System Software

System Software

- System software is a set of programs that support the operation of a computer
- Hides the complexities of the hardware from the user
- The user can focus on an application or other problem to be solved without needing to know the details of how the machine works internally.
- Controls the operation and extends the processing functionalities of a computer system.
- Makes the operation of a computer more fast, effective, and secure.
- Application Software is a program that does real work for the user. It is mostly created to perform a specific task for a user.

System Software

- Compiler
- Assembler
- Interpreter
- Text Editor
- Debugger
- Macro processor
- Loader
- Linker

Introduction

System Software

- General purpose software
- Intended to support the operation and use of computer
- Related to the architecture of the machine on which they are to run
- Machine dependent

Application Software

- Specific purpose software
- Primarily concerned with the solution of some problem using the computer as a tool
- Focus is on the application, not on the computing system
- Machine independent

System Software Vs Machine Architecture

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

Simplified Instructional Computer(SIC) Machine

- Hypothetical computer that includes the hardware features most often found on real machines
- Two versions:
  - Standard model, SIC
  - XE version (SIC/XE) (extra equipment or extra expensive)
  - Upward compatible
    - An object program for the standard SIC will also execute properly on a SIC/XE system

- **SIC Machine Architecture**
- Memory
  - 8 bit bytes
  - 3 consecutive bytes from a word
  - All addresses are byte addresses
    - words addressed by the location of the lowest numbered byte
  - Address=15 bits
  - Memory size=2^{15} bytes

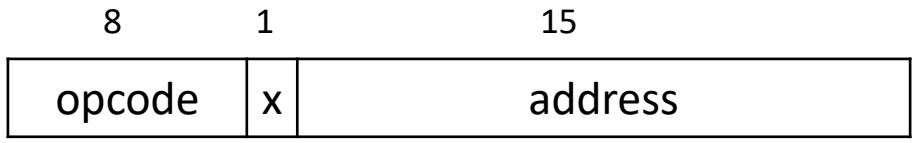
SIC Machine Architecture

- Registers
- Five registers, 24 bit 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)

SIC Machine Architecture

- **Data Formats**
  - Integers stored as 24 bit numbers
  - 2’s complement representation for negative numbers
  - Characters stored using their 8 bit ASCII Code
  - No floating point hardware
- **Instruction Formats**
  - 24 bit format



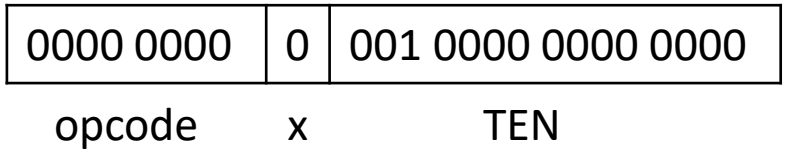
  - Flag bit-x indicates indexed addressing mode

Addressing Modes

- Two addressing modes –direct and indexed

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

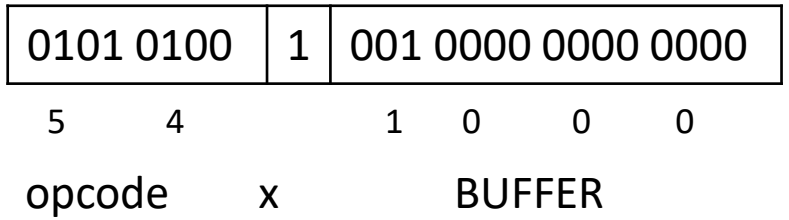
- **Direct addressing mode**
  - LDA TEN



Effective Address (EA) =1000

Addressing Modes

- **Indexed addressing mode**
  - STCH BUFFER, X



Effective Address (EA) =1000 + [X]

Instruction Set

- Instructions that load and store registers
  - LDA, LDX, STA, STX
- Integer arithmetic operations
  - ADD, SUB, MUL, DIV
  - involve register A and a word in memory, result in the register
- Comparison, COMP
  - compares the value in register A with a word in memory and sets the condition code, CC.
- Conditional jump instructions JLT, JEQ, JGT
  - Test the setting of CC
- Subroutine linkage instructions, JSUB and RSUB
  - Return address placed in register L

Input and Output

- Performed by transferring 1 byte at a time to or from the rightmost eight bits of register A
- Each device is assigned a unique 8 bit code.
- Three I/O instructions
- Test Device (TD) checks whether the device is ready to send or receive data
  - Condition code is set to indicate the result of this test
  - < device is ready to send or receive, = device not ready
- Program needing to transfer data must wait until the device is ready, then execute a Read Data (RD) or Write Data (WD).
- Sequence repeated for each byte of data to be read or written

SIC/XE Machine Architecture

- Memory
  - 2^{20} bytes in the computer memory
- Registers
- Additional registers provided by SIC/XE

Mnemonic	Number	Special use
B	3	Base register; used for addressing
S	4	General working register
T	5	General working register
F	6	Floating-point acumulator (48bits)

SIC/XE Machine Architecture

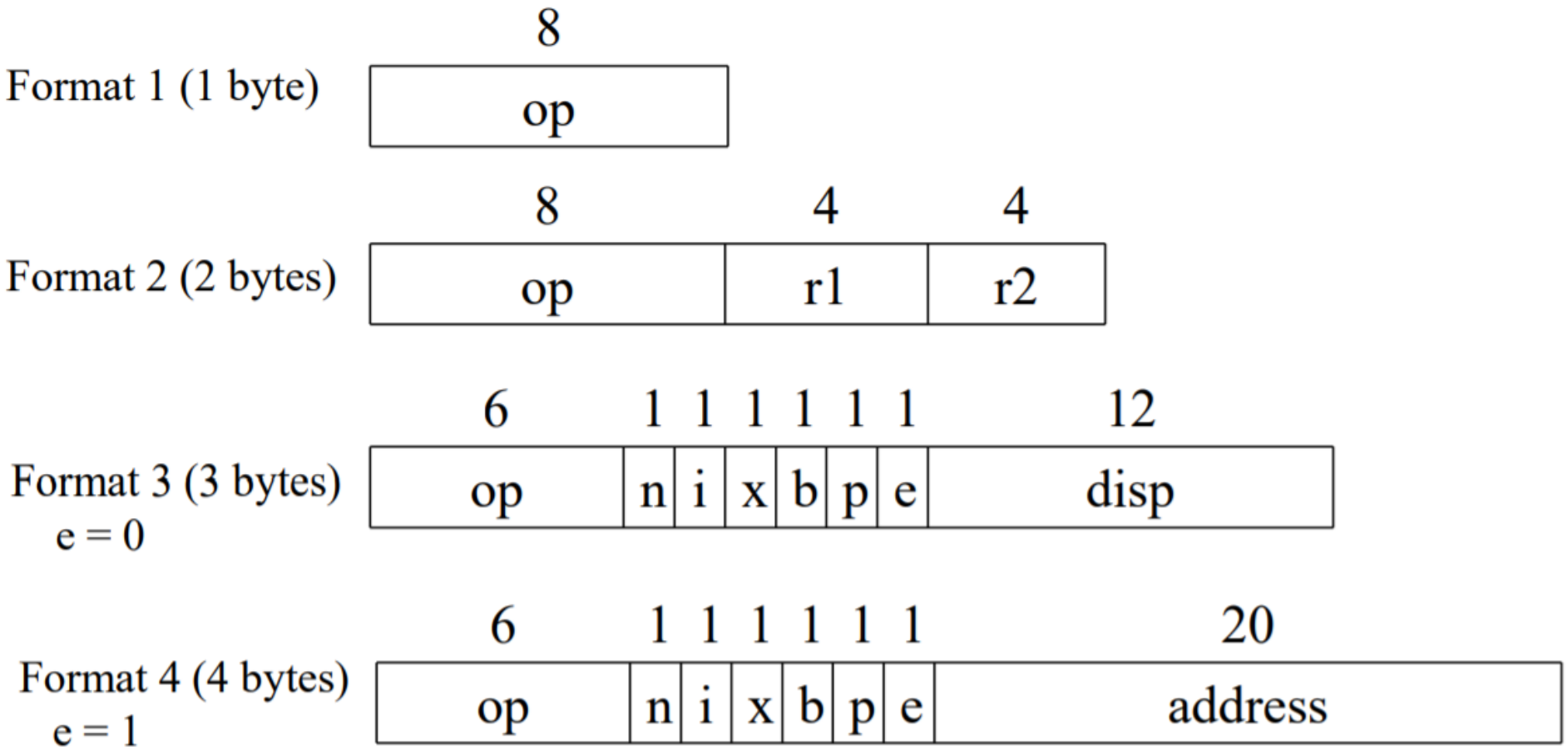
- Data Formats
  - Same data format as the standard version
  - 48 bit floating-point data type

  - Fraction is a value between 0 and 1, the assumed decimal point is immediately before the high order bit
  - For normalized floating point numbers, the high order bit of the fraction must be 1
  - Exponent is an unsigned number between 0 and 2047
  - If the *exponent is e* and the *fraction has a value f*, the absolute value of the number is represented by

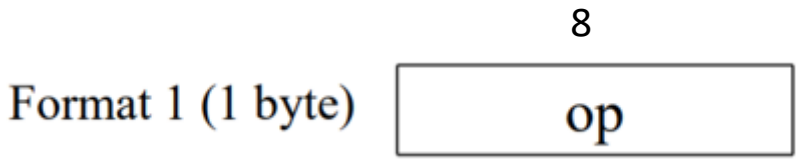
$$f*2^{(e-1024)}$$

- s-represents the sign of the floating point number, s=0 positive, s=1 negative
- To represent zero, set all the bits to zero

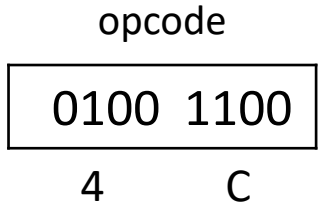
Instruction formats

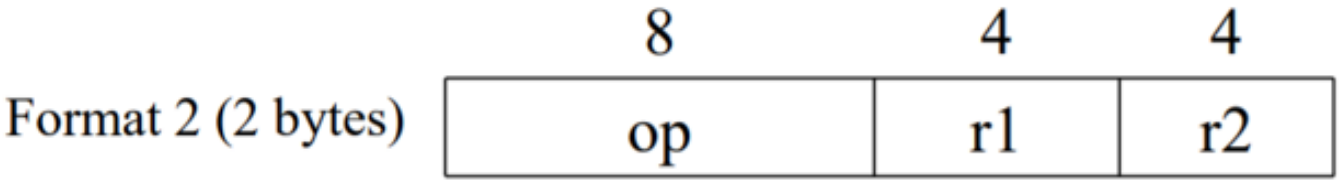


Instruction Format

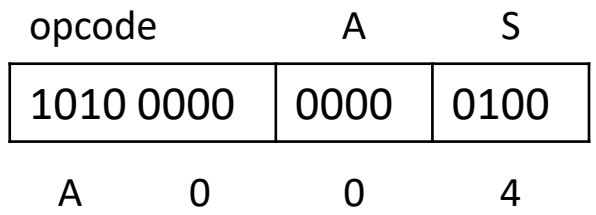


Eg: RSUB



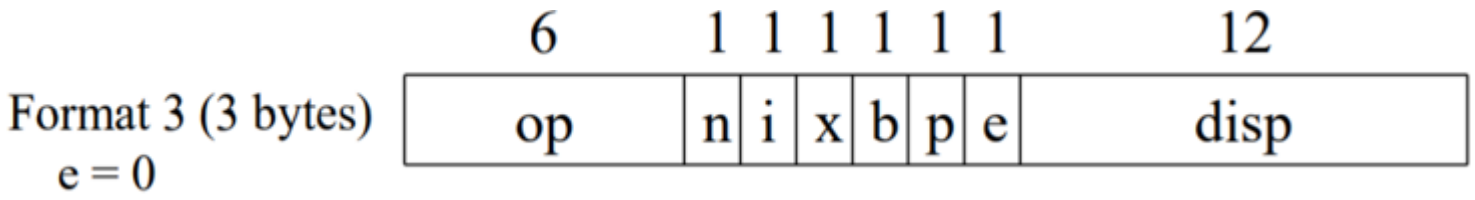


Eg: COMPR A,S

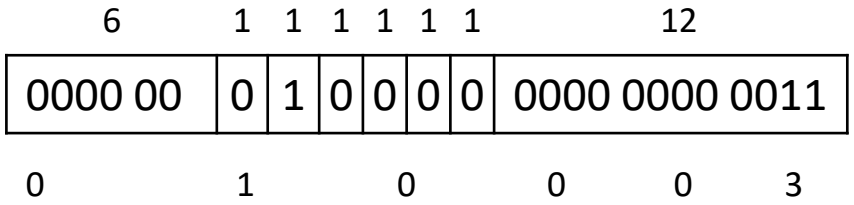


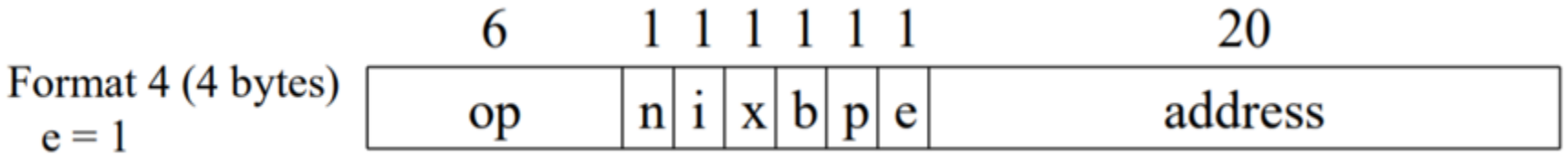
Formats 1 and 2 instructions do not reference memory at all

Instruction Format

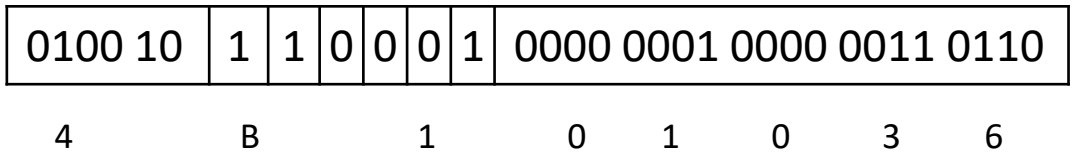


- LDA #3 (Load 3 to Accumulator A)





- +JSUB RDREC



Addressing Modes

- **Format 3 Instruction**
- x – index bit
  - x is set to 1, X register value is added for target address calculation
- Bits b and p specify relative addressing.
  - b = 1 and p = 0: Relative to B.
  - b = 0 and p = 1: Relative to PC.
  - b = 0 and p = 0: Direct address.
- Bits i and n specify extended addressing modes.
  - i = 1 and n = 0: Immediate mode.
  - i = 0 and n = 1: Indirect mode.
  - i = n: Simple addressing.
- Bit e is always 0 for 3-byte instructions.

Addressing Modes

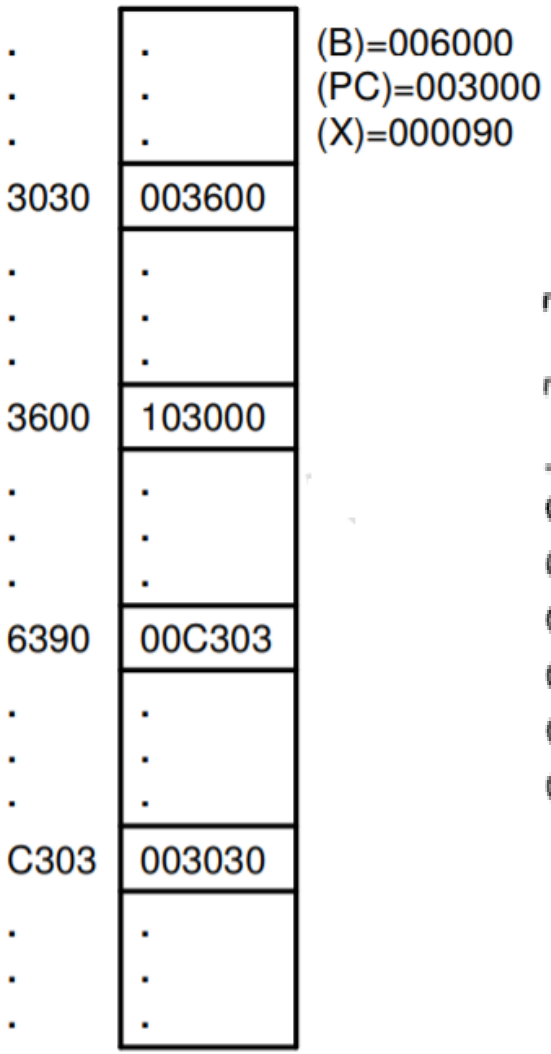
Mode	**Indication**	**Target Address Calculation**
Base relative	b=1, p=0	TA=(B)+ disp (0 ≤ disp ≤ 4095)
Program counter relative	b=0, p=1	TA=(PC)+ disp (-2048 ≤ disp ≤ 2047)
Immediate addressing	i=1, n=0	TA: TA is used as the operand value, no memory reference
Indirect addressing	i=0, n=1	((TA)): The word at the TA is fetched. Value of TA is taken as the address of the operand value
Simple addressing	i=0, n=0 or i=1, n=1	(TA): TA is taken as the address of the operand value
Base relative indexed addressing	b=1, p=0, x=1	TA=(B)+ (X) + disp (0 ≤ disp ≤ 4095)
Program counter relative indexed addressing	b=0, p=1, x=1	TA=(PC)+ (X)+ disp (-2048 ≤ disp ≤ 2047)

Addressing Modes

- For base relative addressing, the displacement field *disp* in format 3 is interpreted as a 12 bit unsigned integer
- For program counter relative mode is used, *disp* is a 12-bit signed integer with negative numbers represented in 2's complement form
- b and p - both set to 0, *disp* field from format 3 instruction is taken to be the target address.

- Format 4 Instruction
  - bits b and p are normally set to 0, 20 bit address is the target address
  - Bit e is always 1

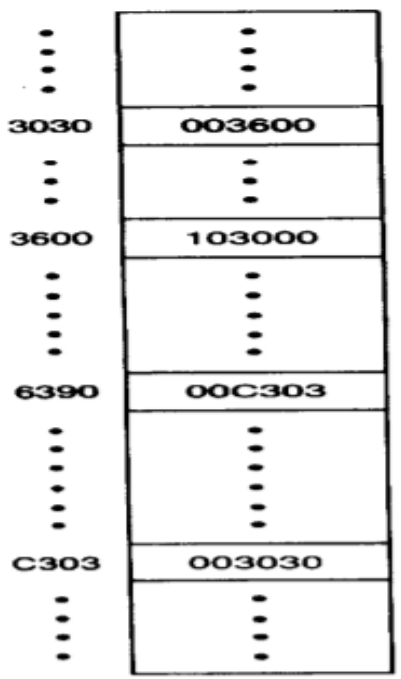
Addressing Modes Example



Machine instruction								
Hex	Binary							
	op	n	i	x	b	p	e	disp/address
032600	000000	1	1	0	0	1	0	0110 0000 0000
03C300	000000	1	1	1	1	0	0	0011 0000 0000
022030	000000	1	0	0	0	1	0	0000 0011 0000
010030	000000	0	1	0	0	0	0	0000 0011 0000
003600	000000	0	0	0	0	1	1	0110 0000 0000
0310C303	000000	1	1	0	0	0	1	0000 1100 0011 0000 0011

(b)

Examples of SIC/XE instructions and addressing modes.



(a)

(B) = 006000
(PC) = 003000
(X) = 000090

Machine instruction									
Hex	Binary							Target address	Value loaded into register A
	op	n	i	x	b	p	e	disp/address											
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

(b)

Examples of SIC/XE instructions and addressing modes.

Instruction Set

- Instructions to load and store the new registers: LDB, STB etc.
- Floating-point arithmetic operations:
  - ADDF, SUBF, MULF, DIVF
- Register move instruction : RMO
- Register-to-register arithmetic operations
  - ADDR, SUBR, MULR, DIVR
- Supervisor call instruction : SVC
  - Executing this instruction generates an interrupt that can be used for communicating with the operating system
- Integer Arithmetic operations: ADD, SUB, MUL, DIV
- Compare instruction, COMP
- Subroutine linkage instructions: JSUB, RSUB
- Branching instructions (Conditional Jump Instruction): JLT, JEQ, JGT

Input and Output

- There are I/O channels that can be used to perform input and output while the CPU is executing other instructions
- Allows overlap of computing and I/O, resulting in more efficient system operation
- The instructions SIO, TIO, and HIO are used to start, test and halt the operation of I/O channels

Assembler Directives

- Assembler directives are instructions that direct the assembler to do something. The assembler directives in SIC are:

1. START

- Indicates the start of program
- Used to define program name and starting address
- Eg: COPY START 1000
  - This means the program name is COPY and the starting address is 1000

2. END

- Used to indicate the end of program.
- Optionally indicates first executable instruction.
- Eg: END ALPHA
  - This means the program ends here. And the first executable instruction is ALPHA

Assembler Directives

3. RESW

- Used to reserve specified words for a data area.
- Eg: ALPHA RESW 4
  - This is to reserve 4 words. (4 words means 4*3 =12 bytes)

4. RESB

- Used to reserve specified bytes for a data area.
- Eg: A RESB 5
  - This is to reserve 5 bytes

5. WORD

- Used to generate one word integer constant
- Eg: B WORD 6
  - This uses one word to store the integer constant 6 and the name B is assigned to the first location

6. BYTE

- Generate character constant using required number of bytes
- Eg: ALPHA BYTE C ‘HAI’
  - This generates number of bytes needed to store HAI

Sample program SIC

- Data Movement Operations

```
        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      REWB    1       ONE BYTE VARIABLE
```

Sample program SIC/XE

- Data Movement Operations

```
        LDA     #5      LOAD VALUE 5 INTO REGISTER A
        STA     ALPHA   STORE IN ALPHA
        LDA     #90     LOAD ASCII CODE FOR  ‘Z’ INTO REGISTER A
        STCH    C1      STORE IN CHARACTER VARIABLE C1
        …
ALPHA   RESW    1       ONE WORD VARIABLE
C1      REWB    1       ONE BYTE VARIABLE
```

Sample Arithmetic Operations- SIC

```
        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
ALPHA   RESW    1         ONE WORD VARIABLE
BETA    RESW    1         ONE WORD VARIABLE
GAMMA   RESW    1         ONE WORD VARIABLE
DELTA   RESW    1         ONE WORD VARIABLE
INCR    RESW    1
```

Sample Arithmetic Operations- SIC/XE

```
        LDS     INCR    LOAD 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
        .......
ALPHA   RESW    1       ONE WORD VARIABLE
BETA    RESW    1
GAMMA   RESW    1
DELTA   RESW    1
INCR    RESW    1
```

Sample Indexing and Looping- SIC

```
         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
```

Sample Indexing and Looping- SIC/XE

```
            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
```

Sample Indexing and Looping- SIC

```
        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
```

Indexing and Looping- SIC/XE

```
        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
```

Sample Input-Output Operations-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     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 subroutine call and record input operations- 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
```

Sample subroutine call and record input operations- SIC/XE

```
        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
```