Module – 1

Introduction : System Software Vs. Application Software, Different System Software– Assembler, Linker, Loader, Macro Processor, Text Editor, Debugger, Device Driver, Compiler, Interpreter, Operating System(Basic Concepts only)
SIC & SIC/XE Architecture, Addressing modes, SIC & SIC/XE Instruction set, Assembler Directives and Programming.

1.1 System Software Vs. Application Software

- Computer softwares are divided into 2 types :
  1. Application software
  2. System software

Application software

- An application program is primarily concerned with the solution of some problem, using the computer as a tool.
- The focus is on the application, not on the computing system.
- Purpose is supporting or improving the software user's work
- Employs the capabilities of computer directly to a dedicated task that the user wishes to perform
- Types of application software are as follows:
  - Word processing S/W - MS word, Notepad, WordPad
  - Database S/W - Oracle, MS Access
  - Spreadsheet S/W - Excel, Lotus, Apple Numbers
  - Multimedia S/W - Real Player, Media Player
  - Presentation Graphics S/W -MS PowerPoint

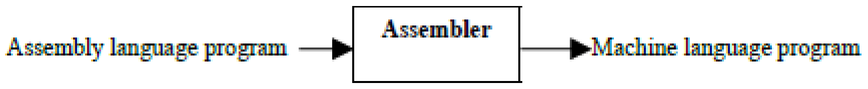
System software

- System software consists of a variety of programs that support the operation of a computer.
- Acts as an intermediary b/w computer hardware & application programs
- This software makes it possible for the user to focus on an application or other problem to be solved, without needing to know the details of how the machine works internally.
- Controls the computer system & enhances its performance
- System programs are intended to support the operation and use of the computer itself, rather than any particular application.
- For this reason, they are usually related to the architecture of the machine on which they are run.
- Focus is on the system
- Examples : Assembler , Loader / Linker, Macro processor ,Text Editor, Debugger ,Device driver, Compiler, Interpreter ,Operating system
- **The characteristic in which system software differs from application software is machine dependency**.

1.2 Different System Softwares

1. Assembler

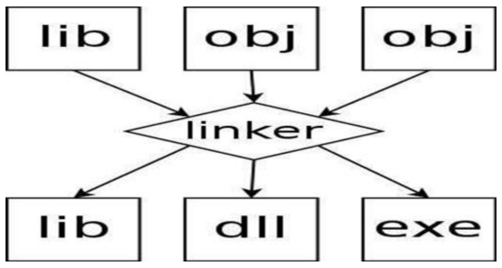
- Programmers found it difficult to write or read programs in machine language.
- In a quest for a convenient language, they began to use a mnemonic (symbol) for each machine instructions which would subsequently be translated into machine language. Such a mnemonic language is called Assembly language.
- Programs known as Assemblers are written to automate the translation of assembly language into machine language.



- Fundamental functions:
    1. Translating mnemonic operation codes to their machine language equivalents.
    2. Assigning machine addresses to symbolic tables used by the programmers.

2. Linker

Combines two or more separate object programs and supplies the information needed to allow references between them



3. Loader

Loader is a system program that loads the object program into memory for execution.

4. Macro processor

- A **macro** represents a commonly used group of statements in the source programming language.
- A macro instruction is simply a notational convenience for the programmer to write a shorthand version of a program.
- The macro instruction is replaced by the macro processor with the corresponding group of source language statements. This operation is called “expanding the macro”
- For example:

    – Suppose it is necessary to save the contents of all registers before calling a subroutine.

– This requires a sequence of instructions.
– We can define and use a macro, SAVEREGS, to represent this sequence of instructions.

- A **macro processor** functions essentially involve the substitution of one group of characters or lines for another. Normally, it performs no analysis of the text it handles.
- Macro processors are used in
  - assembly language
  - high-level programming languages, e.g., C or C++

5. Text editor

- An Interactive text editor has become an important part of almost any computing environment

- Text editor acts as a primary interface to the computer for all type of "knowledge workers" as they compose, organize, study, and manipulate computer-based information

- A text editor allows you to edit a text file (create, modify etc…)

- Text editors on Windows OS

  -Notepad, WordPad, Microsoft Word

- Text editors on UNIX OS

  - vi, emacs, jed, pico,

6. Debugger

- Tests & debugs errors

- Debugging is a methodical process of finding and reducing the number of bugs, or defects, in a computer program

- An interactive debugging system provides programmers with facilities that aid in testing and debugging of programs

- Conditional Expressions – Programmers can define some conditional expressions ,evaluated during the debugging session, program execution is suspended, when conditions are met, analysis is made, later execution is resumed

- Breakpoint- The programmer may define break points which cause execution to be suspended, when a specified point in the program is reached. After execution is suspended, the debugging command is used to analyze the progress of the program and to diagnose errors detected.

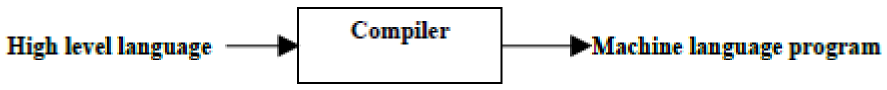
- A Debugging system should also provide functions such as tracing and trace back. Trace back can show the path by which the current statement in the program was reached. It can also show which statements have modified a given variable or parameter.

7. Device driver

- They are distinct "black boxes" that make a particular piece of hardware respond to a well-defined internal programming interface
- They hide completely the details of how the device works.
- User activities are performed by means of a set of standardized calls that are independent of the specific driver; mapping those calls to device-specific operations that act on real hardware is then the role of the device driver.
- This programming interface is such that drivers can be built separately from the rest of the kernel, and "plugged in" at runtime when needed.
- This modularity makes Linux drivers easy to write, to the point that there are now hundreds of them available.

8. Compiler

- A compiler is a program that translates programs written in any high level language into its equivalent machine language program.
- It bridges the semantic gap between a programming language domain and the execution domain.
- The program instructions are taken as a whole.



9. Interpreter

- It is a translator program that translates a statement of high-level language to machine language and executes it immediately. The program instructions are taken line by line.
- The interpreter reads the source program and stores it in memory
- During interpretation, it takes a source statement, determines its meaning and performs actions which implements it. This includes computational and I/O actions.
- Program counter (PC) indicates which statement of the source program is to be interpreted next. This statement would be subjected to the interpretation cycle.
- The interpretation cycle consists of the following steps:

    - Fetch the statement.
    - Analyze the statement and determine its meaning.
    - Execute the meaning of the statement.
- The following are the characteristics of interpretation:
    - The source program is retained in the source form itself, no target program exists.
    - A statement is analyzed during the interpretation.

10. Operating system

- It is the most important system program that act as an interface between the users and the system.
- It makes the computer easier to use.
- It provides an interface that is more user-friendly than the underlying hardware.
- The functions of OS are:
    - Process management
    - Memory management
    - Resource management
    - I/O operations
    - Data management
    - Providing security to user's job.

Difference between system software and application software

System software	Application software
System software is used for operating computer hardware.	Application software is used by user to perform specific task.
System softwares are installed on the computer when operating system is installed.	Application softwares are installed according to user's requirements.
In general, the user does not interact with system software because it works in the background.	In general, the user interacts with application sofwares.
System software can run independently. It provides platform for running application softwares.	Application software can't run independently. They can't run without the presence of system software.
Some examples of system softwares are compiler, assembler, debugger, driver, etc.	Some examples of application softwares are word processor, web browser, media player, etc.

1.3 The Simplified Instructional Computer (SIC)

- SIC is a hypothetical computer that has been carefully designed to include the hardware features most often found on real machines, while avoiding unusual or irrelevant complexities.
- It is similar to a typical microcomputer. It comes in two versions:
    - The standard model
    - XE version (Extra equipment, Extra expensive)

- The two versions have been designed to be **upward compatible** (An object program for the standard SIC machine will also execute properly on a SIC/XE system)

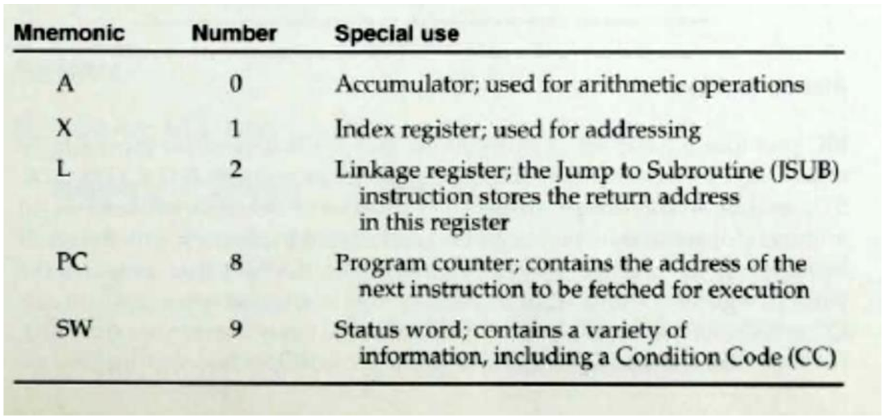
SIC Machine Architecture:

Memory:

- It consists of bytes(8 bits) ,words (24 bits which are consecutive 3 bytes)addressed by the location of their lowest numbered byte.
- All SIC addresses are byte addresses
- There are totally 32,768 (2^15)bytes in memory.

Registers:

There are 5 registers all of which have special uses. Each register is 24 bits in length



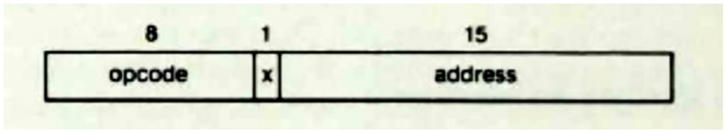
Mnemonic	Number	Special use
A	0	Accumulator; used for arithmetic operations
X	1	Index register; used for addressing
L	2	Linkage register; the Jump to Subroutine (JSUB) instruction stores the return address in this register
PC	8	Program counter; contains the address of the next instruction to be fetched for execution
SW	9	Status word; contains a variety of information, including a Condition Code (CC)

Data formats:

- Integers are stored as 24-bit binary numbers
- 2’s complement representation is used for negative values
- Characters are stored using their 8 bit ASCII codes.
- They do not support floating – point data items.

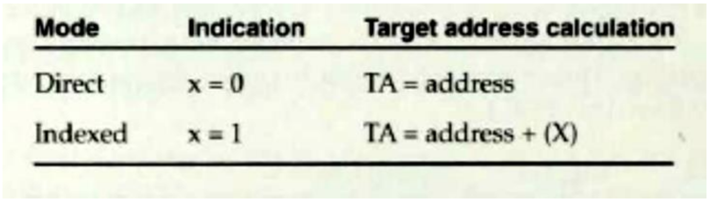
Instruction formats

All machine instructions are of 24-bits wide. The flag bit x is used to indicate indexed addressing mode.



Addressing modes:

Two types of addressing are available namely,
1. Direct addressing mode
2. Indexed addressing mode



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

- Parantheses are used to indicate the contents of a register or a memory location.
- For example (x) represents the contents of the index register X.

Instruction set

It includes instructions like:

1. Data movement instruction
   Ex: LDA, LDX, STA, STX.

2. Arithmetic operating instructions
   Ex: ADD, SUB, MUL, DIV.
   All arithmetic operations involve register A and a word in memory, with the result being left in the register.

3. Comparison instruction
   Ex: COMP – the value in the register A is compared with a word in memory. This instruction sets a condition code (CC) flag to indicate the result (<, >, =)

4.Conditional jump instructions
   Ex: JLT, JEQ, JGT.

4. Subroutine linkage instructions
   Ex: JSUB, RSUB.
   JSUB – Jumps to the subroutine, placing the return address in register L.

RSUB – Returns by jumping to the address contained in register L

Input and Output:

- I/O is performed by transferring one byte at a time to or from the rightmost 8 bitsof register A.
- Each device is assigned a unique 8-bit code.
- There are 3 I/O instructions each of which specifies the device code as an operand.

1) The Test Device (TD) - instructions tests whether the addressed device is ready to send or receive a byte of data.
Condition code is set to indicate the result of this test.
Setting of '<' indicates the device is set to send or receive.
Setting of '= ' indicates the device is not ready

2) RD – Read Data

3) WD – Write Data

A program must wait until the device is ready, and then execute a ReadData (RD) or Write Data (WD. The sequence must be repeated for each byte of data to be read or written.

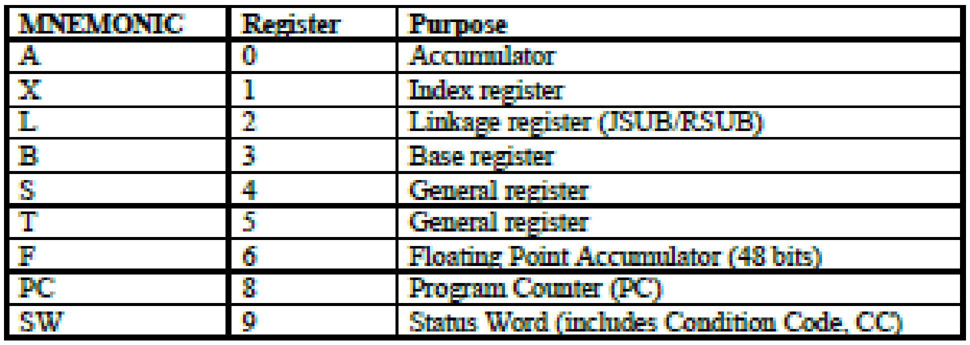
1.4 SIC/XE Architecture

Memory:

- 1 word = 24 bits (3 8-bit bytes)
- Total (SIC/XE) = 2^20 (1,048,576) bytes (1Mbyte)

Registers:

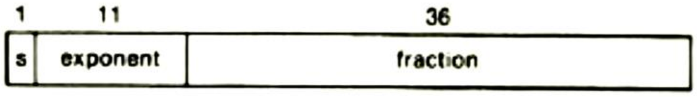
Base register is used for addressing.



MNEMONIC	Register	Purpose
A	0	Accumulator
X	1	Index register
L	2	Linkage register (JSUB/RSUB)
B	3	Base register
S	4	General register
T	5	General register
F	6	Floating Point Accumulator (48 bits)
PC	8	Program Counter (PC)
SW	9	Status Word (includes Condition Code, CC)

Data Format:

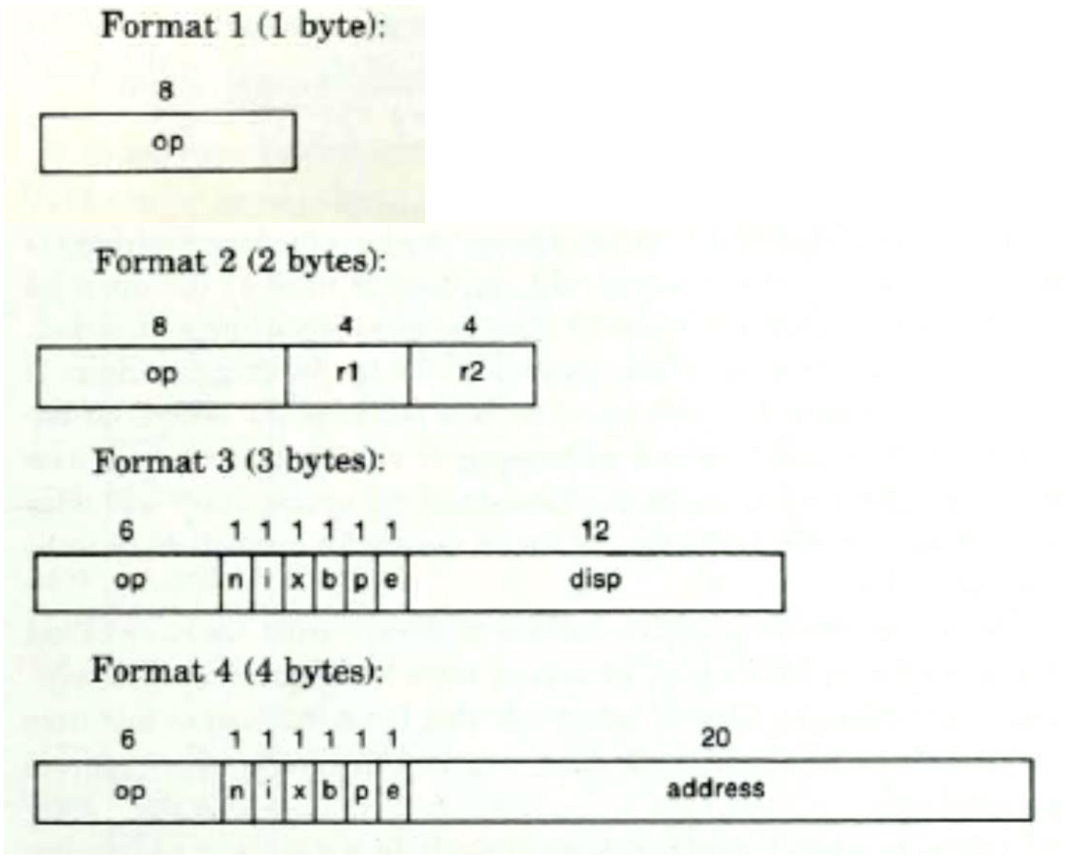
- Integers are stored in 24 bit, 2's complement format
- Characters are stored in 8-bit ASCII format
- Floating point is stored in 48 bit signed-exponent-fraction format:



- The fraction is represented as a 36 bit number and has value between 0 and 1.That is the assumed binary point is immediately before the higher order bit.
- The exponent is represented as a 11 bit unsigned binary number between 0 and2047.
- The sign of the floating point number is indicated by s : 0=positive, 1=negative.
- If the exponent has value e and the fraction has value f, the absolute value of the number represented is f*2^(e-1024).

Instruction Format:

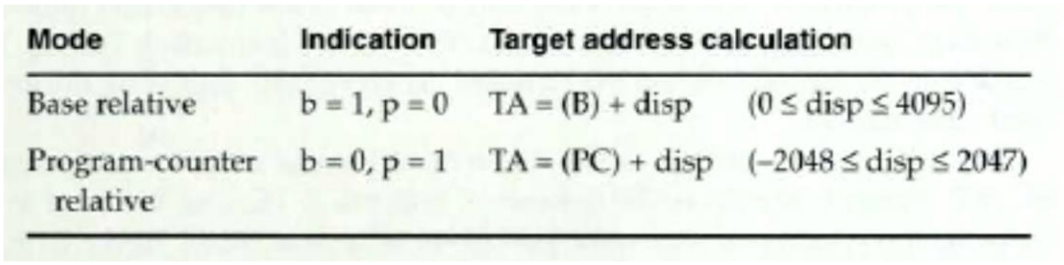
· There are 4 different instruction formats available:



- Format 1 and 2 do not refer memory at all.
- Bit e is used to distinguish between Formats 3 and 4

Addressing modes

Two new addressing modes are available for use with instructions assembled using Format 3



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)

- For **base relative addressing**, the displacement field disp in a Format 3 instruction is interpreted as a 12-bit unsigned integer.
- For **program – counter relative addressing,** disp field is interpreted as a 12 – bit signed integer with negative values represented in 2's complement notation

Flags b and P:

- b=0 & p=0
  Direct addressing - displacement/address field contains TA (Format 4 always uses direct addressing)
- b=0 & p=1
  PC relative addressing - TA=(PC)+disp (-2048<=disp<=2047)
- b=1 & p=0
  Base relative addressing - TA=(B)+disp (0<=disp<=4095)

Flags n and i :

- n=0 & i=1
  Immediate addressing - TA is used as an operand value (no memory reference)
- n=1 & i=0
  Indirect addressing - word at the location given by TA (in memory) is fetched ,the value contained in this word is then taken as the address of the operand value.
- n=0 & i=0
  Simple addressing -TA is the location of the operand
  In this case bits b,p,e are considered to be part of the address field of the instruction. Hence Format 3 identical to the standarad SIC format providing the desired compatibility.

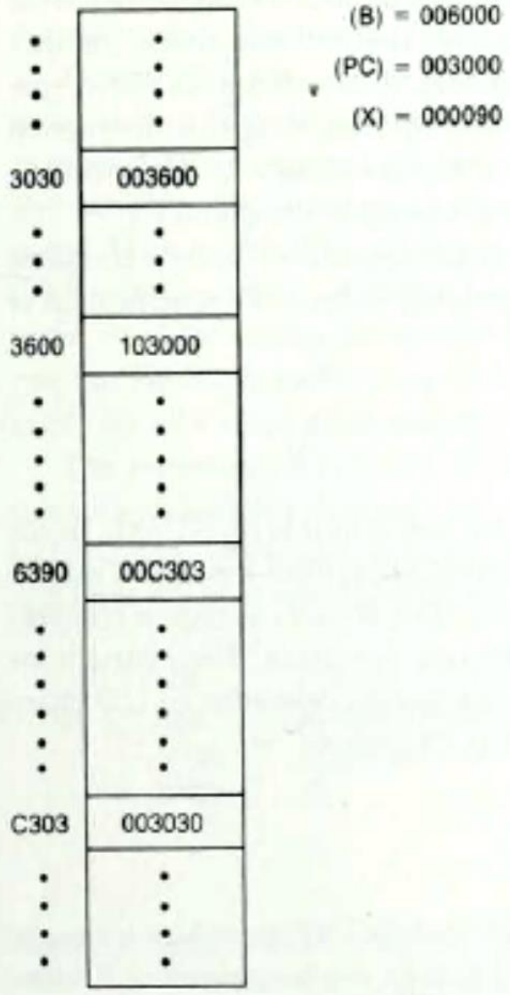
· n=1 & i=1
  Simple addressing same as n=0 & i=0

Flag x:

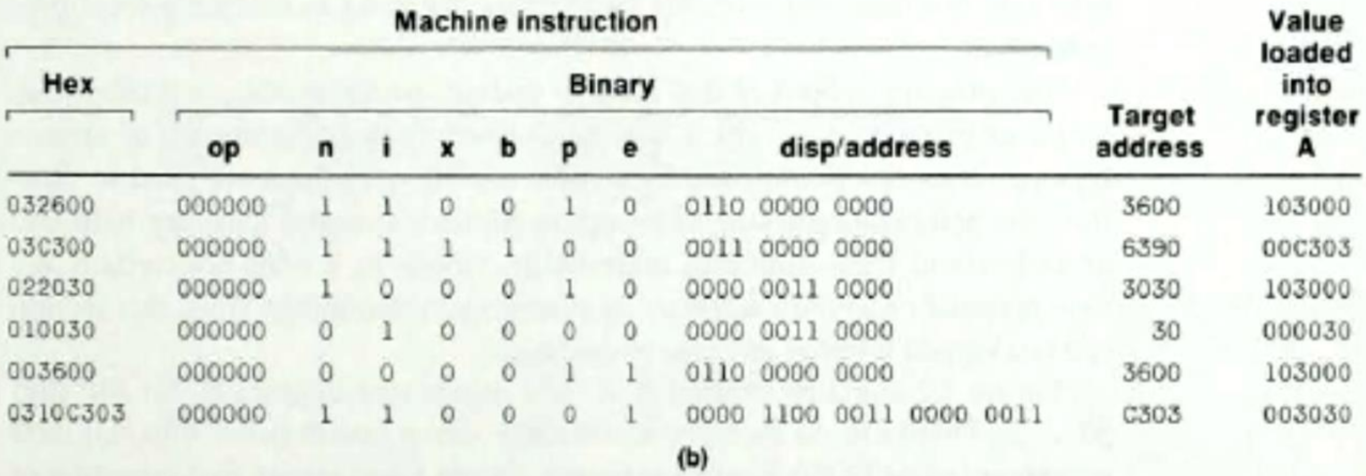
- x=1 Indexed addressing add contents of X register to TA calculation

Flag e:

- e=0 use Format 3
- e=1 use Format 4



(a)



Hex	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

Machine instruction: Hex; Binary (op, n, i, x, b, p, e, disp/address)

(b)

Fig: Examples of SIC/XE instructions and addressing modes

Instruction set

- SIC provides 26 instructions,
- SIC/XE provides an additional 33 instructions (59 total)
- SIC/XE has 9 categories of instructions:

  - Load/store registers (LDA, LDX, LDCH, STA, STX, STCH, etc.)
  - Integer arithmetic operations (ADD, SUB, MUL, DIV) these will use register A and a word in memory, results are placed into register A
  - Compare (COMP) compares contents of register A with a word in memory and sets CC (Condition Code) to <, >, or =
    - Conditional jumps (JLT, JEQ, JGT) - jumps according to setting of CC
  - Subroutine linkage (JSUB, RSUB) - jumps into/returns from subroutine using register L
  - Input & output control (RD, WD, TD)
  - Floating point arithmetic operations (ADDF, SUBF, MULF, DIVF)
  - Register manipulation, operands-from-registers, and register-to-register arithmetics (RMO, RSUB, COMPR, SHIFTR, SHIFTL, ADDR, SUBR, MULR, DIVR, etc)
  - Supervisor call instruction (SVC) – Executing this instruction generates an interrupt that can be used for communication with the operating system.

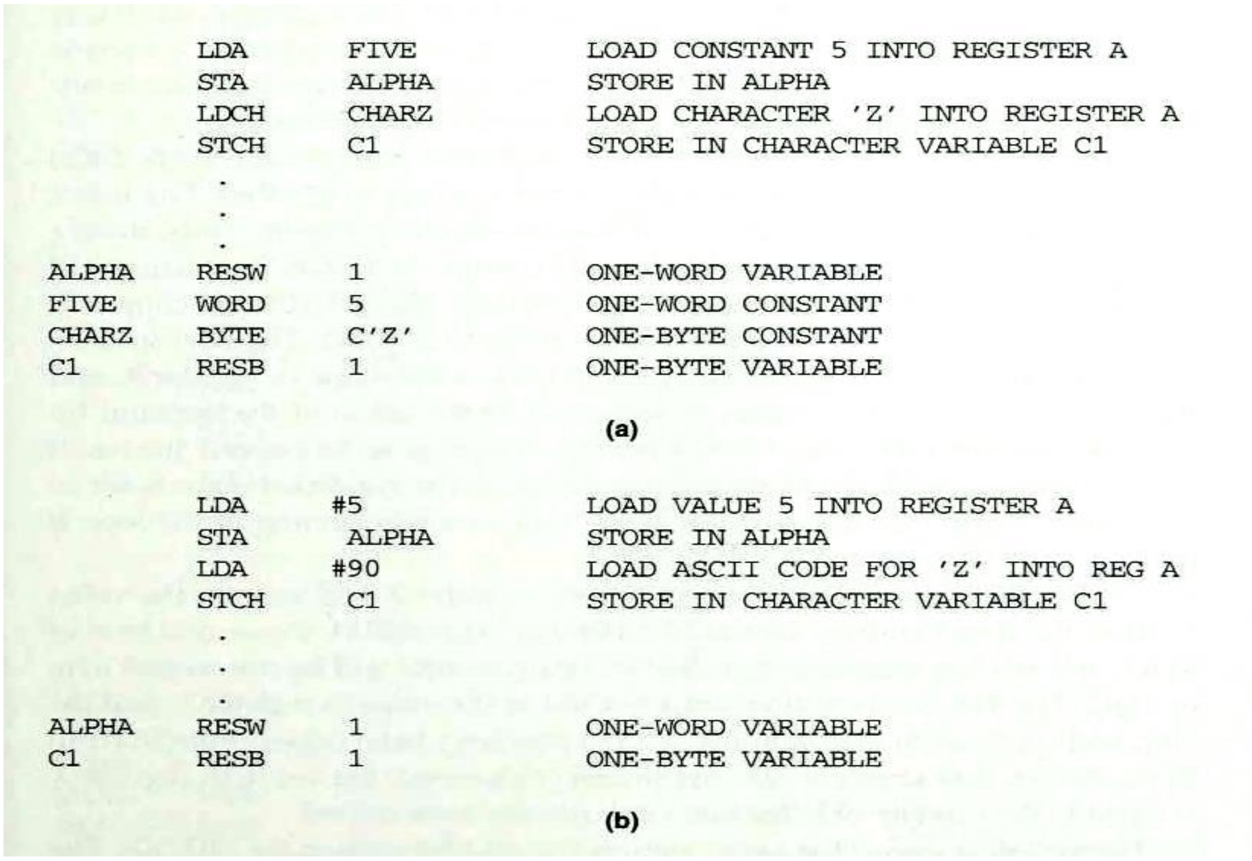
Input and Output (I/O)

- 28 (256) I/O devices may be attached, each has its own unique 8-bit address
- 1 byte of data will be transferred to/from the rightmost 8 bits of register A
- Three I/O instructions are provided:
  - RD - Read Data from I/O device into A
  - WD- Write data to I/O device from A
  - TD -Test Device determines if addressed I/O device is ready to send/receive a byteof data. The CC (Condition Code) gets set with results from this test:
    - < device is ready to send/receive
    - = device isn't ready
- SIC/XE have I/O channels that can be used to perform input and output while CPU does other work .This allows overlap of computing and I/O resulting in more efficient system operation.
- 3 additional instructions are provided:
  - SIO Start I/O
  - HIO Halt I/O
  - TIO Test I/O

Programming examples

The assembler directives used here are as follows

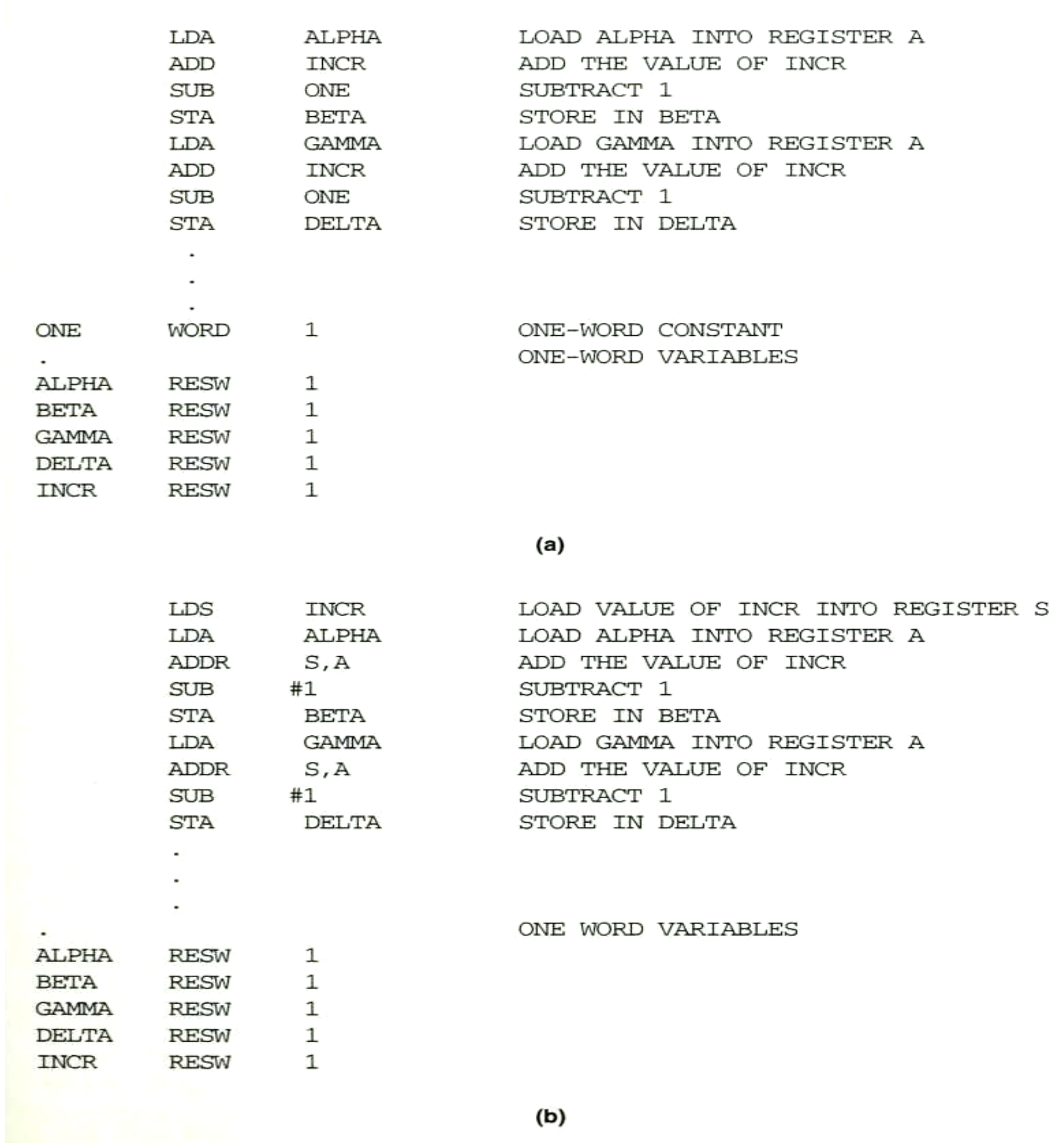
- **BYTE:** Generate character or hexadecimal constant, occupying as many bytes as needed to represent the constant.
- **WORD:** Generate one- word integer constant.
- **RESB:** Reserve the indicated number of bytes for a data area.
- **RESW:** Reserve the indicated number of words for a data area.



```
        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

                            (a)

        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)
```

Sample data movement operations for (a) SIC (b) SIC/XE

ALPHA = 5 and C1='Z'



```
          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
```

(a)

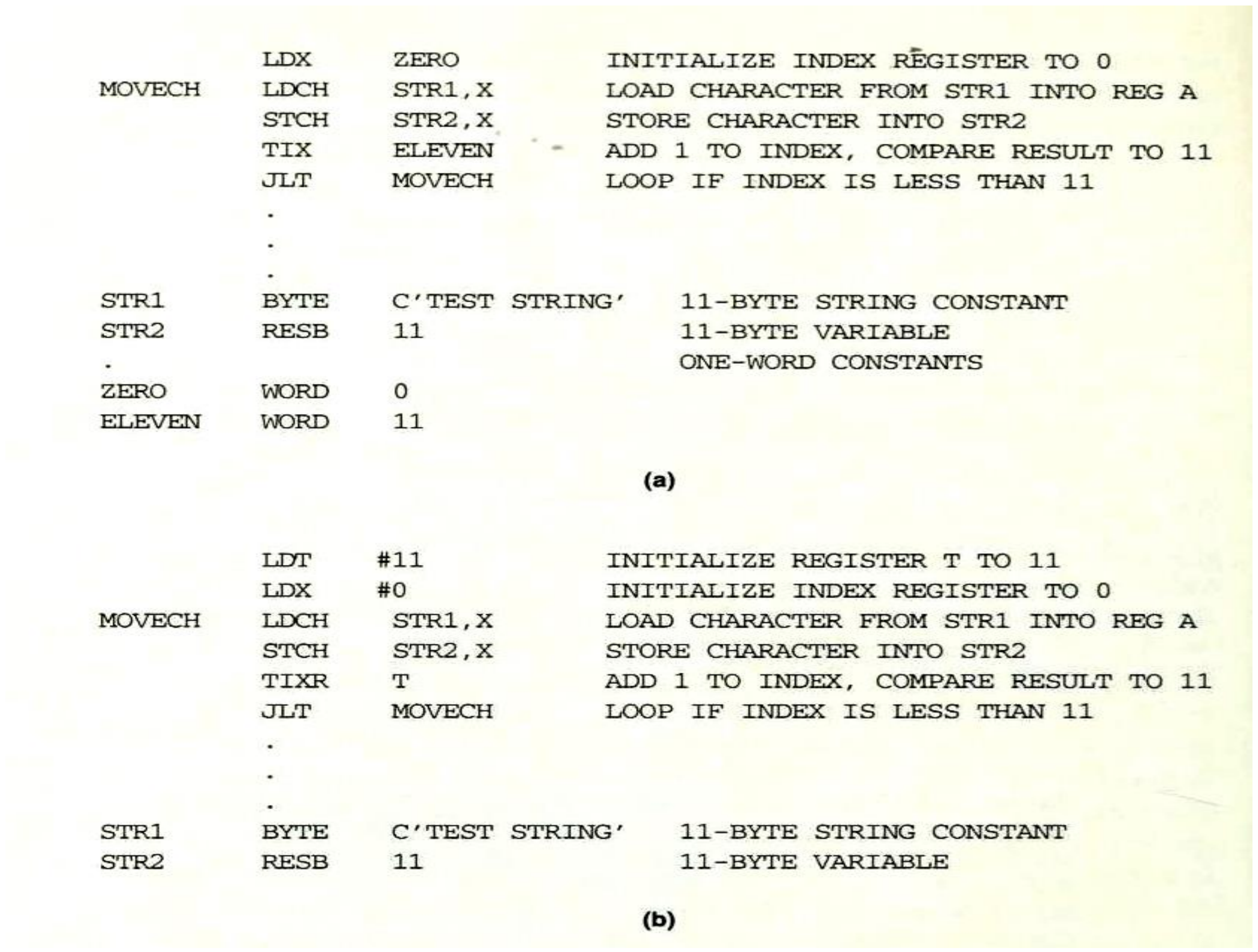
```
          LDS       INCR           LOAD VALUE OF INCR INTO REGISTER S
          LDA       ALPHA          LOAD ALPHA INTO REGISTER A
          ADDR      S,A            ADD THE VALUE OF INCR
          SUB      #1              SUBTRACT 1
          STA       BETA           STORE IN BETA
          LDA       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
```

(b)

Sample arithmetic operations for (a) SIC (b)SIC/XE

BETA = ALPHA + INCR -1

DELTA = GAMMA + INCR -1



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

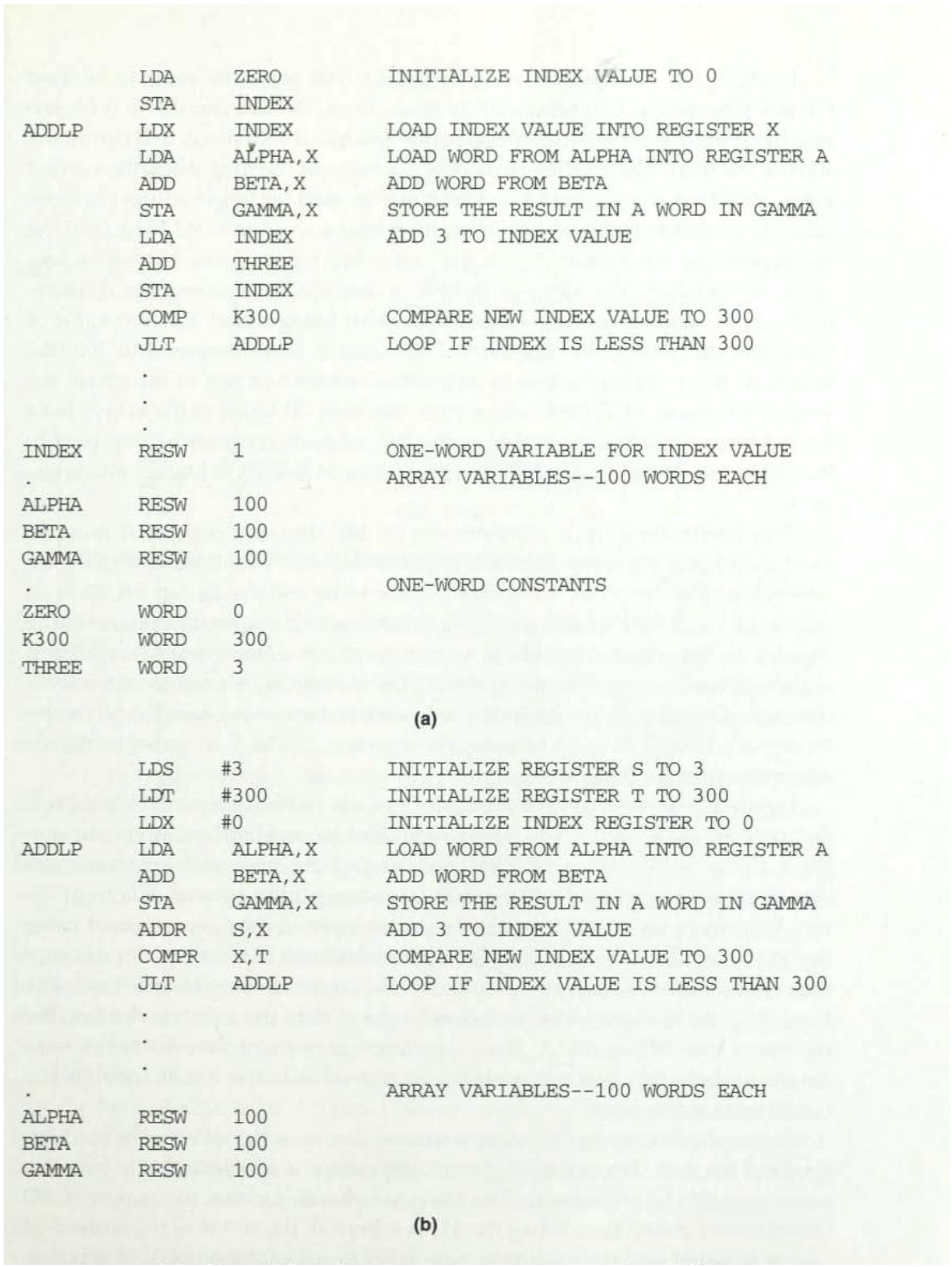
(a)

```
         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
```

(b)

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

Copy a string(TEST STRING) from STR1 to STR2



```
         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
THREE    WORD    3
```

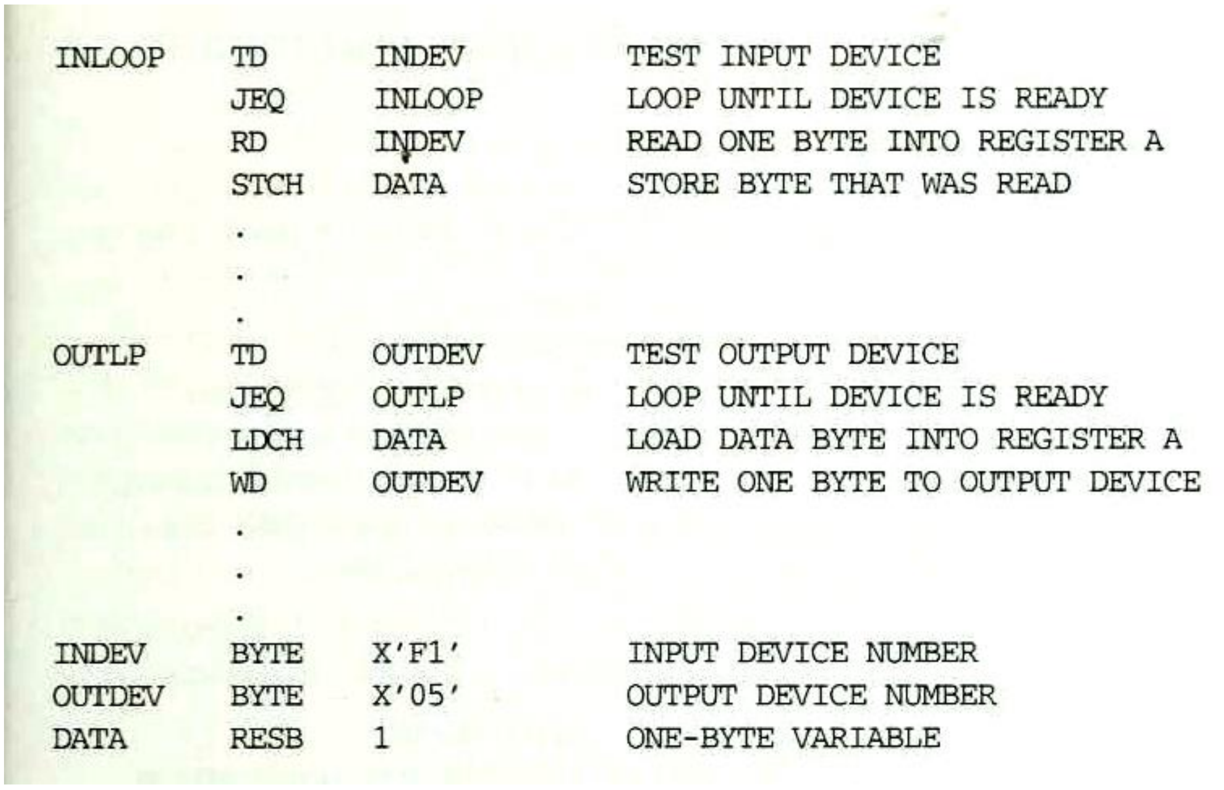
(a)

```
         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)

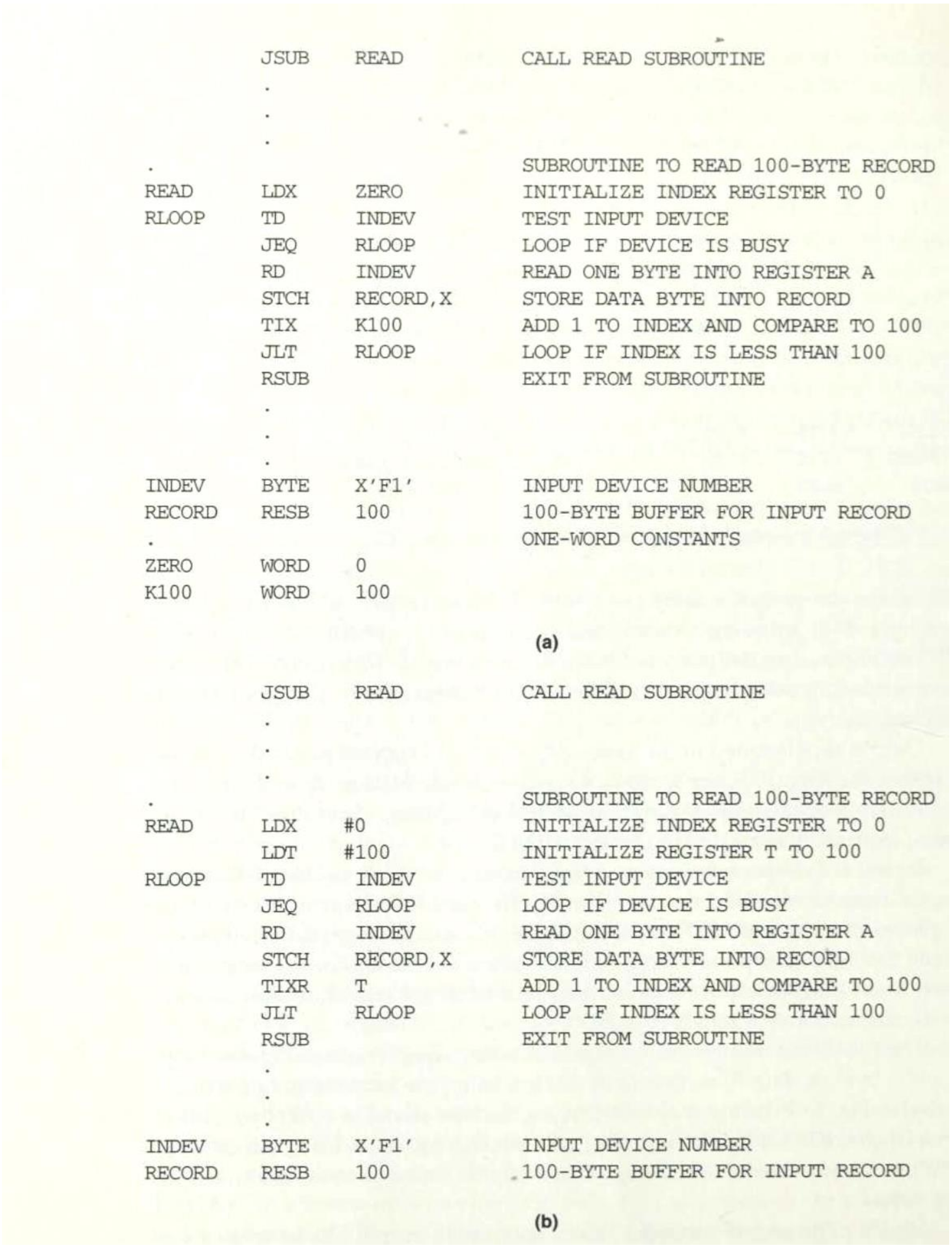
Sample indexing and looping for (a)SIC (b) SIC/XE

ALPHA , BETA and GAMMA are arrays capable of storing 100 words. Add the words in ALPHA and BETA and store it in the respective position in GAMMA.



```
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
```

Sample input and output operations for SIC



```
          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
```

(a)

```
          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
```

(b)

Sample subroutine call and record input operations for (a)SIC (b) SIC/XE

Read a 100 byte record from input device F1 and store it in a buffer named RECORD of 100 bytes