)
  - Write Data (WD)

1.3.2 SIC/XE Machine Architecture

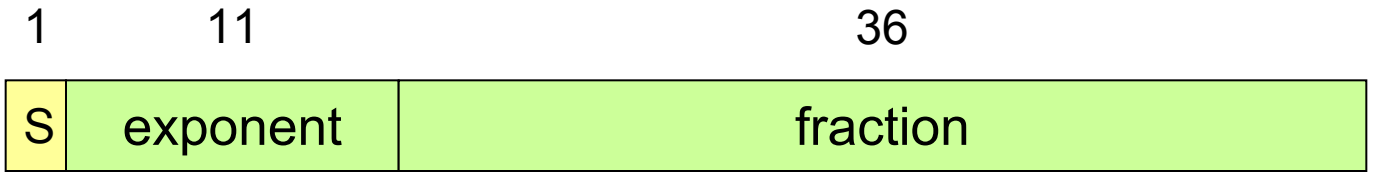
- Memory:

-- 1 megabytes (1024 KB) in memory

- Registers:
- 3 additional registers, 24 bits in length
  - B 3 Base register; used for addressing
  - S4 General working register
  - T 5 General working register
- 1 additional register, 48 bits in length
  - F 6 Floating-point accumulator (48 bits)

1.3.2 SIC/XE Machine Architecture

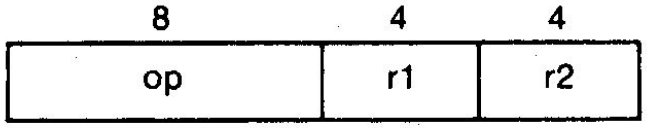
- Data format
  - 24-bit binary number for integer
  - 2's complement for negative values
  - 48-bit floating-point data type
  - The exponent is between 0 and 2047
  - 0: set all bits to 0



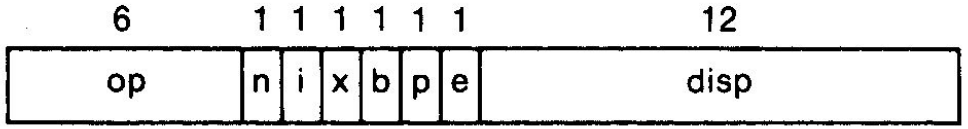
1.3.2 SIC/XE Machine Architecture

- Instruction formats
  - Relative addressing - format 3 (e=0)
  - Extend the address to 20 bits - format 4 (e=1)
  - Don't refer memory at all - formats 1 and 2

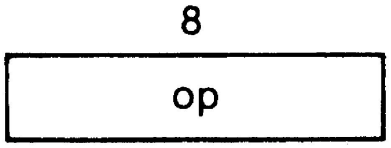
Format 2 (2 bytes):



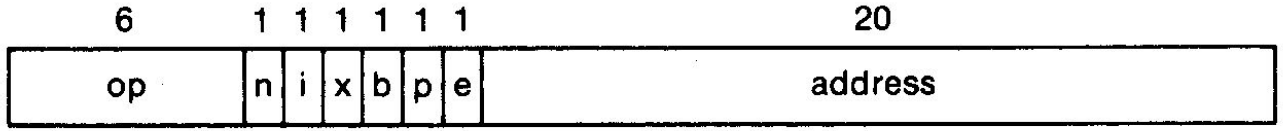
Format 3 (3 bytes):



Format 1 (1 byte):



Format 4 (4 bytes):



1.3.2 SIC/XE Machine Architecture

- Addressing modes
  - n i x b p e
  - Simple n=0, i=0 (SIC) or n=1, i=1
  - Immediaten=0, i=1 TA=Values
  - Indirect n=1, i=0 TA=(Operand)
  - Base relative b=1, p=0 TA=(B)+disp
    0 <= disp <= 4095
  - PC relative b=0, p=1 TA=(PC)+disp
    -2048 <= disp <= 2047

Mode	Indication	Target address calculation	
Base relative	b = 1, p = 0	TA = (B) + disp	(0 ≤ disp ≤ 4095)
Program-counter relative	b = 0, p = 1	TA = (PC) + disp	(–2048 ≤ disp ≤ 2047)

1.3.2 SIC/XE Machine Architecture

- Addressing mode
  - Direct b=0, p=0 TA=disp
  - Index x=1 $TA_{new}=TA_{old}+(X)$
  - Index+Base relative x=1, b=1, p=0 TA=(B)+disp+(X)
  - Index+PC relative x=1, b=0, p=1 TA=(PC)+disp+(X)
  - Index+Direct x=1, b=0, p=0 TA = DISP + (X)
  - Format 4 e=1 TA= address in instrucion

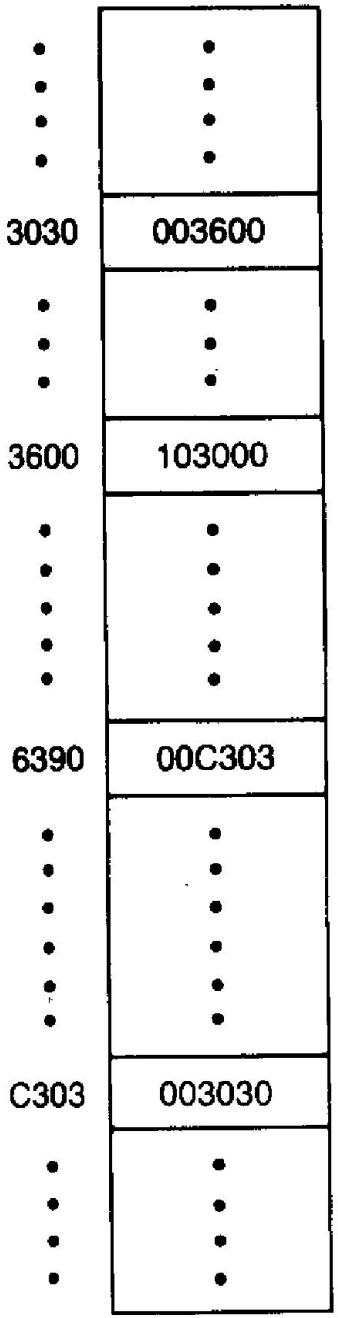
Figure 1.1: Example of SIC Instruction & Addressing modes.

(B) = 006000

(PC) = 003000

(X) = 000090

Machine Instruction									Target Address	Value loaded into register A
Hex	Binary							Disp/address		
	op	n	i	x	b	p	e			
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



(a)

1.3.2 SIC/XE Machine Architecture

- Instruction set
  - Format 1, 2, 3, or 4
  - Load and store registers (LDB, STB, etc.)
  - Floating-point arithmetic operations (ADDF, SUBF, MULF, DIVF)
  - Register-to-register arithmetic operations (ADDR, SUBR, MULR, DIVR)

- I/O
  - 1 byte at a time. TD, RD, and WD are used to Test Device, Read Device, and Write Device.
  - SIO, TIO, and HIO are used to start, test, and halt the operation of I/O channels.

1.3.3 SIC Programming Examples

- Sample data movement operations
  - No memory-to-memory move instructions

```
        LDA     FIVE        LOAD CONSTANT 5 INTO REGISTER A
        STA     ALPHA       STORE IN ALPHA
        LDCH    CHARZ       LOAD CHARACTER 'Z' INTO REGISTER A
        STCH    C1          STORE IN CHARACTER VARIABLE C1
          .
          .
          .
ALPHA   RESW    1           ONE-WORD VARIABLE
FIVE    WORD    5           ONE-WORD CONSTANT
CHARZ   BYTE    C'Z'        ONE-BYTE CONSTANT
C1      RESB    1           ONE-BYTE VARIABLE
```

1.3.3 SIC/XE Programming Examples

```
        LDA     #5              LOAD VALUE 5 INTO REGISTER A
        STA     ALPHA           STORE IN ALPHA
        LDA     #90             LOAD ASCII CODE FOR 'Z' INTO REG A
        STCH    C1              STORE IN CHARACTER VARIABLE C1
          .
          .
          .
ALPHA   RESW    1               ONE-WORD VARIABLE
C1      RESB    1               ONE-BYTE VARIABLE
```

(b)

Figure 1.2 Sample data movement operations for (a) SIC and (b) SIC/XE.

1.3.3 SIC Programming Examples

- Sample arithmetic operations
  - (ALPHA+INCR-1) assign to BETA (Fig. 1.3)
  - (GAMMA+INCR-1) assign to DELTA

```
        LDA     ALPHA       LOAD ALPHA INTO REGISTER A
        ADD     INCR        ADD THE VALUE OF INCR
        SUB     ONE         SUBTRACT 1
        STA     BETA        STORE IN BETA
        LDA     GAMMA       LOAD GAMMA INTO REGISTER A
        ADD     INCR        ADD THE VALUE OF INCR
        SUB     ONE         SUBTRACT 1
        STA     DELTA       STORE IN DELTA
          .
          .
          .
ONE     WORD    1           ONE-WORD CONSTANT
.                           ONE-WORD VARIABLES
ALPHA   RESW    1
BETA    RESW    1
GAMMA   RESW    1
DELTA   RESW    1
INCR    RESW    1
```

1.3.3 SIC/XE Programming Examples

```
        LDS     INCR        LOAD VALUE OF INCR INTO REGISTER S
        LDA     ALPHA       LOAD ALPHA INTO REGISTER A
        ADDR    S,A         ADD THE VALUE OF INCR
        SUB     #1          SUBTRACT 1
        STA     BETA        STORE IN BETA
        LDA     GAMMA       LOAD GAMMA INTO REGISTER A
        ADDR    S,A         ADD THE VALUE OF INCR
        SUB     #1          SUBTRACT 1
        STA     DELTA       STORE IN DELTA
        .
        .
        .
.                           ONE WORD VARIABLES
ALPHA   RESW    1
BETA    RESW    1
GAMMA   RESW    1
DELTA   RESW    1
INCR    RESW    1
```

1.3.3 SIC Programming Examples

- String copy

Initialise the loop before entering

Index in the string

```
        LDX     ZERO        INITIALIZE INDEX REGISTER TO 0
MOVECH  LDCH    STR1,X      LOAD CHARACTER FROM STR1 INTO REG A
        STCH    STR2,X      STORE CHARACTER INTO STR2
        TIX     ELEVEN      ADD 1 TO INDEX, COMPARE RESULT TO 11
        JLT     MOVECH      LOOP IF INDEX IS LESS THAN 11
```

Increment by one index register X, compare with operand (11) and update condition code (CC)

Jump to beginning of loop, if CC is LT (Less Than)

```
STR1    BYTE    C'TEST STRING'    11-BYTE STRING CONSTANT
STR2    RESB    11                11-BYTE VARIABLE
.                                 ONE-WORD CONSTANTS
ZERO    WORD    0
ELEVEN  WORD    11
```

1.3.3 SIC/XE Programming Examples

```
        LDT     #11             INITIALIZE REGISTER T TO 11
        LDX     #0              INITIALIZE INDEX REGISTER TO 0
MOVECH  LDCH    STR1,X          LOAD CHARACTER FROM STR1 INTO REG A
        STCH    STR2,X          STORE CHARACTER INTO STR2
        TIXR    T               ADD 1 TO INDEX, COMPARE RESULT TO 11
        JLT     MOVECH          LOOP IF INDEX IS LESS THAN 11
        .
        .
        .
STR1    BYTE    C'TEST STRING'  11-BYTE STRING CONSTANT
STR2    RESB    11              11-BYTE VARIABLE
```

1.3.3 SIC Programming Examples

```
            LDA     ZERO        INITIALIZE INDEX VALUE TO 0
            STA     INDEX
ADDLP       LDX     INDEX       LOAD INDEX VALUE INTO REGISTER X
            LDA     ALPHA,X     LOAD WORD FROM ALPHA INTO REGISTER A
            ADD     BETA,X      ADD WORD FROM BETA
            STA     GAMMA,X     STORE THE RESULT IN A WORD IN GAMMA
            LDA     INDEX       ADD 3 TO INDEX VALUE
            ADD     THREE
            STA     INDEX
            COMP    K300        COMPARE NEW INDEX VALUE TO 300
            JLT     ADDLP       LOOP IF INDEX IS LESS THAN 300
            .
            .
            .
INDEX       RESW    1           ONE-WORD VARIABLE FOR INDEX VALUE
.                               ARRAY VARIABLES--100 WORDS EACH
ALPHA       RESW    100
BETA        RESW    100
GAMMA       RESW    100
.                               ONE-WORD CONSTANTS
ZERO        WORD    0
K300        WORD    300
```

1.3.3 SIC/XE Programming Examples

```
        LDS     #3          INITIALIZE REGISTER S TO 3
        LDT     #300        INITIALIZE REGISTER T TO 300
        LDX     #0          INITIALIZE INDEX REGISTER TO 0
ADDLP   LDA     ALPHA,X     LOAD WORD FROM ALPHA INTO REGISTER A
        ADD     BETA,X      ADD WORD FROM BETA
        STA     GAMMA,X     STORE THE RESULT IN A WORD IN GAMMA
        ADDR    S,X         ADD 3 TO INDEX VALUE
        COMPR   X,T         COMPARE NEW INDEX VALUE TO 300
        JLT     ADDLP       LOOP IF INDEX VALUE IS LESS THAN 300
        .
        .
        .
.                           ARRAY VARIABLES--100 WORDS EACH
ALPHA   RESW    100
BETA    RESW    100
GAMMA   RESW    100
```

(b)

Figure 1.5 Sample indexing and looping operations for (a) SIC and (b) SIC/XE.

1.3.3 SIC Programming Examples

```
INLOOP    TD      INDEV     TEST INPUT DEVICE
          JEQ     INLOOP    LOOP UNTIL DEVICE IS READY
          RD      INDEV     READ ONE BYTE INTO REGISTER A
          STCH    DATA      STORE BYTE THAT WAS READ
          .
          .
          .
OUTLP     TD      OUTDEV    TEST OUTPUT DEVICE
          JEQ     OUTLP     LOOP UNTIL DEVICE IS READY
          LDCH    DATA      LOAD DATA BYTE INTO REGISTER A
          WD      OUTDEV    WRITE ONE BYTE TO OUTPUT DEVICE
          .
          .
          .
INDEV     BYTE    X'F1'     INPUT DEVICE NUMBER
OUTDEV    BYTE    X'05'     OUTPUT DEVICE NUMBER
DATA      RESB    1         ONE-BYTE VARIABLE
```

Figure 1.6 Sample input and output operations for SIC.

1.3.3 SIC Programming Examples

```
            JSUB    READ            CALL READ SUBROUTINE
            .
            .
            .
.                                   SUBROUTINE TO READ 100-BYTE RECORD
READ        LDX     ZERO            INITIALIZE INDEX REGISTER TO 0
RLOOP       TD      INDEV           TEST INPUT DEVICE
            JEQ     RLOOP           LOOP IF DEVICE IS BUSY
            RD      INDEV           READ ONE BYTE INTO REGISTER A
            STCH    RECORD,X        STORE DATA BYTE INTO RECORD
            TIX     K100            ADD 1 TO INDEX AND COMPARE TO 100
            JLT     RLOOP           LOOP IF INDEX IS LESS THAN 100
            RSUB                    EXIT FROM SUBROUTINE
            .
            .
            .
INDEV       BYTE    X'F1'           INPUT DEVICE NUMBER
RECORD      RESB    100             100-BYTE BUFFER FOR INPUT RECORD
.                                   ONE-WORD CONSTANTS
ZERO        WORD    0
K100        WORD    100
```

1.3.3 SIC\XE Programming Examples

```
            JSUB    READ            CALL READ SUBROUTINE
            .
            .
            .
.                                   SUBROUTINE TO READ 100-BYTE RECORD
READ        LDX     #0              INITIALIZE INDEX REGISTER TO 0
            LDT     #100            INITIALIZE REGISTER T TO 100
RLOOP       TD      INDEV           TEST INPUT DEVICE
            JEQ     RLOOP           LOOP IF DEVICE IS BUSY
            RD      INDEV           READ ONE BYTE INTO REGISTER A
            STCH    RECORD,X        STORE DATA BYTE INTO RECORD
            TIXR    T               ADD 1 TO INDEX AND COMPARE TO 100
            JLT     RLOOP           LOOP IF INDEX IS LESS THAN 100
            RSUB                    EXIT FROM SUBROUTINE
            .
            .
            .
INDEV       BYTE    X'F1'           INPUT DEVICE NUMBER
RECORD      RESB    100             100-BYTE BUFFER FOR INPUT RECORD
```

- 1. Write a sequence of instructions for SIC to compute ALPHA equal to the product of BETA and GAMMA.

```
        LDA     BETA
        MUL     GAMMA
        STA     ALPHA
        :
        :
ALPHA   RESW  1
BETA    RESW 1
GAMMA       RESW 1
```

- 2. Write a sequence of instructions for SIC/XE to set ALPHA equal to 4 * BETA-9
- Use immediate addressing for the constants.

```
LDA BETA
LDS #4
MULR    S,A
SUB     #9
STA ALPHA
:
:
ALPHA   RESW   1
BETA    RESW    1
```

- 3. Write SIC instructions to swap the values of ALPHA and BETA.

```
LDA     ALPHA
STA      GAMMA
LDA     BETA
STA     ALPHA
LDA     GAMMA
STA     BETA
:
ALPHA RESW  1
BETA   RESW  1
GAMMA     RESW 1
```

- 4. Write a sequence of instructions for SIC to set ALPHA equal to the integer portion of BETA ÷ GAMMA.

```
LDA     BETA
DIV     GAMMA
STA     ALPHA
:
ALPHA  RESW 1
BETA   RESW 1
GAMMA     RESW 1
```

- 5. Write a sequence of instructions for SIC/XE to divide BETA by GAMMA, setting ALPHA to the integer portion of the quotient and DELTA to the remainder. Use register-to-register instructions to make the calculation as efficient as possible.

```
LDA BETA
LDS GAMMA
DIVR S, A
STA ALPHA
MULR S, A
LDS BETA
SUBR A, S
STS DELTA
:
 ALPHA RESW 1
BETA RESW 1
GAMMA RESW 1
DELTA RESW 1
```

- 6. Write a sequence of instructions for SIC/XE to divide BETA by GAMMA, setting ALPHA to the value of the quotient, rounded to the nearest integer. Use register-to-register instructions to make the calculation as efficient as possible.

```
LDF BETA
DIVF GAMMA
FIX
STA ALPHA
:
 ALPHA RESW 1
BETA RESW 1
GAMMA RESW 1
```

- 7. Write a sequence of instructions for SIC to clear a 20-byte string to all blanks.

```
      LDX     ZERO
LOOP  LDCH  BLANK
      STCH  STR1,X
      TIX     TWENTY
      JLT LOOP
      :
      STR1        RESW  20
      BLANK       BYTE   C ‘ ‘
      ZERO        WORD  0
      TWENTY    WORD   20
```

- 8. Write a sequence of instructions for SIC/XE to clear a 20-byte string to all blanks. Use immediate addressing and register-to-register instructions to make the process as efficient as possible.

```
        LDT #20
        LDX #0
LOOP    LDCH #0
        STCH STR1,X
        TIXR T
        JLT LOOP
        :
        STR1 RESW 20
```

-

- 9. Suppose that ALPHA is an array of 100 words. Write a sequence of instructions for SIC to set all 100 elements of the array to 0.

```
        LDA ZERO
        STA INDEX
LOOP    LDX INDEX
        LDA ZERO
        STA ALPHA, X
        LDA INDEX
        ADD THREE
        STA INDEX
        COMP K300
        TIX TWENTY
        JLT LOOP
        :
        INDEX RESW 1
        ALPHA RESW 100
        ZERO WORD 0
        K300 WORD 100
        THREE WORD 3
```

- 10. Suppose that ALPHA is an array of 100 words.Write a sequence of instructions for SIC/XE to set all 100 elements of the array to 0. Use immediate addressing and register-to-register instructions to make the process as efficient as possible.

```
        LDS #3
        LDT #300
        LDX #0
LOOP    LDA #0
        STA    ALPHA, X
         ADDR S, X
        COMPR X, T
        JLT LOOP
        :
ALPHA   RESW 100
```

- 12. Suppose that ALPHA and BETA are the two arrays of 100 words. Another array of GAMMA elements are obtained by multiplying the corresponding ALPHA element by 4 and adding the corresponding BETA elements.

```
        LDS #3
        LDT #300
        LDX #0
ADDLOOP     LDA ALPHA, X
        MUL #4
        ADD BETA, X
        STA GAMMA, X
        ADDR S, X
        COMPR X, T
        JLT ADDLOOP
        :
         ALPHA RESW 100
        BETA RESW 100
        GAMMA RESW 100
```

- 13. Suppose that ALPHA is an array of 100 words. Write a sequence of instructions for SIC/XE to find the maximum element in the array and store results in MAX.

```
        LDS #3
        LDT #300
        LDX #0
CLOOP   LDA ALPHA, X
        COMP MAX
        JLT NOCH
        STA MAX
NOCH    ADDR S, X
        COMPR X, T
        JLT CLOOP
        :
        ALPHA RESW 100
        MAX WORD -32768
```

- 14. Suppose that RECORD contains a 100-byte record. Write a subroutine for SIC that will write this record on to device 05.

```
            JSUB WRREC
:
WRREC   LDX ZERO
WLOOP   TD OUTPUT
            JEQ WLOOP
            LDCH RECORD, X
            WD OUTPUT
            TIX LENGTH
            JLT WLOOP
            RSUB
            :
ZERO        WORD 0
LENGTH  WORD 1
OUTPUT  BYTE X “05‟
 RECORD       RESB 100
```

/

- 15. Suppose that RECORD contains a 100-byte record,. Write a subroutine for SIC/XE that will write this record on to device 05.

```
            JSUB    WRREC
        :
WRREC  LDX #0
        LDT #100
WLOOP  TD OUTPUT
        JEQ WLOOP
        LDCH RECORD, X
        WD OUTPUT
        TIXR T
        JLT WLOOP
        RSUB
        :
OUTPUT BYTE X ’05’
 RECORD      RESB 100
```

- 16. Write a subroutine for SIC that will read a record into a buffer. The record may be any length from 1 to 100 bytes. The end of record is marked with a “null” character (ASCII code 00). The subroutine should place the length of the record read into a variable named LENGTH.

```
        JSUB RDREC
        :
RDREC   LDX ZERO
RLOOP   TD INDEV
        JEQ RLOOP
        RD INDEV
        COMP NULL
        JEQ EXIT
        STCH BUFFER, X
        TIX K100
        JLT RLOOP
EXIT        STX LENGTH
        RSUB
        :
ZERO        WORD 0
NULL        WORD 0
K100        WORD 1INDEV
BYTE X ‘F1’
 LENGTH  RESW 1
BUFFER  RESB 100
```

- 17. Write a subroutine for SIC/XE that will read a record into a buffer. The record may be any length from 1 to 100 bytes. The end of record is marked with a “null” character (ASCII code 00). The subroutine should place the length of the record read into a variable named LENGTH. Use immediate addressing and register-to-register instructions to make the process as efficient as possible.

```
        JSUB RDREC
        :
RDREC   LDX #0
        LDT #100
        LDS #0
RLOOP   TD INDEV
        JEQ RLOOP
        RD INDEV
        COMPR A, S
        JEQ EXIT
        STCH BUFFER, X
        TIXR T
        JLT RLOOP
EXIR    STX LENGTH
        RSUB
        :
INDEV   BYTE X ‘F1’
LENGTH  RESW 1
BUFFER  RESB 100
```

-