System Programming

System Software

Introduction

- Definition
  - System software consists of a variety of programs that support the operation of a computer
  - One characteristic in which most <u>system software</u> differ from <u>application software</u> is machine dependency
- Example:
  - e.g. when you took the first <u>programming course</u>
    - text editor, compiler, loader or linker, debugger
  - e.g. when you wrote <u>assembler language</u>
    - assembler, macro processor
  - e.g. you control all of these processes by interacting with the operation system

System Software vs. Machine Architecture

- One characteristic in which most <u>system software</u> differ from <u>application software</u> is machine dependency
- System Programs are intended to support the operation and use of the computer itself, rather than any particular application.
- They are usually related to the architecture of the machine on which they are to run.

System Software vs. Machine Architecture

  - e.g. assembler translate mnemonic instructions into machine code considering the instruction format, addressing modes etc.
  - e.g. compilers must generate machine language code taking in to account H/W characteristics such as the number and type of registers and machine instructions available
  - e.g. 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
  - e.g. general design and logic of an assembler
  - e.g. code optimization techniques used by the compiler
  - e.g. linking together independently assembled subprograms

The Simplified Instructional Computer (SIC)

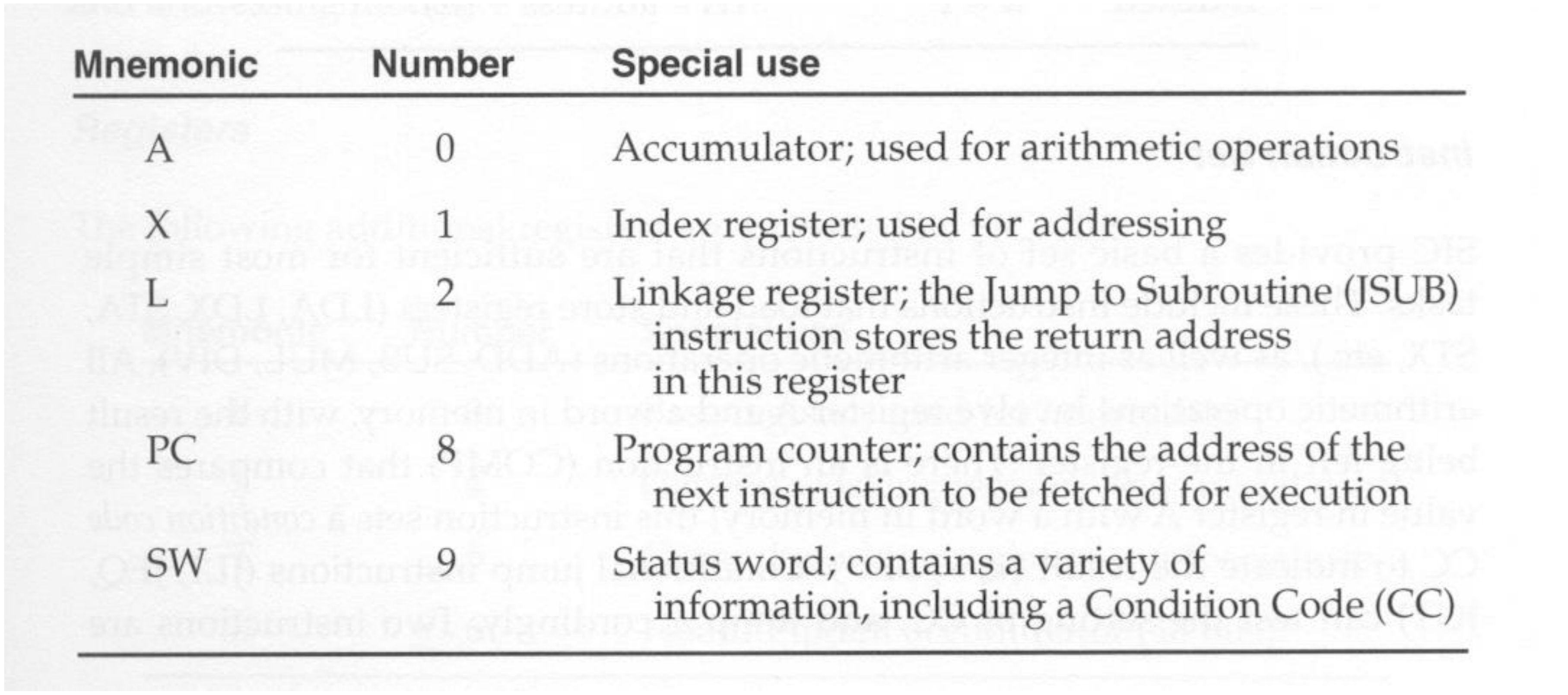
- SIC is a hypothetical computer that includes the hardware features most often found on real machines
- Two versions of SIC
  - standard model, SIC
  - XE version , SIC/XE(“extra equipment”)

SIC Architecture

- Memory
- Memory consists of 8-bit bytes.
- 3 consecutive bytes form a word (24 bits).
- All addresses in SIC are byte addresses
- A total of 2^{15} bytes (32,768) in the memory.

Registers

- Five registers, each have special use
- Each register is 24 bits in length



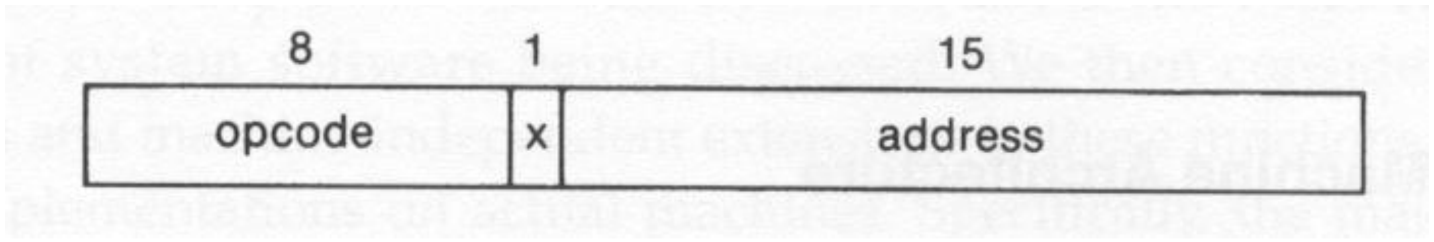
Mnemonic	Number	Special use
A	0	Accumulator; used for arithmetic operations
X	1	Index register; used for addressing
L	2	Linkage register; the Jump to Subroutine (JSUB) instruction stores the return address in this register
PC	8	Program counter; contains the address of the next instruction to be fetched 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 numbers.
- Characters are stored using their 8-bit ASCII codes.
- There is no floating-point hardware on SIC.

Instruction Format

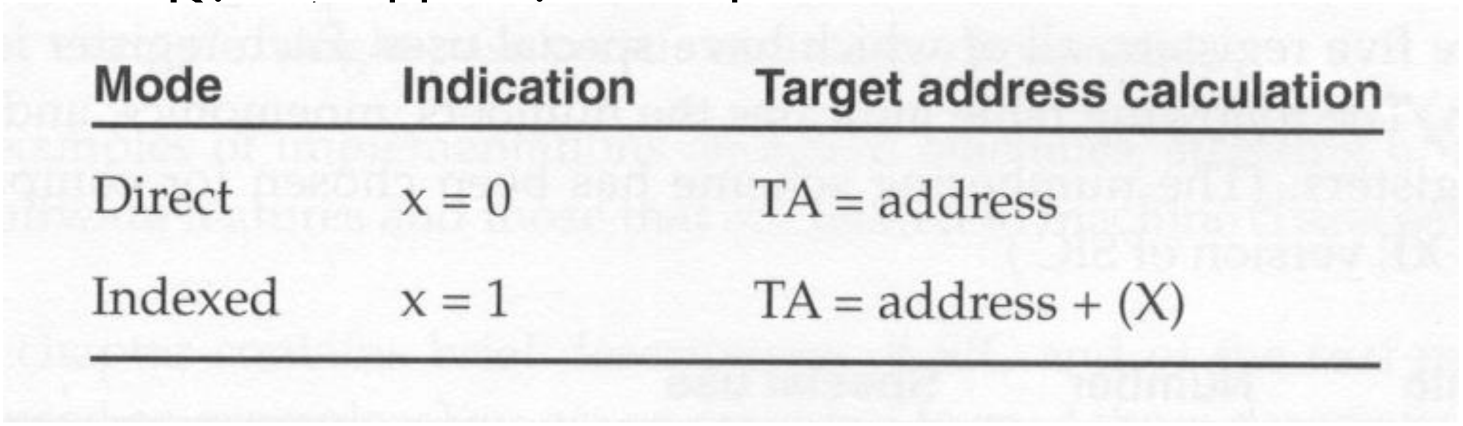
- All machine instructions on SIC has the following 24-bit format.



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

Addressing Modes

- Addressing modes refer to the way in which the operand of an instruction is specified.
- Only two modes are supported: indicated by the setting up of ‘x’ bit in the instruction.



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

address – address given in the instruction

(X) – contents of register X

SIC Machine Architecture

- Instruction Set
  - load and store: LDA, LDX, STA, STX, etc.
  - integer arithmetic operations: ADD, SUB, MUL, DIV, etc.
    - All arithmetic operations involve register A and a word in memory, with the result being left in the register
  - comparison: COMP
    - COMP compares the value in register A with a word in memory, this instruction sets a condition code CC to indicate the result(<,= or >).

- conditional jump instructions: JLT, JEQ, JGT
  - these instructions test the setting of 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

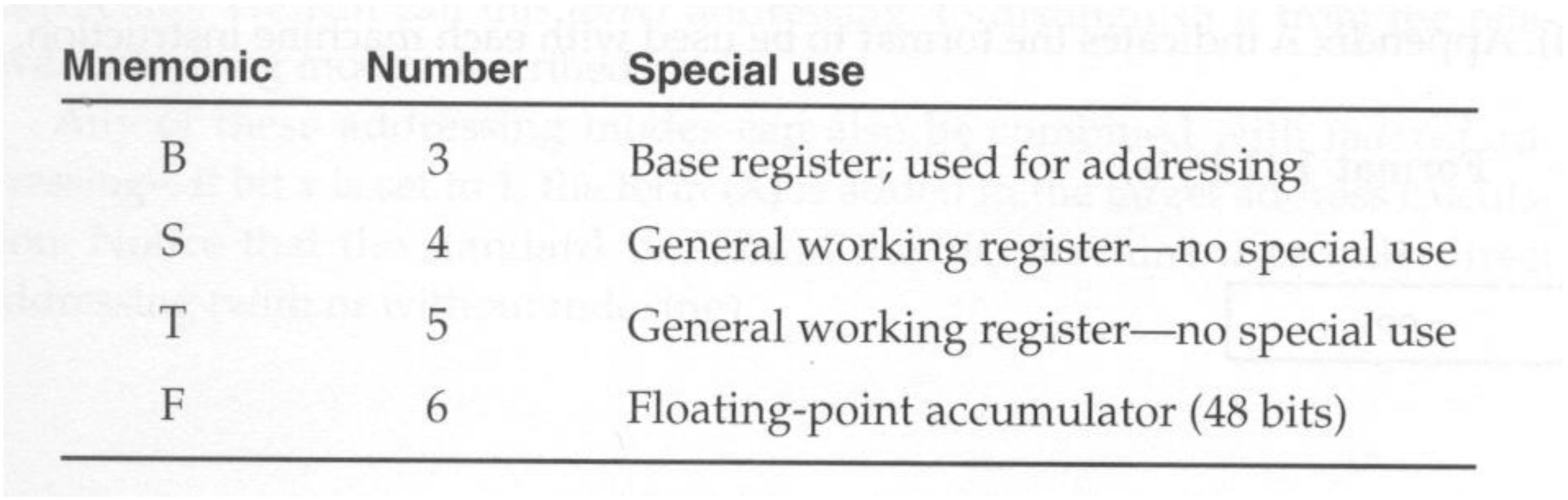
Input and Output

- I/O are performed by transferring one byte at a time to or from the rightmost 8 bits of register A.
- Each device has a unique 8-bit code.
- Three I/O instructions; each use device code as operand.
- Test device (TD): tests if a device is ready to send or receive a byte of data.
- Read data (RD): read a byte from the device to register A
- Write data (WD): write a byte from register A to the device.

SIC/XE Architecture

- The memory structure of SIC/XE is same as that for SIC.
- Memory: 1 megabytes (2^{20} bytes)

Additional Registers



Mnemonic	Number	Special use
B	3	Base register; used for addressing
S	4	General working register—no special use
T	5	General working register—no special use
F	6	Floating-point accumulator (48 bits)

SIC/XE Machine Architecture

- Data Formats
  - 48-bit Floating-point data type:

s	exponent (11)	fraction (36)

    - Fraction : a value b/w 0 and 1 (0~1)
    - Exponent : 0~2047
- s – indicate the sign of the floating point number.
- s=0 (positive)
- s=1 (negative)
- Absolute value of a number : frac*2$^{(exp-1024)}$

Instruction Formats

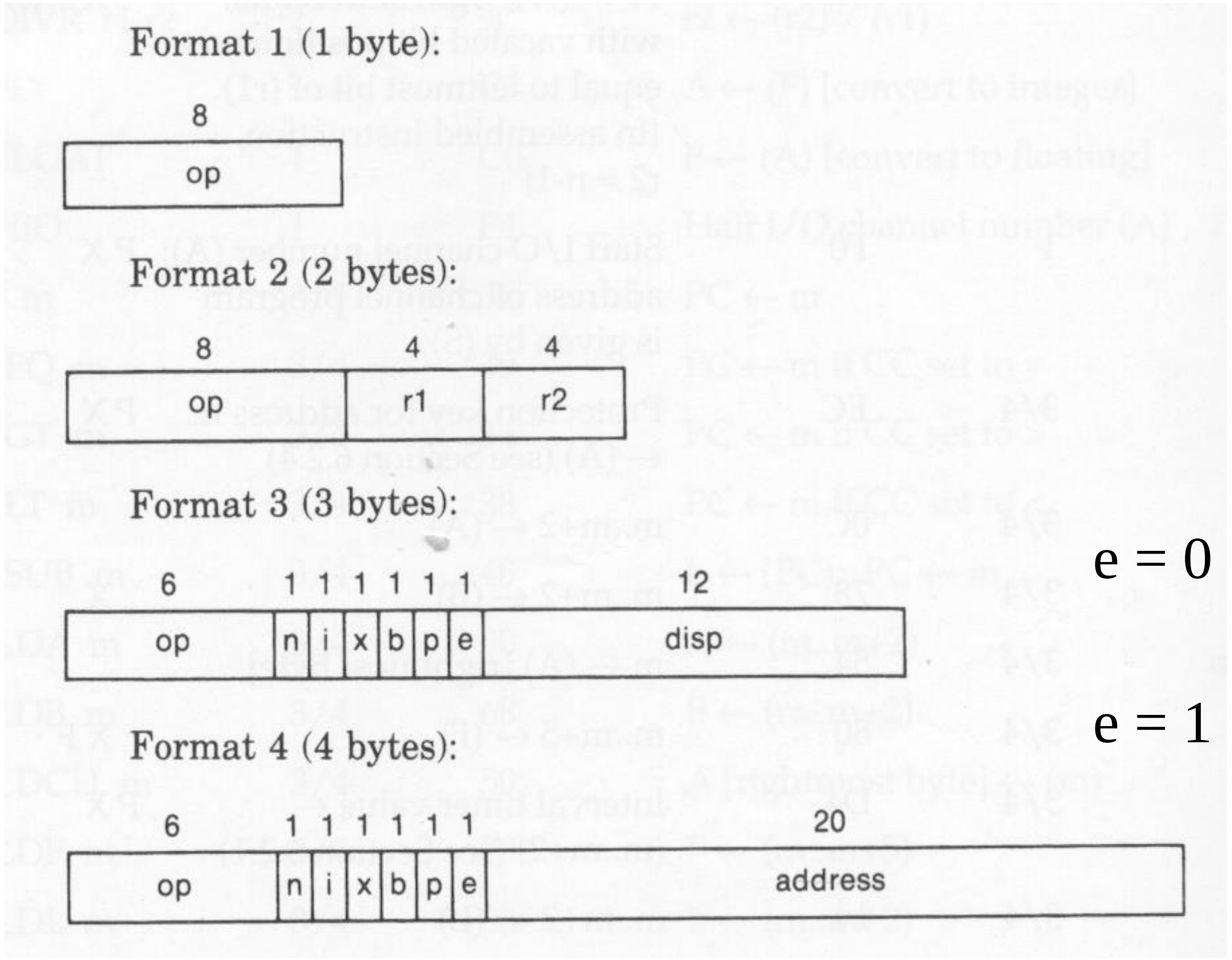
- larger memory -> extend addressing capacity
- Two Options:

a) Use relative addressing or (format 3)

b) Extend the address field to 20 bits. (format 4)

- SIC/XE also supports some instructions that do not reference memory at all. (format 1 & 2)

Instruction Formats

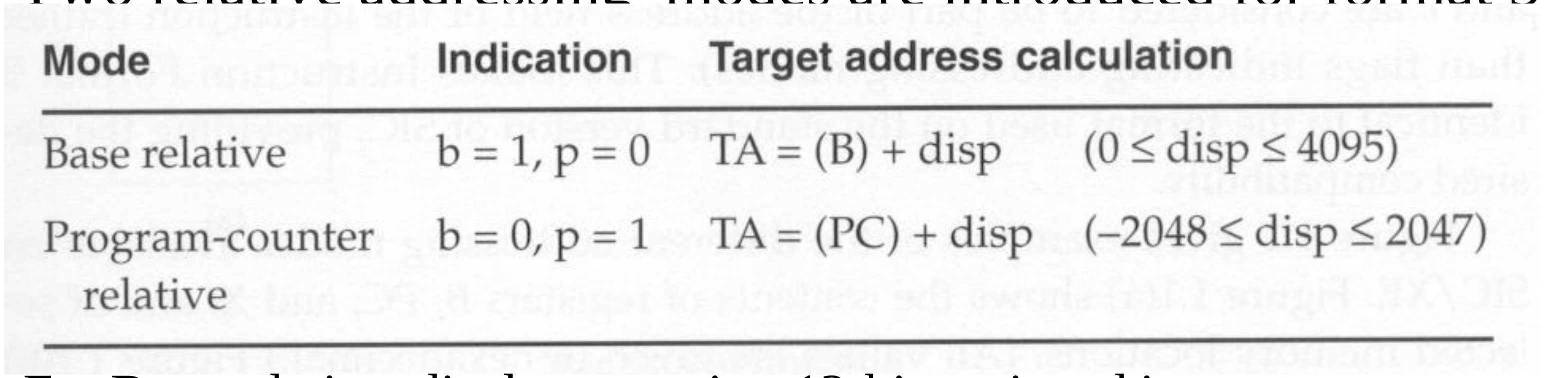


e = 0

e = 1

Addressing Mode

Two relative addressing modes are introduced for format 3:



Mode	Indication	Target address calculation	
Base relative	b = 1, p = 0	TA = (B) + disp	$(0 \le \text{disp} \le 4095)$
Program-counter relative	b = 0, p = 1	TA = (PC) + disp	$(-2048 \le \text{disp} \le 2047)$

For Base relative, displacement is a 12-bit unsigned integer.
For Program-counter relative, disp is a 12-bit signed integer.

In format 3 and 4, if both bits b and p are set to 0, the disp field (address) is taken as the target address. This is called direct addressing mode.

Both format 3 and 4 modes can be combined with indexed addressing – if bit x is set to 1, the value of register X is added in the target Address calculation.

Addressing Modes

For format 3 and 4:

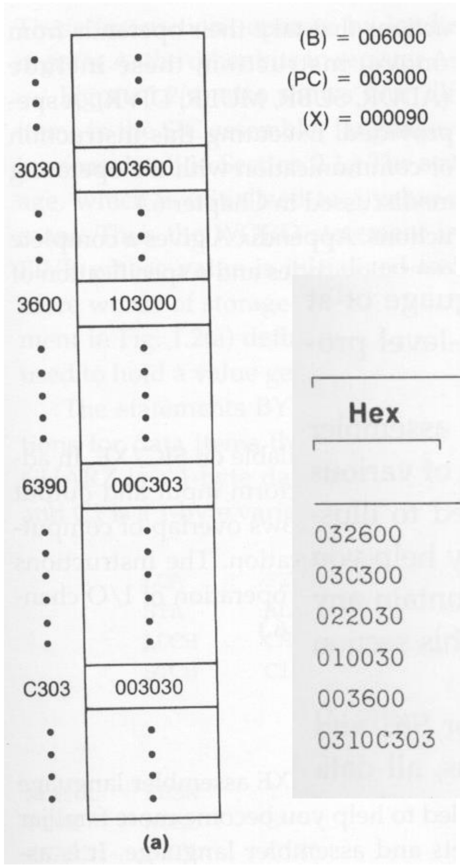
- Bits i and n specify how the target address is used.
- (i =1, n = 0): immediate addressing mode. The target address is used as the operand. No memory reference.
- (i = 0, n = 1): indirect addressing mode. The word at the location given by the target address is fetched; the value contained in this word is then used as the address of the operand value.
- (i = 0, n = 0) used by SIC, (i=1, n=1) used by SIC/XE: simple addressing mode. The target address is taken as the location of the operand.

Indexing cannot be used with immediate or indirect modes.

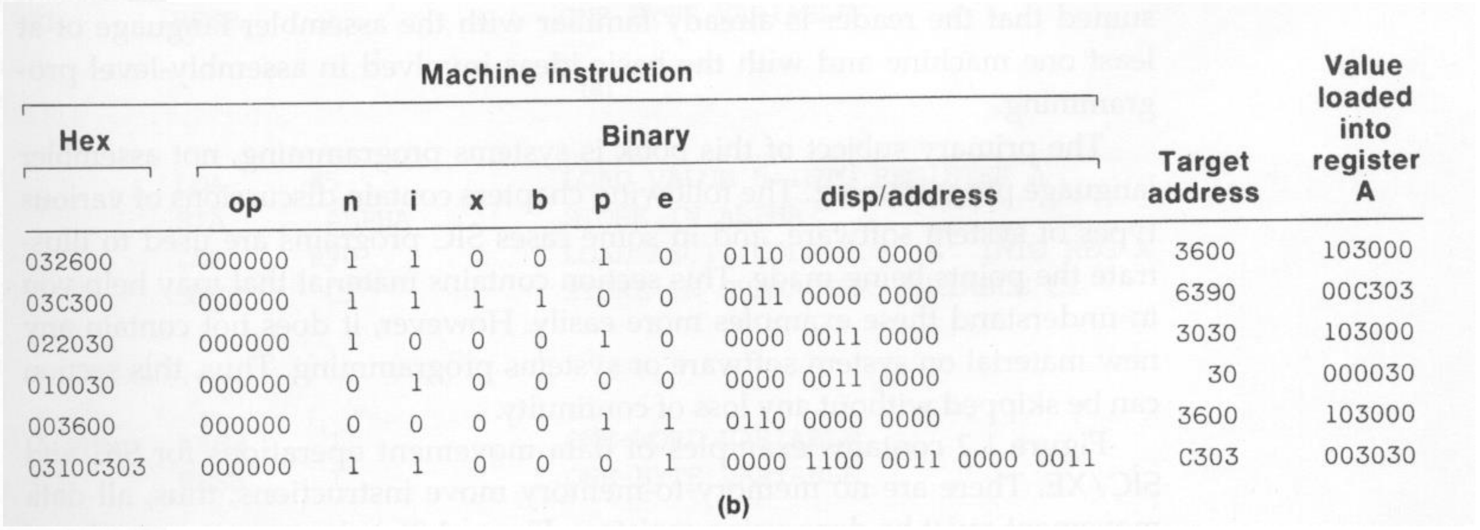
Addressing type	Flag bits n i x b p e	Assembler language notation	Calculation of target address TA	Operand	Notes
Simple	110000	op c	disp	(TA)	D
	110001	+op m	addr	(TA)	4 D
	110010	op m	(PC) + disp	(TA)	A
	110100	op m	(B) + disp	(TA)	A
	111000	op c,X	disp + (X)	(TA)	D
	111001	+op m,X	addr + (X)	(TA)	4 D
	111010	op m,X	(PC) + disp + (X)	(TA)	A
	111100	op m,X	(B) + disp + (X)	(TA)	A
	000- - -	op m	b/p/e/disp	(TA)	D S
	001- - -	op m,X	b/p/e/disp + (X)	(TA)	D S
Indirect	100000	op @c	disp	((TA))	D
	100001	+op @m	addr	((TA))	4 D
	100010	op @m	(PC) + disp	((TA))	A
	100100	op @m	(B) + disp	((TA))	A
Immediate	010000	op #c	disp	TA	D
	010001	+op #m	addr	TA	4 D
	010010	op #m	(PC) + disp	TA	A
	010100	op #m	(B) + disp	TA	A

Example

All of these instructions are LDA.



(a)



Machine instruction										
Hex	Binary									
	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

(b)

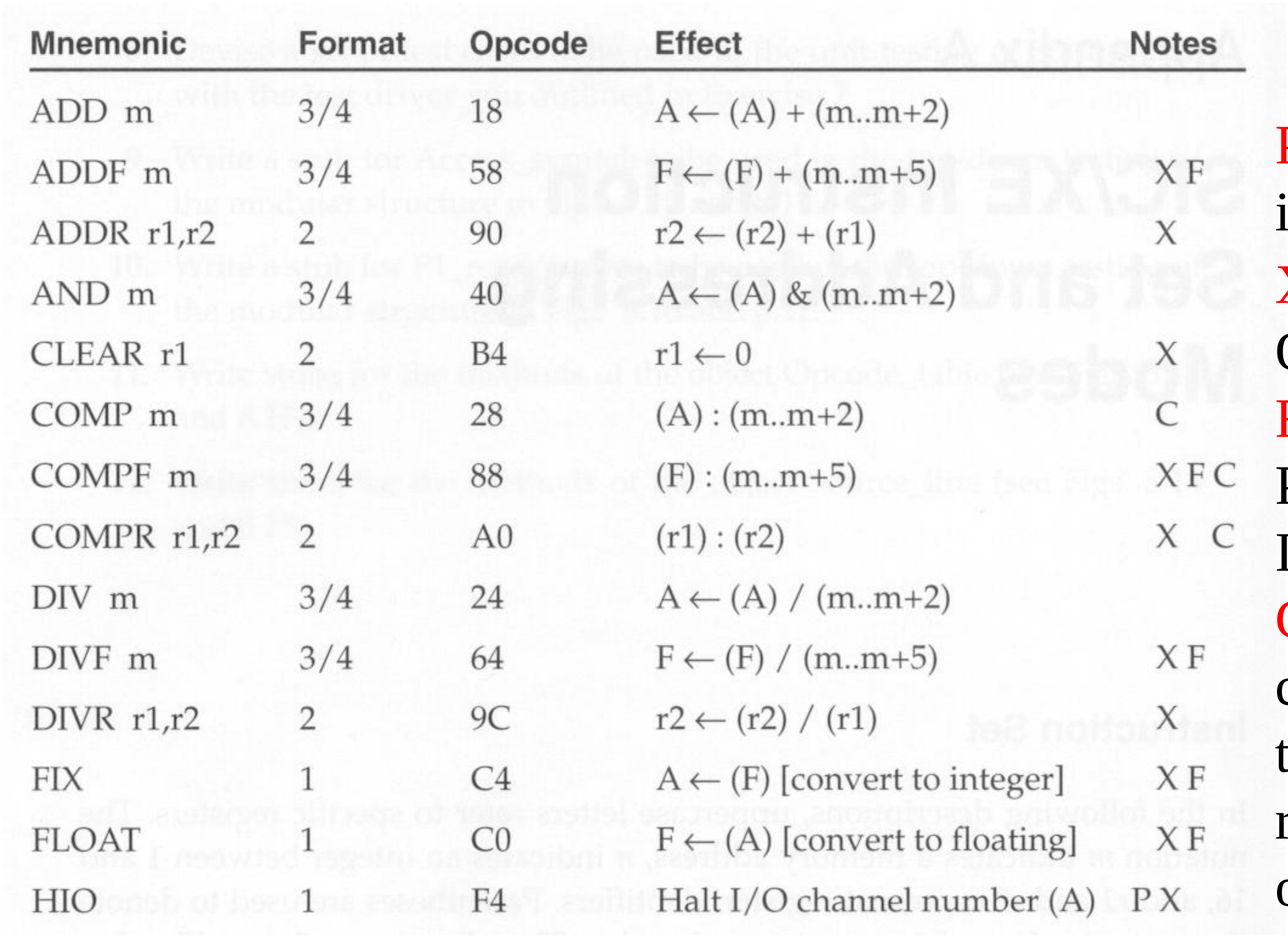
Instruction Set

- Instructions to load and store new registers.(eg:-LDB and STB etc.)
- Floating-point operations (ADDF, SUBF, MULF, DIVF)
- Instructions that take operands from registers: Register move (RMO)
- Register-to-register operations (ADDR, SUBR, MULR, DIVR)
- Supervisor call (SVC) for generating system calls (interrupt) into the operating system.

Input and Output

- SIC/XE has input channels.
- Input channels are used to perform input and output while CPU is executing other instructions.
- It allows overlapping of computing and I/O.
- I/O channel operation (SIO: start, TIO: test, HIO: halt), similar to DMA

Complete Instruction Set



Mnemonic	Format	Opcode	Effect	Notes
ADD m	3/4	18	A ← (A) + (m..m+2)	
ADDF m	3/4	58	F ← (F) + (m..m+5)	X F
ADDR r1,r2	2	90	r2 ← (r2) + (r1)	X
AND m	3/4	40	A ← (A) & (m..m+2)	
CLEAR r1	2	B4	r1 ← 0	X
COMP m	3/4	28	(A) : (m..m+2)	C
COMPF m	3/4	88	(F) : (m..m+5)	X F C
COMPR r1,r2	2	A0	(r1) : (r2)	X C
DIV m	3/4	24	A ← (A) / (m..m+2)	
DIVF m	3/4	64	F ← (F) / (m..m+5)	X F
DIVR r1,r2	2	9C	r2 ← (r2) / (r1)	X
FIX	1	C4	A ← (F) [convert to integer]	X F
FLOAT	1	C0	F ← (A) [convert to floating]	X F
HIO	1	F4	Halt I/O channel number (A)	P X

P: privileged instruction

X: available Only on XE

F: floating-Point Instruction

C: condition code CC set to indicate result of operation

J m	3/4	3C	PC ← m	
JEQ m	3/4	30	PC ← m if CC set to =	
JGT m	3/4	34	PC ← m if CC set to >	
JLT m	3/4	38	PC ← m if CC set to <	
JSUB m	3/4	48	L ← (PC); PC ← m	
LDA m	3/4	00	A ← (m..m+2)	
LDB m	3/4	68	B ← (m..m+2)	X
LDCH m	3/4	50	A [rightmost byte] ← (m)	
LDF m	3/4	70	F ← (m..m+5)	X F
LDL m	3/4	08	L ← (m..m+2)	
LDS m	3/4	6C	S ← (m..m+2)	X
LDT m	3/4	74	T ← (m..m+2)	X
LDX m	3/4	04	X ← (m..m+2)	
LPS m	3/4	D0	Load processor status from information beginning at address m (see Section 6.2.1)	P X
MUL m	3/4	20	A ← (A) * (m..m+2)	

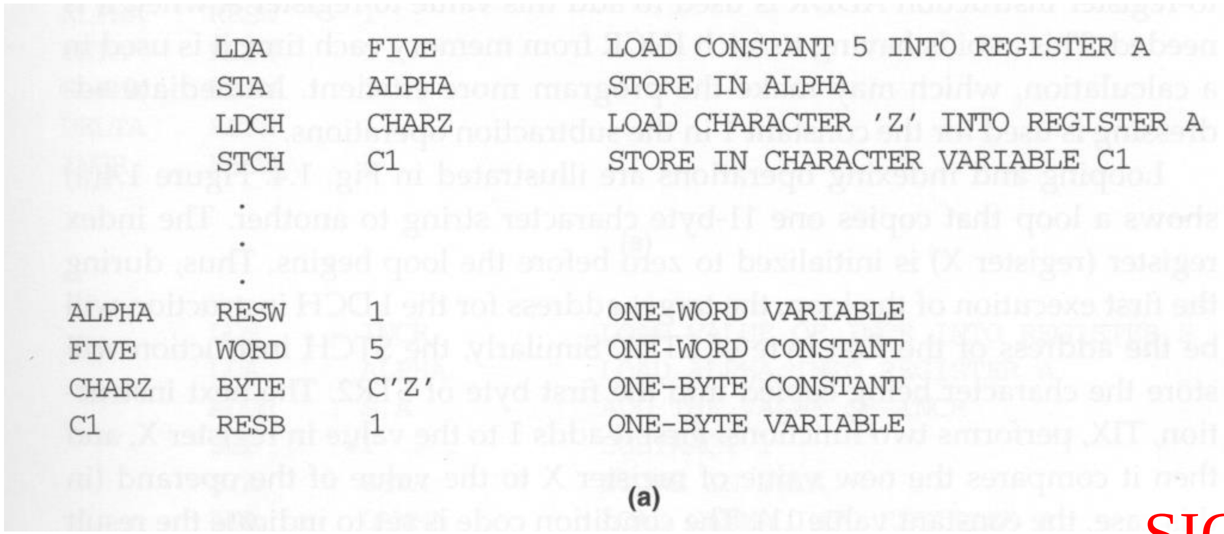
Mnemonic	Format	Opcode	Effect	Notes
MULF m	3/4	60	F ← (F) * (m..m+5)	X F
MULR r1, r2	2	98	r2 ← (r2) * (r1)	X
NORM	1	C8	F ← (F) [normalized]	X F
OR m	3/4	44	A ← (A) | (m..m+2)	
RD m	3/4	D8	A [rightmost byte] ← data from device specified by (m)	P
RMO r1,r2	2	AC	r2 ← (r1)	X
RSUB	3/4	4C	PC ← (L)	
SHIFTL r1,n	2	A4	r1 ← (r1); left circular shift n bits. {In assembled instruction, r2 = n–1}	X
SHIFTR r1,n	2	A8	r1 ← (r1); right shift n bits, with vacated bit positions set equal to leftmost bit of (r1). {In assembled instruction, r2 = n-1}	X
SIO	1	F0	Start I/O channel number (A); address of channel program is given by (S)	P X

Mnemonic	Format	Opcode	Effect	Notes		
SUBR r1,r2	2	94	r2 ← (r2) – (r1)		X	
SVC n	2	B0	Generate SVC interrupt. {In assembled instruction, r1 = n}		X	
TD m	3/4	E0	Test device specified by (m)	P		C
TIO	1	F8	Test I/O channel number (A)	P	X	C
TIX m	3/4	2C	X ← (X) + 1; (X): (m..m+2)			C
TIXR r1	2	B8	X ← (X) + 1; (X): (r1)		X	C
WD m	3/4	DC	Device specified by (m) ← (A) [rightmost byte]	P		

SSK m	3/4	EC	Protection key for address m ← (A) (see Section 6.2.4)	P X
STA m	3/4	0C	m..m+2 ← (A)	
STB m	3/4	78	m..m+2 ← (B)	X
STCH m	3/4	54	m ← (A) [rightmost byte]	
STF m	3/4	80	m..m+5 ← (F)	X F
STI m	3/4	D4	Interval timer value ← (m..m+2) (see Section 6.2.1)	P X
STL m	3/4	14	m..m+2 ← (L)	
STS m	3/4	7C	m..m+2 ← (S)	X
STSW m	3/4	E8	m..m+2 ← (SW)	P
STT m	3/4	84	m..m+2 ← (T)	X
STX m	3/4	10	m..m+2 ← (X)	
SUB m	3/4	1C	A ← (A) – (m..m+2)	
SUBF m	3/4	5C	F ← (F) – (m..m+5)	X F

SIC Programming Example (Data Movement operations)

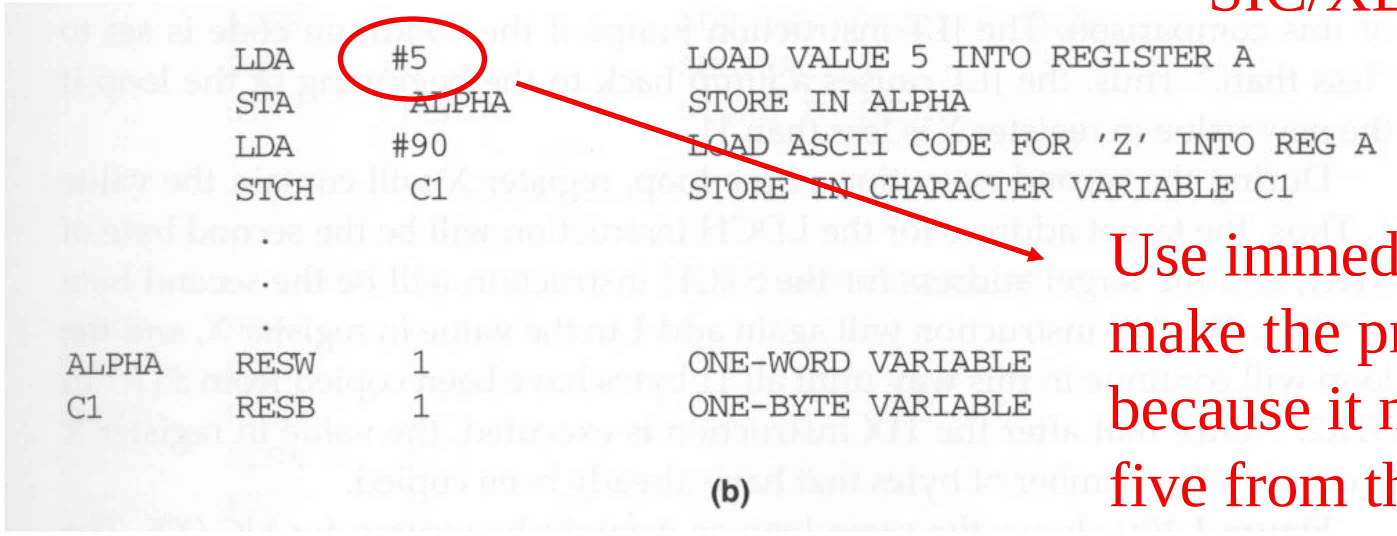
SIC instruction sets



```
        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)

SIC/XE instruction sets

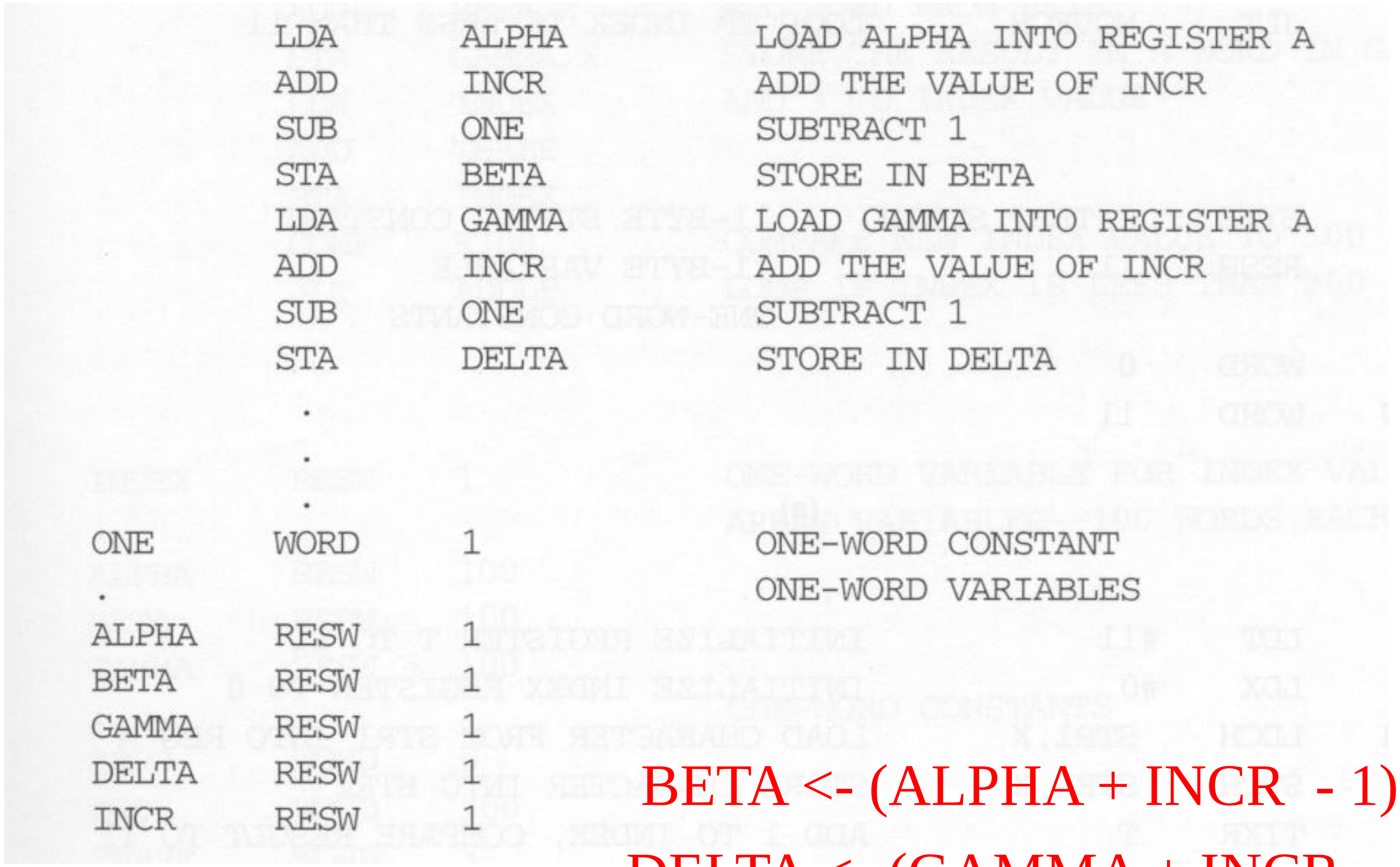


```
        LDA     #5        LOAD VALUE 5 INTO REGISTER A
        STA     ALPHA     STORE IN ALPHA
        LDA     #90       LOAD ASCII CODE FOR 'Z' INTO REG A
        STCH    C1        STORE IN CHARACTER VARIABLE C1
          .
          .
          .
ALPHA   RESW    1         ONE-WORD VARIABLE
C1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

(b)

Use immediate mode will make the program run faster because it need not fetch five from the memory.

SIC Programming Example (Arithmentic Instructions)

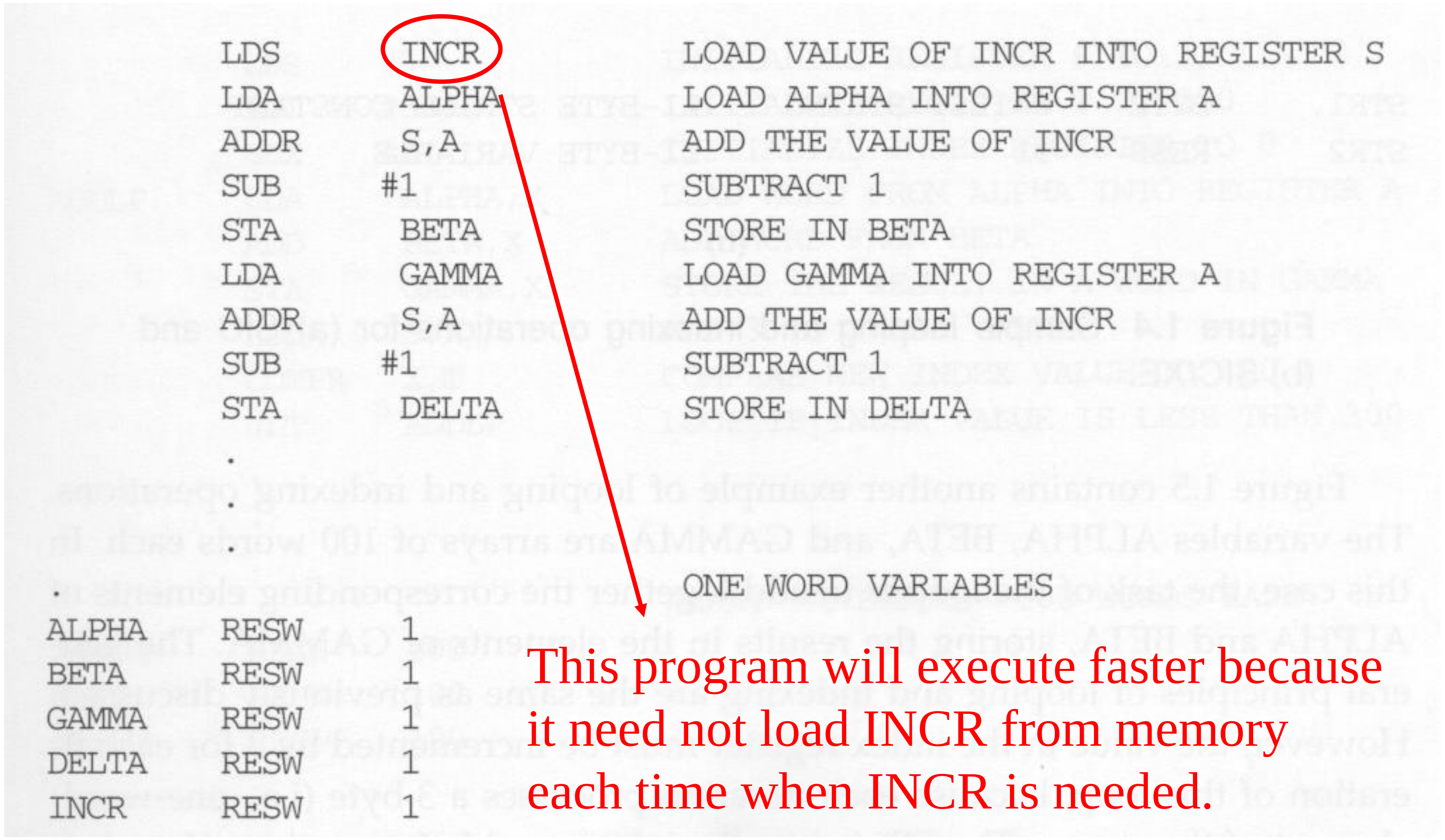


```
          LDA     ALPHA           LOAD ALPHA INTO REGISTER A
          ADD     INCR            ADD THE VALUE OF INCR
          SUB     ONE             SUBTRACT 1
          STA     BETA            STORE IN BETA
          LDA     GAMMA           LOAD GAMMA INTO REGISTER A
          ADD     INCR            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)

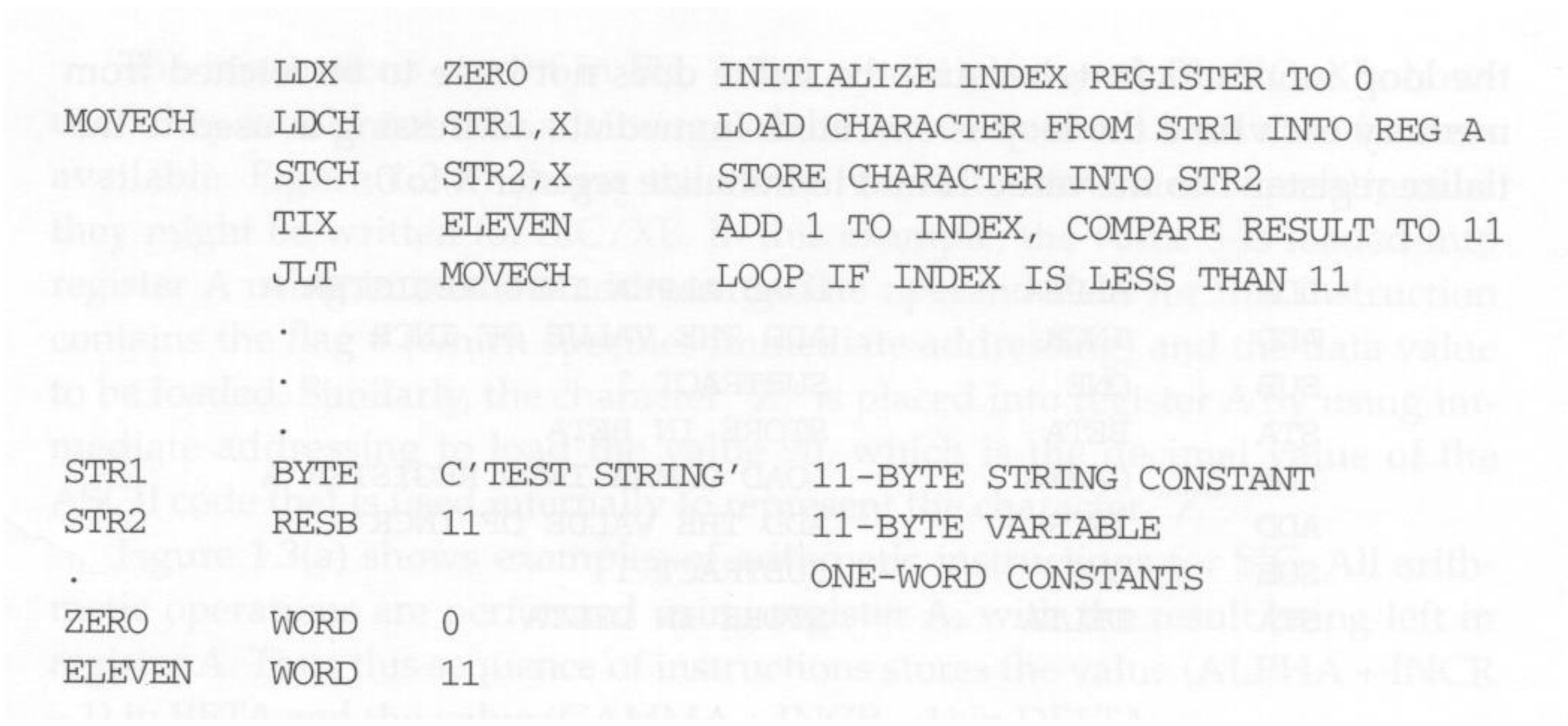
SIC/XE Programming Example (Arithmentic Instructions)



```
            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
```

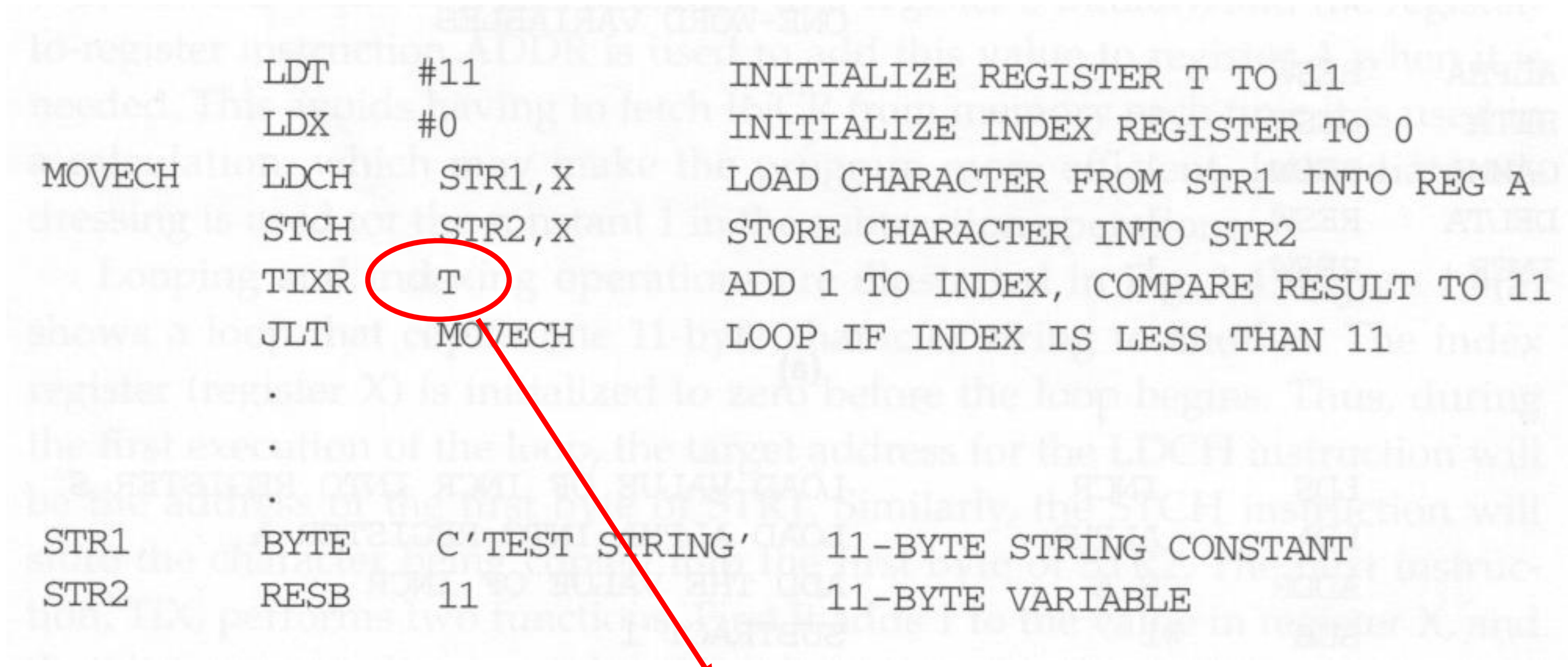
This program will execute faster because it need not load INCR from memory each time when INCR is needed.

SIC Programming Example (Looping and Indexing operations)



```
         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
```

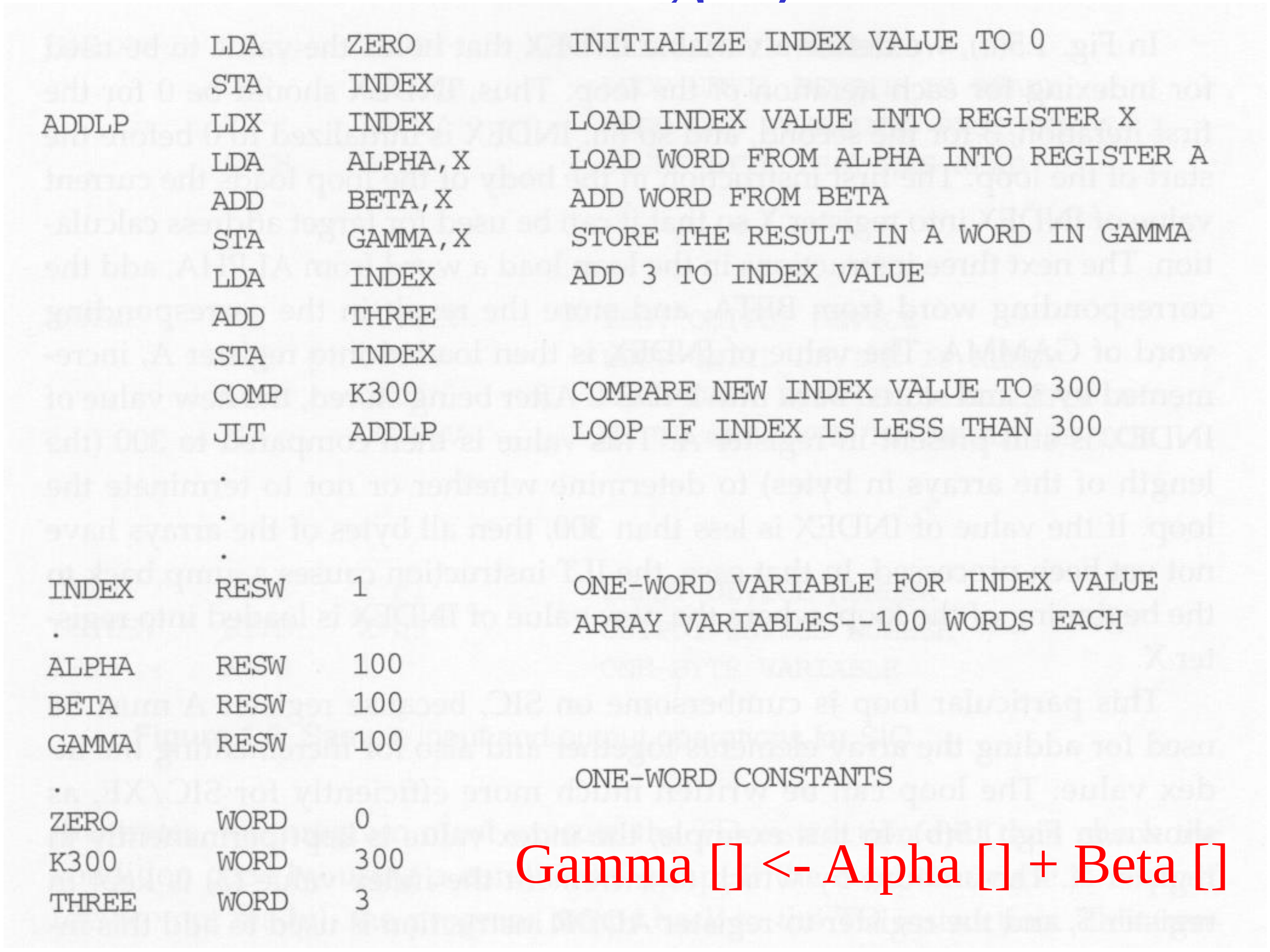
SIC/XE Programming Example (Looping and Indexing operations)



```
        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
```

This program will execute faster because TIXR need not compare the index value to a memory variable.

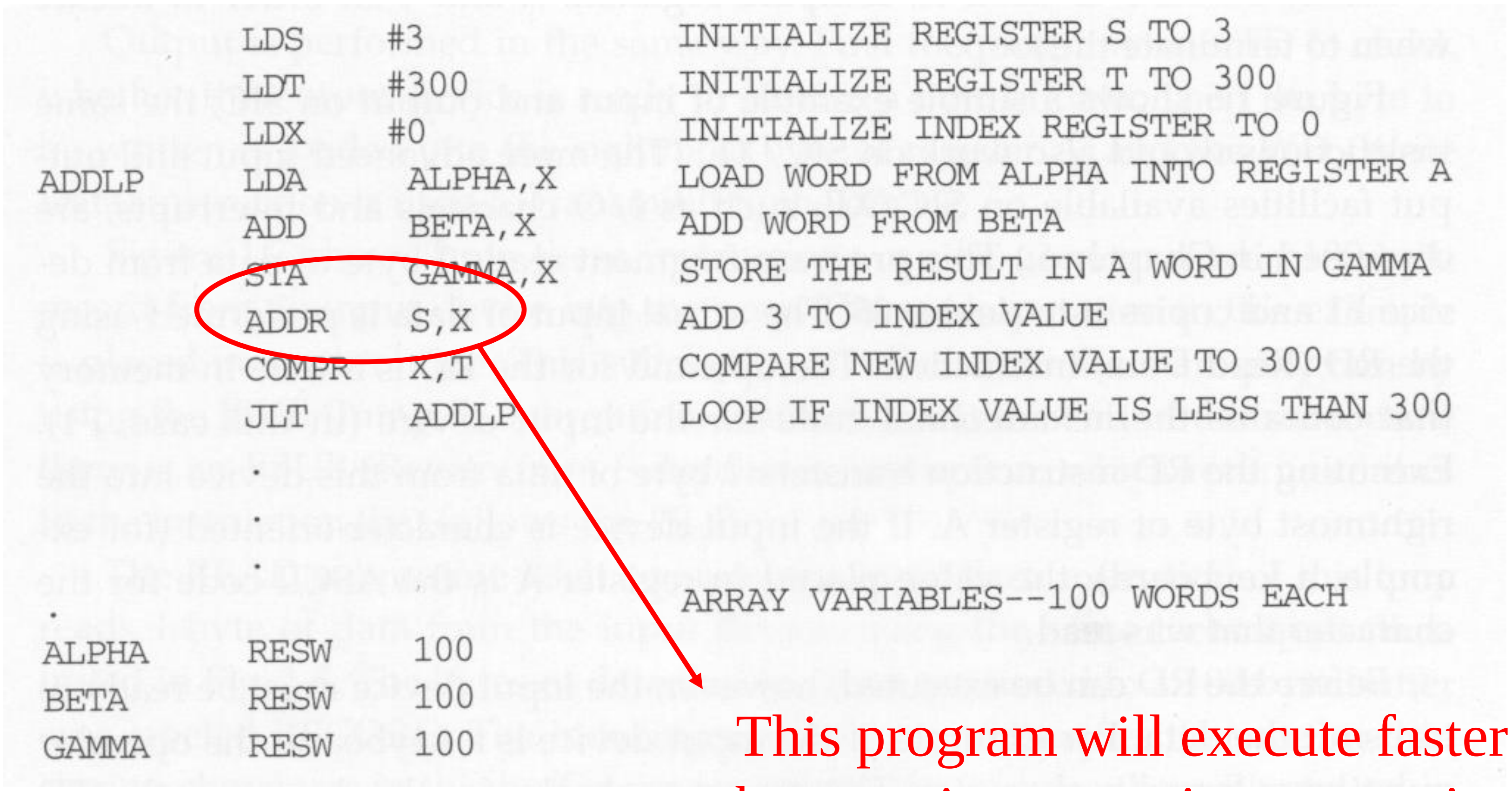
SIC Programming Example (Looping and Indexing-2)



```
         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
```

Gamma [] <- Alpha [] + Beta []

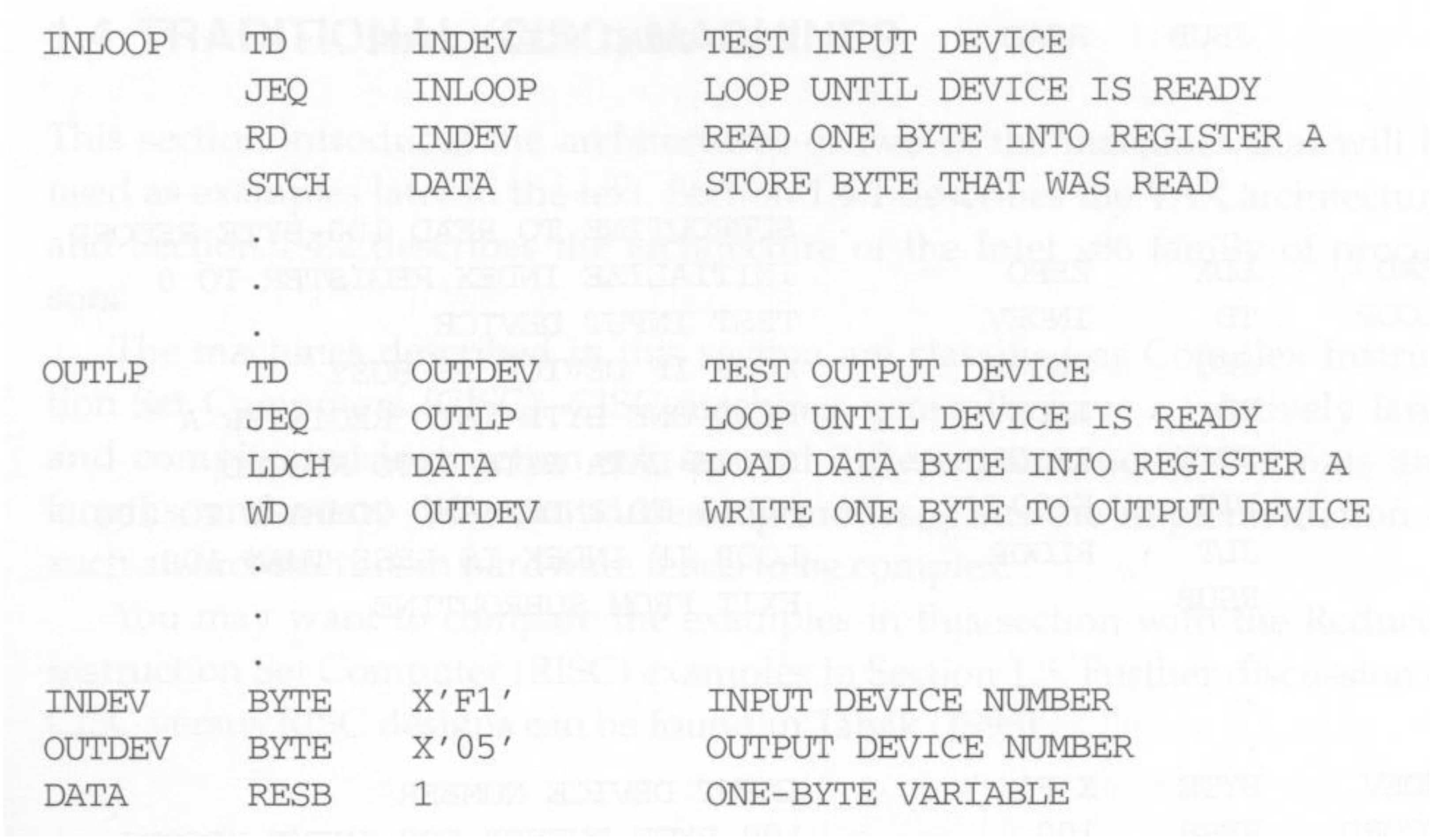
SIC Programming Example (Looping and Indexing)



```
        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
```

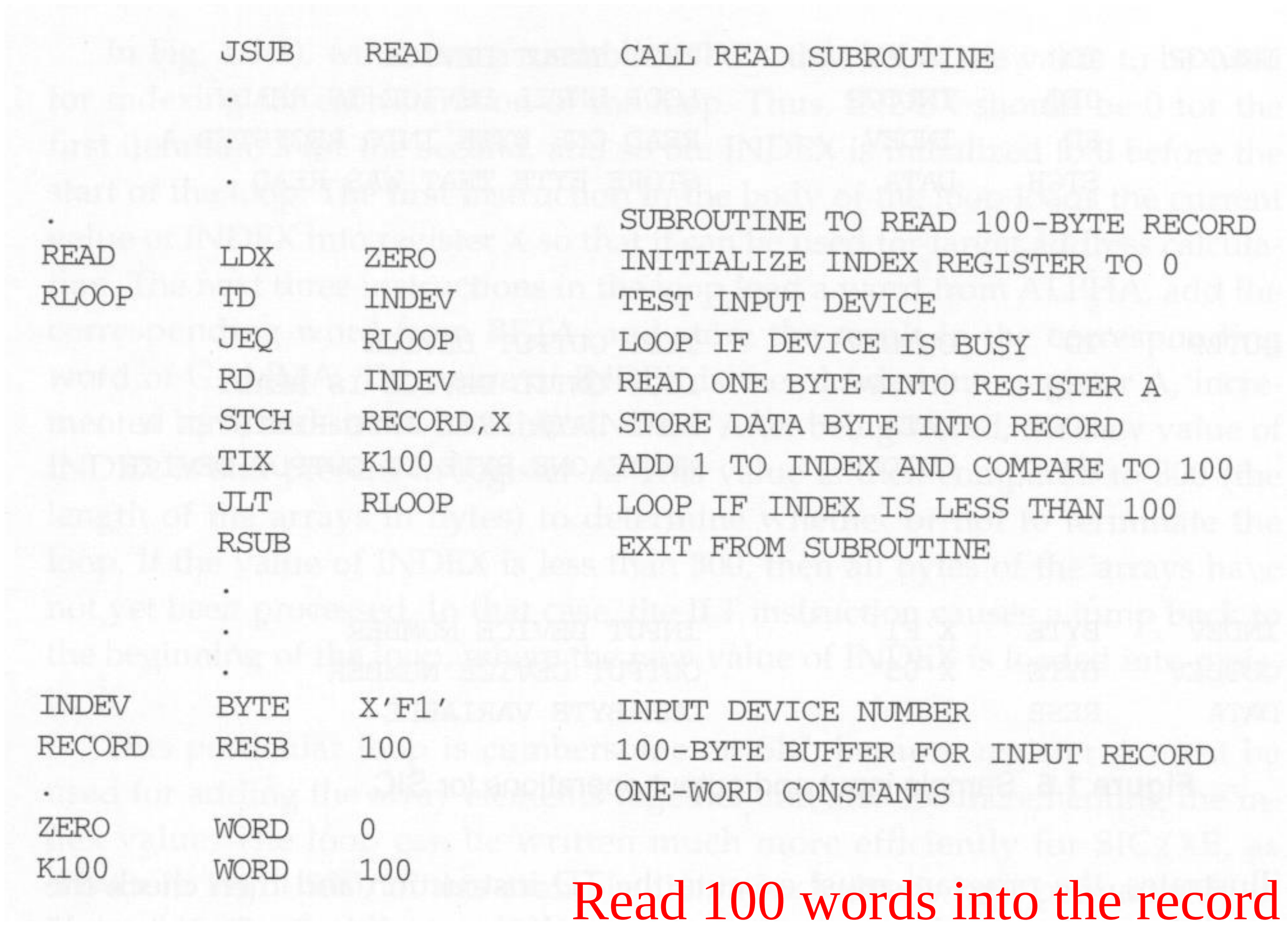
This program will execute faster because it uses register-to-register add to reduce memory accesses.

Input and Output Examples



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

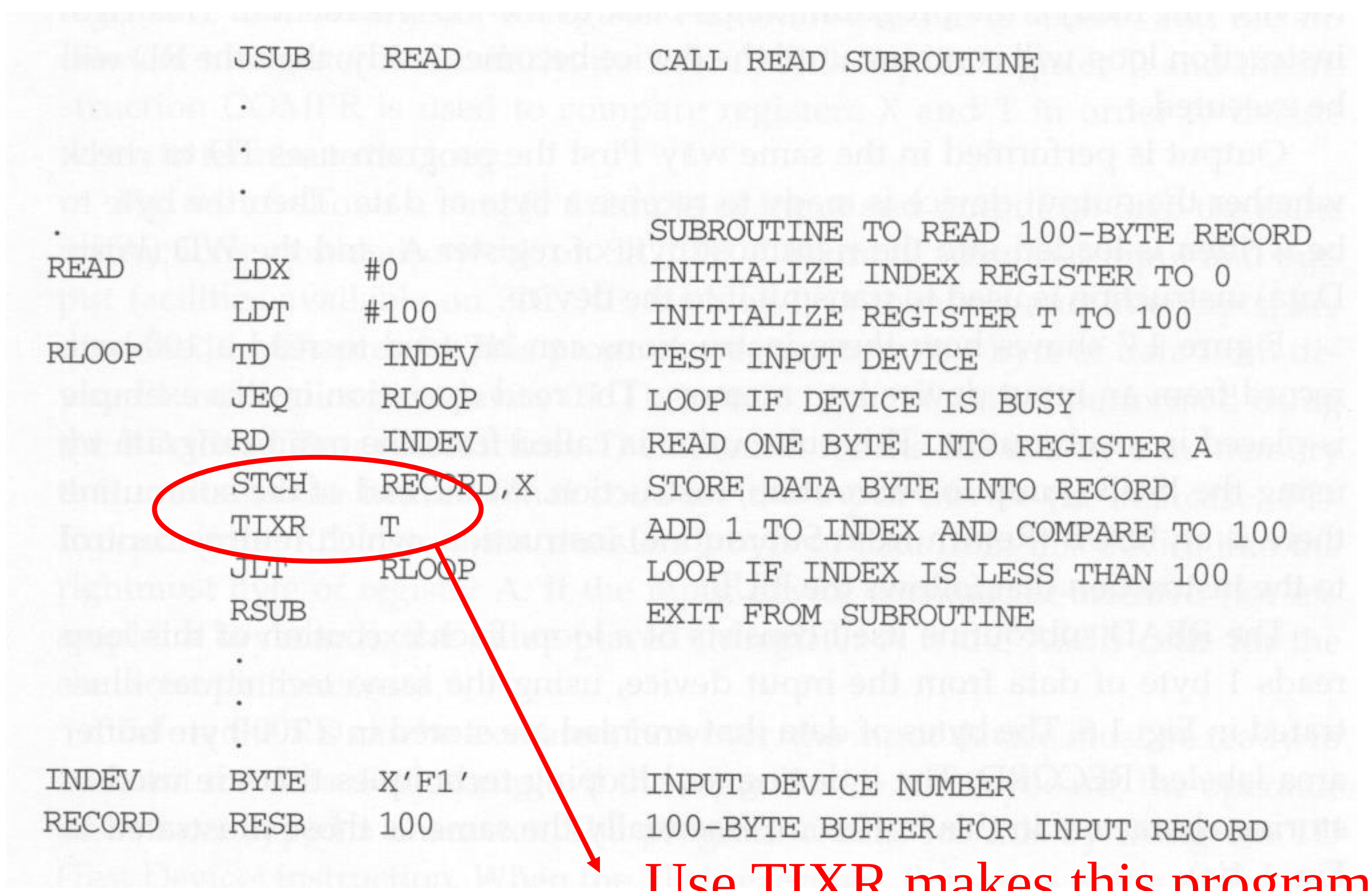
Subroutine Call Example



```
          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
```

Read 100 words into the record buffer

Subroutine Call Example (5.b)



```
         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
```

Use TIXR makes this program run faster