System Programming

Chapter One: Introduction

Haymanot F.(MSC)

Debre markos university
School of computing
Department of Software Engineering

2013 E.C.

CONTENTS

- Basic terminology
- System software and machine architecture
- The Simplified Instructional Computer (SIC)
  - SIC and SIC/XE Machine architecture
  - Data and instruction formats
  - Addressing modes
  - Instruction sets
  - I/O
  - Programming Examples

Basic terminology

- System: Group of related things that works towards common goal.
  - An organized collection of parts(sub systems)are integrated together to accomplish goal
  - E.g. computer(made from integrated components like I/O, cpu , storage etc.)
- Program/Software: executable sw runs on a machine like computer
  - Consists a compiled code that can run directly from the computer's operating system.
- Types of software: there are two types
  - Application SW
  - System SW

CON…

Application Software

- A set of programs written in a specific programming language to solve a particular problem.
- It is independent of the machine on which it is operated.
- Concerned mainly with the solution of the problem and makes use of the computer software as a tool.
- Purpose of supporting or improving the user's work with a software (it gives services for user)
- The focus is on the application

- **Example:**
  - Word processing :MS word, WordPad
  - Database :Oracle, MS Access
  - Spreadsheet :Excel, Apple Numbers
  - Multimedia :Real Player, Media Player
  - Presentation Graphics :MS PowerPoint

CON…
System Software

- It is software designed to provide a platform for other software.
  - A set of programs that creates the environment to facilitate working of an application software.
  - Able effective execution of application Software
  - Acts as an intermediary between computer hardware and application programs.
  - Controls the computer system and enhances its performance.
  - Focus is on the system.
  - Machine dependent(Instruction Set, Instruction Format, Addressing Mode)
  - However , general design and logic of an assembler and compiler, code optimization techniques are machine independent
- System programming means development of computer programs that allow the entire system to function as a single unit.

Examples of System Software

- Operating **systems:** like macOS, Linux, Android and Microsoft Windows, computational science **software**, game engines, industrial automation,
- Text Editor
  - Allows user to create/modify a file having only plain text.
  - Edit contents to get an immediate visual feedback
  - Notepad (Microsoft)
- Compiler
  - Translates high level language to machine language.
  - Source code into data, that computer can understand.
  - Source code to object code
    - ADA, BASIC, Fortran, Pascal
    - C, C++, C#
    - Open Source like java
- Linker
  - Gathers 1 or more objects codes generated by the compiler & combines into a single EXE

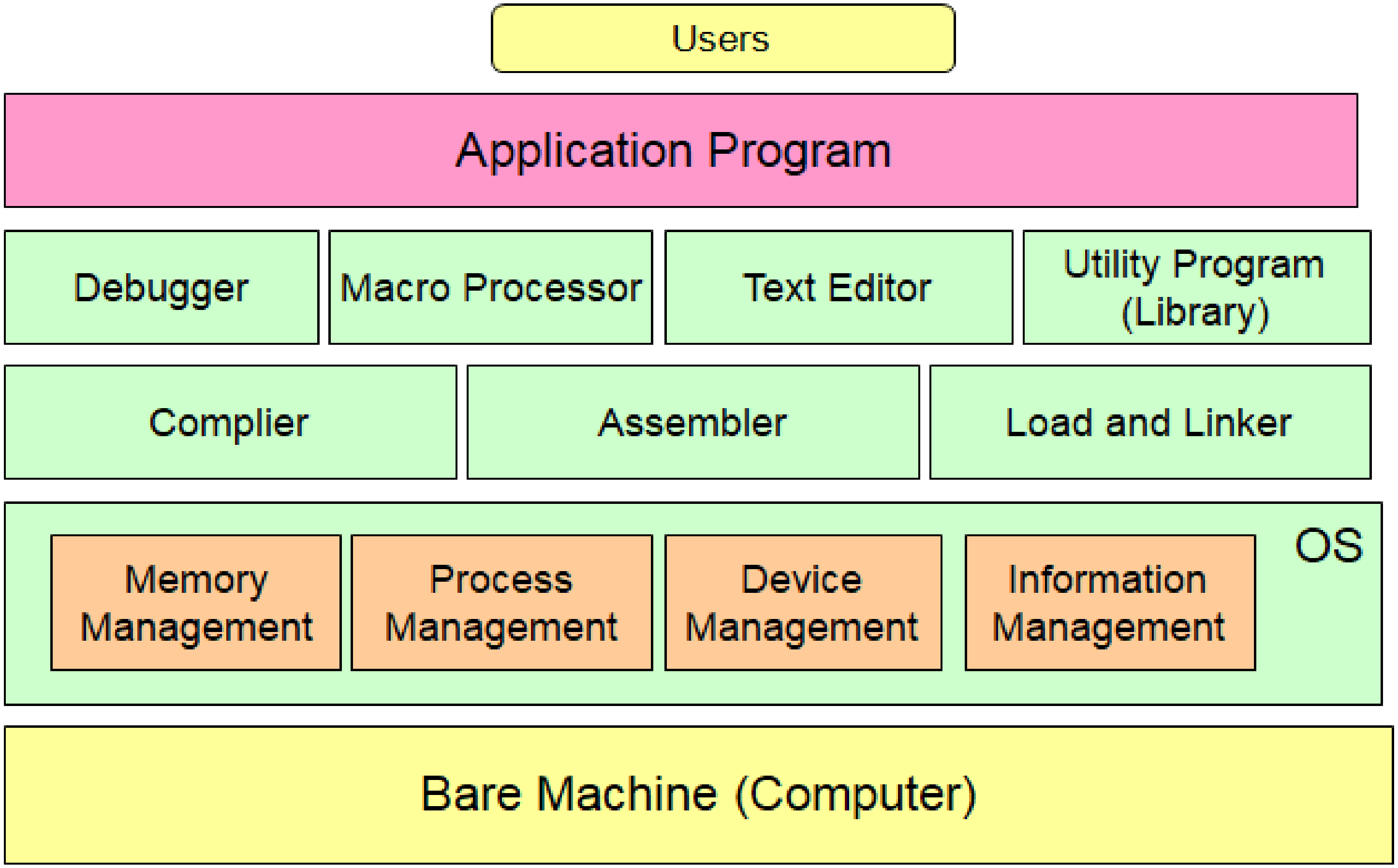
CON…

- Loader
  - Loads object code into memory for execution
- Debugger
  - Tests & debugs errors
  - Code run on Instruction Set Simulator (ISS)
    - DEBUG – Microsoft DOS built-in debugger
    - Nemiver – Graphical C/C++ debugger
    - Ups – Fortran, C
    - VB Watch – Visual Basic
    - Jswat – Open source Java
    - Xdebug – PHP
- Assembler
  - Assembly language to machine language
  - Mnemonics to machine code , Produces executable machine code
- Interpreter
  - High level language to machine language

Need for System Software

- Controls some of the aspects of operation of computers.
  - File operation
  - I/O operation
  - Memory management
  - Etc.

SYSTEM SOFTWARE CONCEPT



System software and machine architecture

- One characteristic in which most system_software differ from application_software is **machine dependency.**
  - e.g. assembler translate mnemonic instructions into machine code
  - e.g. compilers must generate machine language code
  - e.g. operating systems are directly concerned with the management of nearly all of the resources of a computing system
- Some aspects of system software that do not directly depend upon the type of computing system
  - e.g. general design and logic of an assembler and compiler
  - e.g. code optimization techniques

The Simplified Instructional Computer (SIC)

- **SIC** is a hypothetical computer system introduced in *System Software.*
  - It includes the hardware features most often found on real machines while avoiding unusual or irrelevant complexities
  - Since it is difficult to learn system programming on real computer system SIC introduced.
- **Two versions of SIC**
  - SIC - standard model
  - SIC/XE (extra equipments or extra expensive)
    - SIC program can be executed on SIC/XE (upward compatible)
- The two versions has been designed to be upward compatible.
      - Object program for the standard SIC machine will execute properly on SIC/XE system.

SIC Machine Architecture

- Memory
  - Memory consists of 8-bit , 3 consecutive bytes form a word (24 bits), SIC is designed as a 24-bit machine. Total of 32768 bytes (32 KB) in the computer memory.
  - A word is addressed by is lowest numbered byte (i.e., addressing starts at byte 0).
- Register: SIC machines have 5 registers

Register	Number	Known as	Use
A	0	Accumulator	Arithmetic operations
X	1	Index register	Addressing
L	2	Linkage register	jumping to specific memory addresses and storing return addresses
PC	8	Program counter	Contains the address of the next instruction to execute
SW	9	Status word	Contains a variety of information, such as carry or overflow flags

- Register : is a temporary storage area built into a CPU, **registers** are used to pass data from memory to the processor
- Flags : is a value that acts as a signal for a function or process

SIC Machine Architecture …

- **Data Formats**
  - Integers are stored as 24-bit binary numbers
  - 2's complement for negative values
  - 8-bit ASCII for characters
  - No floating point hardware in standard SIC
- **Instruction Formats**
  - 24-bit format
  - x indicates indexed addressing mode

8	1	15
opcode	**X**	**address**

SIC Machine Architecture …

- **Addressing modes**
  - 2 addressing modes available
    - Indicated by x bit in the instruction

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

    - (x) represents the contents of register x

SIC MACHINE ARCHITECTURE …

- **Instruction Set**
  - **load and store**: To move or store data from accumulator to memory or vice-versa example : LDA, LDX, STA, STX …
  - **Arithmetic operations**: ADD, SUB, MUL, DIV, etc.
    - Used to perform operations on accumulator and memory and store result in accumulator (register A)
  - **comparison**: COMP
    - compares the value in register A with a word in memory, this instruction sets a condition code CC to indicate the result
  - **conditional jump instructions**: JLT, JEQ, JGT
    - these instructions test the setting of condition code CC and jump accordingly
  - **subroutine linkage**: 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
    - a **subroutine** is a sequence of program instructions that performs a specific task

SIC MACHINE ARCHITECTURE …

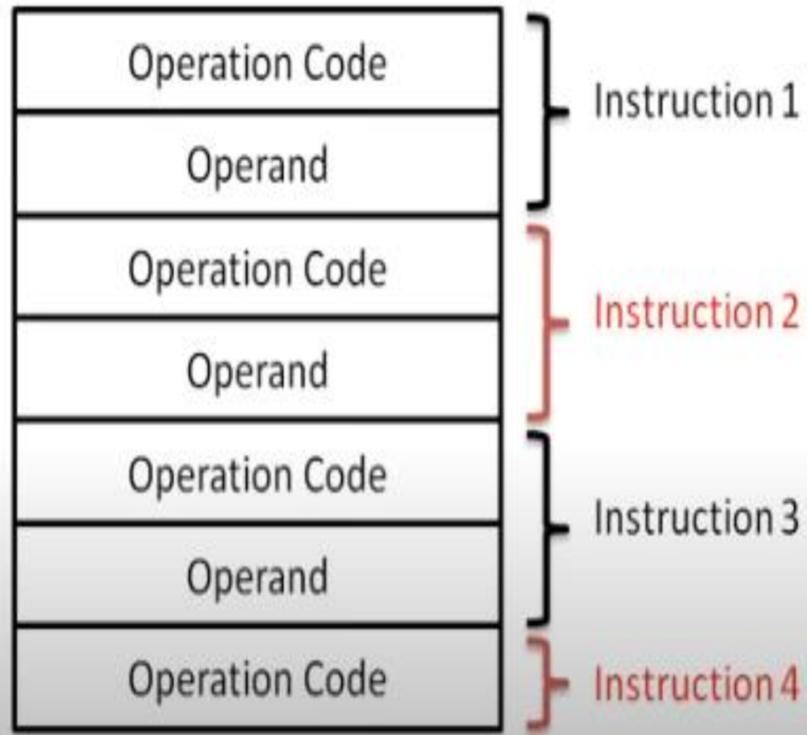
- **Input and Output**
  - Input and output are performed by transferring 1 byte at a time to or from the rightmost 8 bits of register A
  - **The Test Device** (TD): instruction tests whether the addressed device is ready to send or receive a byte of data
  - *"Less Than " if device is ready; "Equal" if device is busy.*
  - **Read Data (RD)**
    - read a byte from the device to register A
  - **Write Data (WD)**
    - write a byte from register A to the device

MACHINE CODE INSTRUCTION

- **Machine code** is a set of **instructions** executed directly by a **computer's** central processing unit
- Each **instruction** performs a very specific task, such as a load, a jump, or an ALU operation on a unit of data in a CPU register or memory.
- Machine code instruction usually consists operation code(op code) &operands
- Some instructions consists only operation code
- Op code is a mnemonic used to refer to a microprocessor instruction in assembly language
- Operand is A quantity to which an operator is applied (in 3 - x, the operands of the subtraction operator are 3 and x).
- All operation code and operands are binary code

Con…

A machine code program consists of a sequence of Op Codes and Operands stored in the computers memory (e.g. RAM)



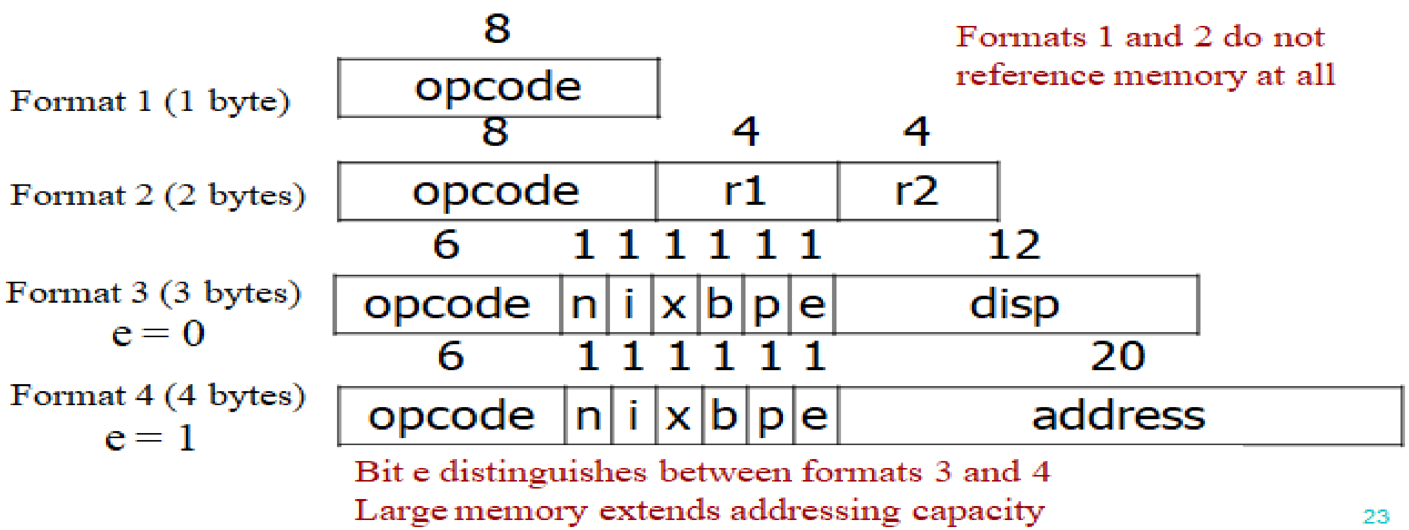
SIC/XE Machine Architecture

- Memory
  - Larger Memory
    - 2^{20} bytes (1 megabyte) in the computer memory
    - Leads to change in instruction formats & addressing modes
- Registers
  - Additional registers provided

Mnemonic	Number	Use
B	3	Base register – used for addressing
S	4	General purpose register – no special use
T	5	General purpose register – no special use
F	6	Floating point accumulator (48 bits instead 24 bits)

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
  - value represented = $f*2^{(exponent-1024)}$, f is fraction b/n 0 &1
- Instruction format



SIC/XE Machine Architecture…

- **Format 1**: Consists of 8 bits of allocated memory to store instructions. Don't refer memory location
- **Format 2**: Consists of 16 bits of allocated memory to store 8 bits of instructions and two 4-bits segments to store operands(source and destination register)
- **Format 3**: Consists of 6 bits to store an instruction, 6 bits of flag values, and 12 bits of displacement.
- **Format 4**: Only valid on SIC/XE machines, consists of the same elements as format 3, but instead of a 12-bit displacement, stores a 20-bit address.
- Both format 3 and format 4 have six-bit flag values in them, consisting of the following flag bits:
  - **n**: Indirect addressing flag
  - **i**: Immediate addressing flag
  - **x**: Indexed addressing flag
  - **b**: Base address-relative flag
  - **p**: Program counter-relative flag
  - **e**: Extended for Format 4 instruction flag

SIC/XE Machine Architecture…

- Addressing modes
  - Base relative (n=1, i=1, b=1, p=0)
  - Program-counter relative (n=1, i=1, b=0, p=1)
  - Direct (n=1, i=1, b=0, p=0)
  - Immediate (n=0, i=1, x=0)
  - Indirect (n=1, i=0, x=0)
  - Indexing (both n & i = 0(SIC) or 1(SIC/EX), x=1)
  - Extended (e=1) for format 4 and e=0 for format 3
    - Note: Indexing cannot be used with immediate or indirect addressing.

SIC/XE MACHINE…

- Base Relative Addressing Mode

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

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

- Program-Counter Relative Addressing Mode

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

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

SIC/XE MACHINE…

- Direct Addressing Mode

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

n=1, i=1, b=0, p=0 TA=disp $(0 \leq disp \leq 4095)$

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

n=1, i=1, b=0, p=0 TA=(X)+disp

(with index addressing mode)

SIC/XE Machine…

- Immediate Addressing Mode

	n	i	x	b	p	e	
opcode	0	1	0				disp

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

- Indirect Addressing Mode

	n	i	x	b	p	e	
opcode	1	0	0				disp

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

CON..

- PC-relative simple addressing: **(PC) + disp**
- Base-relative indexed simple addressing: **(B) + disp + (X)**
- PC-relative indirect addressing: **(PC) + disp**
- Immediate addressing: **disp**
- Indirect addressing :**disp**

SIC/XE MACHINE ARCHITECTURE…

- Instruction Set
  - new registers: LDB, STB, etc.
  - floating-point arithmetic: ADDF, SUBF, MULF, DIVF
  - register move: RMO
  - register-register arithmetic: ADDR, SUBR, MULR, DIVR
  - supervisor call: SVC
    - generates an interrupt for OS
- Input/output
  - SIO, TIO, HIO: start, test, halt the operation of I/O device

SIC Assembly Syntax

- SIC uses a special assembly language with its own operation codes that hold the hex values needed to assemble and execute programs.
- A sample program is provided below to get an idea of what a SIC program might look like.
- The first column represents a forwarded symbol that will store its location in memory.
- The second column denotes either a SIC instruction (opcode) or a constant value (BYTE or WORD).
- The third column takes the symbol value obtained by going through the first column and uses it to run the operation specified in the second column.
- This process creates an object code, and all the object codes are put into an object file to be run by the SIC machine.

SIC and SIC/XE Programming Examples

- Data movement operations
- Arithmetic operations
- Looping and indexing operations
- Data input and output operations

Programming Examples

WORD – Integer constant

BYTE – Character constant

```
         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)

RESW - Reserve indicated number of words for a data area

RESB - Reserve indicated number of bytes for a data area

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

Programming Examples

```
         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
```

BETA ⟵ (ALPHA + INCR - 1)
DELTA ⟵ (GAMMA + INCR - 1)

Fig: Arithmetic operations SIC

SIC Programming Example loop

```
        LDA     ZERO        initialize index value to 0
        STA     INDEX
ADDLP   LDX     INDEX       load index value to reg X
        LDA     ALPHA,X     load word from ALPHA into reg A
        ADD     BETA,X
        STA     GAMMA,X     store the result in a word in GAMMA
        LDA     INDEX
        ADD     THREE       add 3 to index value
        STA     INDEX
        COMP    K300        compare new index value to 300
        JLT     ADDLP       loop if less than 300
        ...
        ...
INDEX   RESW    1
ALPHA   RESW    100         array variables—100 words each
BETA    RESW    100
GAMMA   RESW    100
ZERO    WORD    0           one-word constants
THREE   WORD    3
K300    WORD    300
```

Gamma[] = Alpha[] + Beta[]

SIC Programming Example I/O

- Input and output

```
INLOOP    TD      INDEV     test input device
          JEQ     INLOOP    loop until device is ready
          RD      INDEV     read one byte into register A
          STCH    DATA
          .
          .
OUTLP     TD      OUTDEV    test output device
          JEQ     OUTLP     loop until device is ready
          LDCH    DATA
          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
```

SIC/XE Programming Examples

Arithmetic operation.

BETA ← (ALPHA + INCR - 1)
DELTA ← (GAMMA + INCR - 1)

```
        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
```

This program will execute faster because it need not load INCR from memory each time when INCR is needed.

SIC/EX Programming Examples

Looping and indexing

```
        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
```

This program will execute faster because it uses register-to-register add to reduce memory accesses.

Fig: Looping & indexing SIC/XE

SIC/EX Programming Examples

Input /output

```
         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
```

Fig: Subroutine call & record input operations

Thank you!

Question?