MODULE 1

SOFTWARE

- Set of instructions given to the computer.
- We cannot touch and feel it.
- Developed by writing instructions in programming language.
- Operations of computer are controlled via this.
- If damaged or corrupted, back up copy can be installed again.
- Eg:- Antivirus, Microsoft Office Tools.

HARDWARE

- Physical parts of a computer.
- We can touch and feel it.
- Constructed using physical components.
- Operates under control of software.
- If damaged, can be replaced.
- Eg:- Keyboard, Monitor, Mouse

SOFTWARE vs HARDWARE

SOFTWARE	HARDWARE
1. Collection of instructions that tells computer what to do	1. Physical elements of computer
2. Divided in to a. System Software b. Application Software	2. Categories a. Input Devices. b. Output Devices

c. Utility Software	c. Storage Devices
3. Should be installed in to computer	3. Once software is loaded these can be used.
4. Prone to viruses	4. No virus attacks
5. If damaged/ corrupted reinstallation is possible	5. If damaged, can be replaced.
Eg:- Microsoft Office, Adobe	Eg:- Mouse, Monitor, Keyboard

TYPES OF SOFTWARE

1. System Software:
    - Contains collection of programs that support operation of computer.
    - Helps to run computer hardware and computer system.
    - Handles running of computer hardware.
    - These are of different types"
        a) Operating System
        b) Language Translators
            i. Compiler
            ii. Assembler
            iii. Interpreter
            iv. Macro Processor
        c) Loader
        d) Linker
        e) Debugger
        f) Text Editor

2. Application Software:
    - It allows end users to accomplish one or more specific tasks.
    - Focus on application or problem to be solved.

Operating System

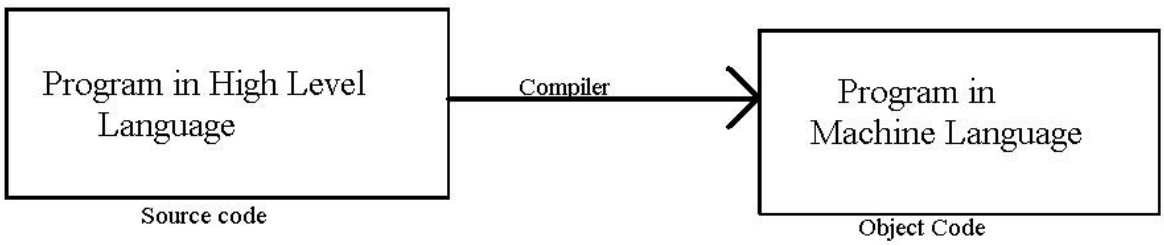
- Acts as interface between user and system.
- Provide user friendly interface.
- Functions:
    a) Process Management
    b) Memory Management
    c) Resource Management
    d) I/O Operations
    e) Data Management
    f) Provide Security for job.

Language Translators

- Program that takes input program in one language and produces an output in another language.

I. Compilers

- Translates program in high level language in to machine level language.
- Conversion or translation is taking place by taking program as whole.
- Bridges the semantic gap between language domain and execution domain.
- Perform syntax analysis, semantics analysis and intermediate code generation.



II. Interpreters

- Translates statement of high level language in to machine level language by taking the program line by line.
- Interpretation cycle includes:

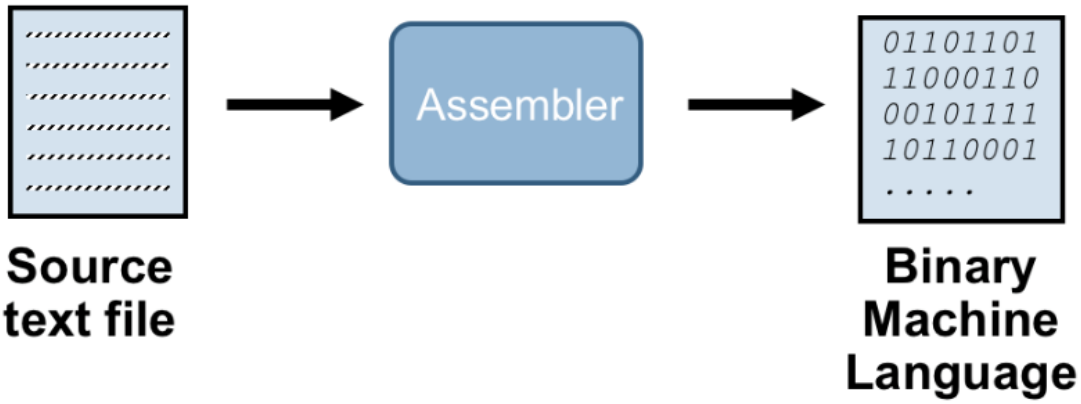
  i) Fetch the statement.

  ii) Analyze the statement and determine its meaning.

  iii) Execute the meaning of statement.

III. Assemblers

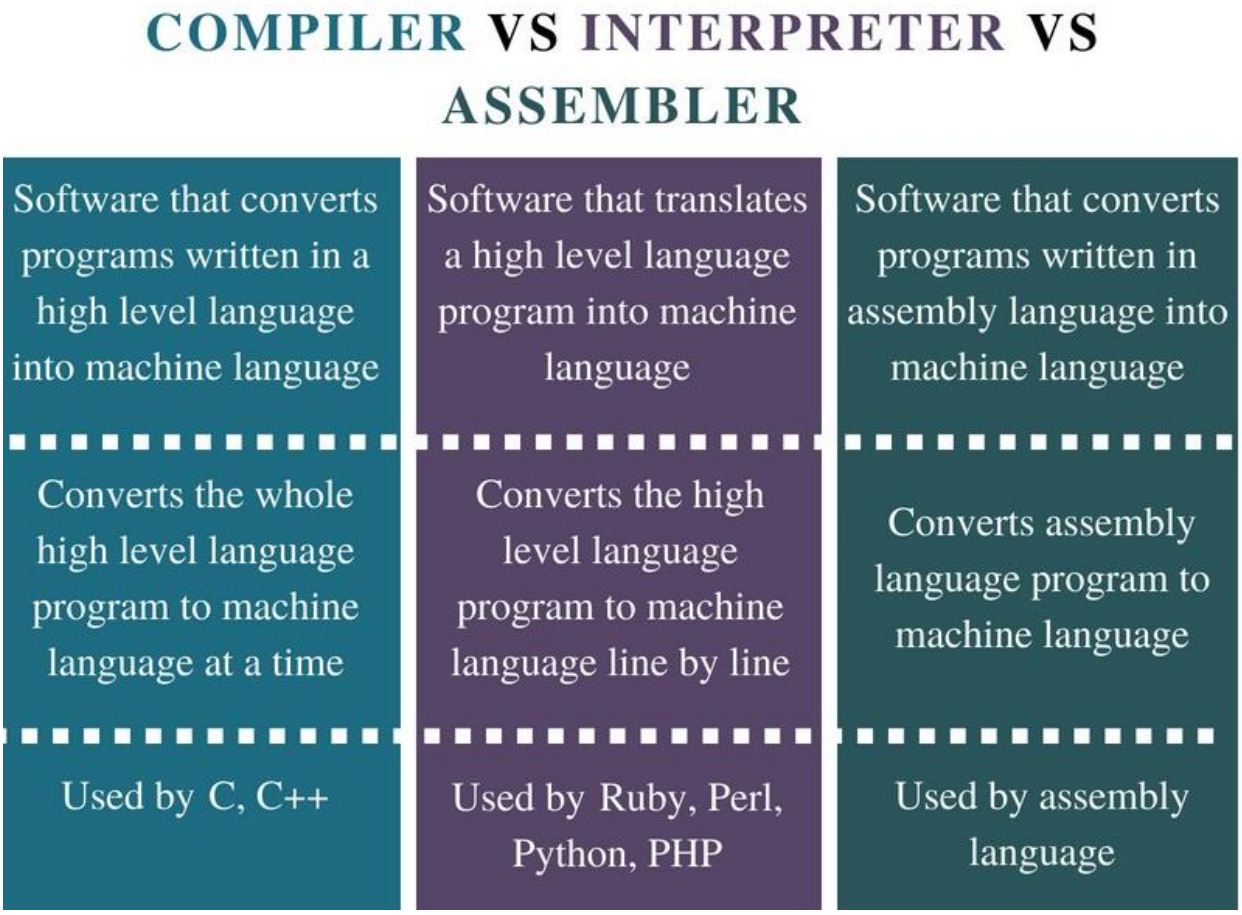
- Programmers found it difficult to read or write programs in machine language, so for convenience they used mnemonic symbols for each instruction which is translated to machine language.
- Assemblers translate assembly language to machine language.
- Translate mnemonic code to machine language equivalents.
- Assign machine address to symbol table.



Working:

- Find the required information to perform task.
- Analyze and design suitable data structures to hold and manipulate information.
- Find the process or steps needed to gather information and maintain it.
- Determine processing step required to execute each identified task.

COMPILER vs INTERPRETER vs ASSEMBLER



Linker

- Process of collecting and combining various pieces of code and data in to single file that can be loaded in to memory and executed.
- Linking performed a compile time, when source code is translated to machine code, at load time, when program is loaded in to memory and executed by loader and at run time by application programs.

Types:

a) Linking Loader: Performs all linking and relocation operations directly in to main memory for execution.

b) Linkage Editor: Produce a linked version of program called as load module or executable image. This load module is written in to file or library for later execution.

c) Dynamic Linker: This linking postpones the linking function until execution time. Also called as dynamic loading.

Loader

- Utility of an operating system.
- Copies program from a storage device to computer's main memory.
- They can replace virtual address with real address.
- They are invisible to user.

Debugger

- An Interactive debugging system provides programmers with facilities that aid in testing and debugging of programs.
- Debugging means locating bugs or faults in program.
- Helps in fixing error.
- Determination of exact nature and location of error in the program.

Device Driver

- It is a software module which manages the communication and control of specific I/O device on type of device.
- Convert logical requests from the user in to specific commands directed to device itself.

Macro Processor

- Macro is the unit of specification of program generation through expansion.
- Macros are special code fragments that are defined once in the program and used by calling them from various places within the program.
- Macro processor is a program that copies stream of text from one place to another, making a systematic set of replacements as it does so.
- They are often embedded in other programs such as assemblers and compilers.

- Before you can use a macro, you must *define* it explicitly with the `#define' directive. `#define' is followed by the name of the macro and then the code it should be an abbreviation for. For example,

```
#define BUFFER_SIZE 1020
```

  defines a macro named `BUFFER_SIZE' as an abbreviation for the text `1020'

Text Editors

- Program that allows the user to create the source program in the form of text in to the main memory.
- Creation, edition, deletion, updating of document or files can be done with the help of text editor.

SIMPLIFIED INSTRUCTIONAL COMPUTER (SIC)

- It is a hypothetical computer that has hardware features which are found in real machines.
- To versions:

  a). SIC Standard Model

  b). SIC/XE (Extra Equipment)

Machine Dependent features of Software System:

1. Assembler: Instruction format, Addressing mode.
2. Compiler: Registers, Machine Instructions.
3. OS: All resources of computing system.

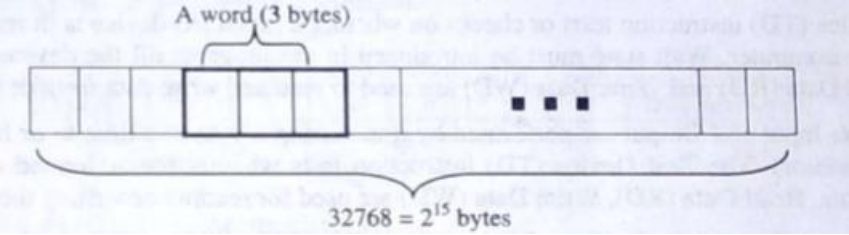
Machine Independent features of Software System:

1. General design and logic of assembler.
2. Code optimization in compiler
3. Linking independently assembled subprogram

SIC ARCHITECTURE- STANDARD MODEL

- It has basic addressing, storing most memory addresses in hexadecimal integer format.
- Its machine architecture includes

1. Memory: There are 2^{15} bytes in the computer memory that is 32768 bytes.



2. Register:
   - Used as storage locations that perform some functions.
   - There are 5 registers each of them is of 24 bits length.

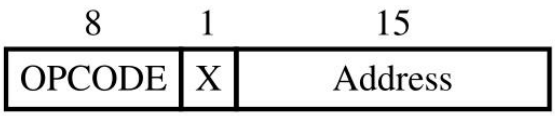
Five Registers

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)

3. Data Formats:
   - It supports only the Integer and Character data formats.

- There is no hardware support for floating point numbers.
- Integers stored as 24 bit binary numbers.
- Negative values represented as 2's complement.
- Character data stored as 8 bit ASCII codes.

4. Instruction Formats:
   - All machine instructions in the standard version of SIC have the following 24 bit format:



   - Flag bit x is used to indicate the indexed addressing mode.

5. Addressing mode: 2 Types
   a) Direct Addressing Mode: Here flag bit x=0

      Target Address= Actual Address

   b) Indexed Addressing Mode: Here flag bit x=1

      Target Address= Actual Address+Index Register (X) contents

      i.e. **Target Address= Address+(X)**

6. Instruction Set:
   a. Data Transfer Instruction: Include instructions that load and store register.

      Eg: LDA, STA, LDX, STX

   b. Arithmetic Operation Instruction: Arithmetic operations can be done which involves register A

      Eg: ADD, SUB, MUL, DIV, COMPR

   c. Conditional Branching Instruction: The conditional jump instruction test the setting of condition code and jumps.

      Eg: JLT, JEQ, JGT

   d. Subroutine Call Instruction: Two instructions are provided to perform subroutine linkage

      i) JSUB: To jump

ii) RSUB: To return

e. Input and Output Instruction:

- I/O operations are executed by transferring a single byte each time.
- Target port is specified by last 8 bits of register A.
- Each device is assigned a unique 8 bit code to send and receive data and control signals.

7. Input and Output:

- Performed by transferring 1 byte at a time to or from right most 8 bits of register A (Accumulator).
- Test Device (TD) instruction tests whether the addressed device is ready to send and receive a byte of data.
- Read Data (RD) and Write Data (WD) is used for reading and writing of data.

8. Data Movement and Storage Definitions:

- LDA, STA, LDX, STX all uses 3 byte word.
- LDCH, STCH are associated with characters which uses 1 byte.
- Storage definitions are:
  a. WORD- ONE WORD CONSTANT
  b. RESW- ONE WORD VARIABLE
  c. BYTE- ONE BYTE CONSTANT
  d. RESB- ONE BYTE VARIABLE

SIC/XE ARCHITECTURE- SIC WITH EXTRA EQUIPMENT

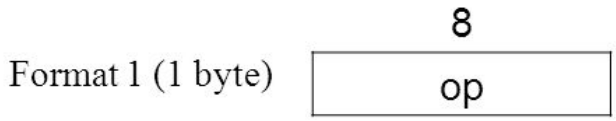
- Architecture is similar to standard model with certain additional components and features.

1. Memory: Maximum memory available on a SIC/XE system is 1MB (2^{20} bytes)
2. Registers: Additional B, S, T and F registers are provided by SIC/XE , in addition to the registers of SIC.

Mnemonic	Number	Special use
B	3	Base register
S	4	General working register
T	5	General working register
F	6	Floating-point accumulator (48 bits)

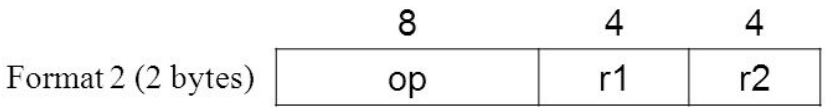
3. Floating point Data type: There is a 48 bit floating point data type, $F*2^{(e-1024)}$

4. Instruction format: New set of instruction formats for SIC/XE are as follows:
   a. Format 1 (1 Byte): Contains only operation code



   Eg: RSUB (Return to Subroutine)

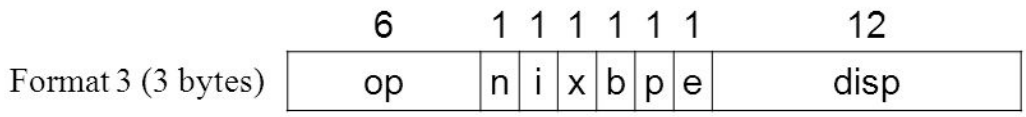
   b. Format 2 (2 Bytes): First 8 bits for operation code, next four for register 1 and following for register 2.



   Eg: COMPR A, S (Compare contents of register A and S)

   c. Format 3 (3 Bytes) : Here e=0
      - First 6 bits contain operation code.
      - Next 6 bits contain flags.
      - Last 12 bits contain displacement for the address of the operand.

- Flags are in order -n, i, x, b, p, e.
- e indicates instruction format.
- Bits i and n are used for target address calculation



Eg: LDA #3 (Load 3 to Accumulator A)

Format 3 has many cases:

i. If i=0, n=1, word given by target address is fetched and value in word is taken as address of operand value- Indirect Addressing (Prefix #).

ii. If i=1, n=0, target address is used as operand value.

Also called Immediate Addressing mode (Prefix #)

a) Case 1: Value contained location in word=operand value

Eg: ADD X, [500]

Here word in location 500 is fetched .

It gives address of first operand, second operand is given in indirect addressing mode.

b) Case 2: Target Address= Operand Value

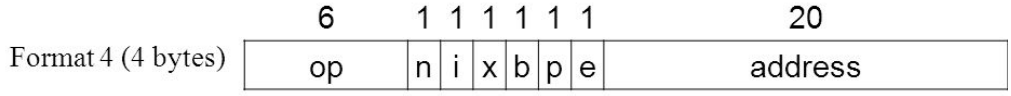
Eg:- If TA=10, Operand Value =10

iii. If i=0, n=0 or i=1, n=1 target address is the location of operand. Also called as Simple Addressing.

TA=location of operand

d. Format 4 (4 bytes): Here e=1

- It is same as format 3 with an extra 2 hex digits for address that require more than 12 bits to be represented.



5. Addressing mode and Flag bits:

a. Direct (x,b and p All set to 0):

- Operand address goes as it is.
- n and i are both set to the same value, either 0 or 1.

b. Relative (Either b or p equal to 1 and the other one to 0): Address of operand should be added to the current value stored at the B register (if b=1) or to the value stored at the PC register (if p=1)

c. Immediate (i=1,n=0): The operand value is already enclosed on the instruction.

d. Indirect (i=0, n=1): The operand value points to an address that holds the address for operand value.

e. Indexed (x=1):

- Value to be added to the value stored at the register x to obtain real address of operand.
- Can be combined with any of previous mode except immediate.

Indexing is not possible with immediate or indirect addressing mode.

Two relative addressing modes are:

i) Base relative addressing mode.

ii) Program counter relative addressing mode.

Mode	Indication	Target Address Calculation
Base Relative Addressing Mode	b = 1 P = 0	**TA = Displacement + (B)** B – Base Register Displacement is 12 bit unsigned register. Displacement lies between 0 to 4095
Program Counter Relative Addressing Mode	b = 0 P = 1	**TA = Displacement + (PC)** PC – program counter Displacement is 12 bit signed integer. Displacement lies between – 2048 to 2047.
Direct Addressing Mode	b = 0, P = 0 (Format 4 instruction) b = 0, P = 0 (Format 3 instruction)	**TA = address field of format 4 instruction** TA = Displacement field value of format 3 instruction
Base Relative Indexed Addressing Mode	b = 1, P = 0 X = 1	**TA = Displacement + (B) + (X)** B – Base register X – Index register Displacement is 12 bit unsigned register. Displacement lies between 0 to 4095
Program Counter Relative Indexed Addressing Mode	b = 0, P = 1 X = 1	**TA = Displacement + (PC) + (X)** PC – program counter X – Index Register Displacement is 12 bit signed integer. Displacement lies between – 2048 to 2047

6. Instruction set:

i. Instruction that load and store new register ‘B’:

a. LDB- Load the register ‘B’ with some value.

   Eg: LDBx- Load value of x in to register B.

b. STB- Store the register ‘B’ content in to some variable.

   Eg: STBx- Store register ‘B’ content in to variable x.

ii. Instruction those perform floating point Arithmetic operation

a. ADDF

b. SUBF

c. MULF

d. DIVF

   Here F is the floating point register

   Eg: ADDF, here register’ B’ contents are added with Accumulator content and result is left with accumulator.

iii. Instruction that take operand from Register

   RMO-Register move

   Eg: RMO S,B Register ‘S’ content is moved to ‘B’ register.

iv. Instruction which perform register arithmetic operation

a. ADDR

b. SUBR

c. MULTR

d. DIVR

   Eg: ADDR S,B

   add value of register B with register Sand store result in register B.

7. Input and Output:

- The SIC/XE supports all the I/O instructions in the standard version.
- There are special I/O channels which are utilized for data transfer when CPU is involved in another process at same time.
- Channels control associated I/O channels.
- There can be maximum of 16 I/O channels each supporting maximum of 16 devices.

- RD and WD is used to read and write data from or to specified I/O devices.

SIO	Instruction is used to Start an I/O Channel number
TIO	Instruction is used to Test an I/O Channel number
HIO	Instruction is used to Halt an I/O Channel number

SIC vs SIC/XE

Basis	SIC	SIC/XE
Registers	Only 5 registers are used, which are A, X, L, SW and PC.	There are 9 registers are used, which are A, X, L, SW, PC, B, T and F.
Floating Point Hardware	There is no floating point hardware.	Floating point hardware is used.
Instruction Format	Only one instruction format is used.	There are four different types of instructions format.
Addressing Modes	There are two addressing modes.	There are many more addressing modes.

Also refer the pdf (Comparison SIC and SIC XE)

ASSEMBLER DIRECTIVES

- Pseudo instructions.
- Provide instruction to assembler itself
- They are not translated in to machine operation code.
- SIC and SIC/XE has following assemble directives:

  START- Specify name and starting address of the program

  END- Indicate end of the source program and specify first executable statement in program

  BYTE- Generate character or hexadecimal constant.

  WORD- Generate one word integer constant.

  RESB- Reserves the indicated number of bytes for data area.

  RESW- Reserve the indicated number of words for data area.

Data movement in SIC and SIC/XE

1. Data Movement in SIC

	LDA	**EIGHT**	**load constant 8 in to the register A**
	STA	**FIRST**	**store in FIRST**
	LDCH	**CHARZ**	**load character 'Z' in to register A**
	STCH	**C1**	**store in character variable C1**
	.		
	.		
	.		
FIRST	**RESW**	**1**	**One word variable**
EIGHT	**WORD**	**8**	**One word constant**
CHARZ	**BYTE**	**C'Z'**	**One byte constant**
C1	**RESB**	**1**	**One byte variable**

Note, In SIC:

- RESB and RESW is used for variables
- BYTE and WORD is used for values
- RESB is used for variable for eg: C1
- RESW is used for variables represented using words For eg: FIRST, it is a variable name represented in form of letter/ word. C can be another example which uses the assembler directive RESW
- BYTE is used for character values/constants for eg: char Z
- WORD is used for values expressed in word form, for eg: EIGHT represents value 8 in word form

2. Data Movement in SIC/XE

- Here immediate addressing scheme is used.

	LDA	**#8**	**load value 8 in to the register A**
	STA	**FIRST**	**store in FIRST**
	LDCH	**#90**	**load ASCI code of 'Z' in to register A**
	STCH	**C1**	**store in character variable C1**
	.		
	.		
	.		
FIRST	**RESW**	**1**	**One word variable**
C1	**RESB**	**1**	**One byte variable**

Note, In SIX/XE:

- The values are represented with a prefix # and in numerical form , eg: #8
- Character values are represented using their ASCII values, eg: for Z we used its ASCII value 90

Arithmetic Operations in SIC and SIC/XE

1. In SIC

```
        LDA     FIRST     load FIRST into register A
        ADD     INCR      add value of INCR
        SUB     ONE       subtract 1
        STA     SECOND    store in SECOND
        LDA     THIRD     load THIRD into register A
        ADD     INCR      add value of INCR
        SUB     ONE       subtract 1
        STA     FOURTH    store in FOURTH
        .
        .
        .
FIRST   RESW    1         One word variable
ONE     WORD    1         One word Constant
SECOND  RESW    1         One word variable
THIRD   RESW    1         One word variable
FOURTH  RESW    1         One word variable
INCR    RESW    1         One word variable
```

2. In SIC/XE

```
        LDS     INCR    load value of INCR in to the register S
        LDA     FIRST   load FIRST into register A
        ADDR    S, A    add value of INCR
        SUB     #1      subtract 1
        STA     SECOND  store in SECOND
        LDA     THIRD   load THIRD into register A
        ADDR    S, A    add value of INCR
        SUB     #1      subtract 1
        STA     FOURTH  store in FOURTH
        .
        .
        .
FIRST   RESW    1       One word variable
SECOND  RESW    1       One word variable
THIRD   RESW    1       One word variable
FOURTH  RESW    1       One word variable
INCR    RESW    1       One word variable
```

Input/ Output Operations in SIC and SIC/XE

1. In SIC

```
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     load 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
```

2. In SIC/XE

```
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     load 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
```