LSI-2 EXTENDED INSTRUCTION SET

SUPPLEMENT

TO

NAKED MINI®/ALPHA LSI SERIES

PROGRAMMING REFERENCE MANUAL

This document contains information that is supplementary to the NAKED MINI®/ALPHA LSI SERIES PROGRAMMING REFERENCE MANUAL -- Document 10077-00B1.

This supplement consists of three sections and three appendices. Each section and appendix provides information that will be incorporated into Document 10077-00B1 at a future date. For correlation purposes, the section and appendices of this supplement relate to the basic manual as follows: ,

Supplement	Basic Manual	Title
Section 1	Section 1	General Information
Section 2	Section 3	Instructions and Directives
Section 3	To be added	Programming Examples
Appendix A	Appendix D	Instruction Set in Alphabetical Order
Appendix B	Appendix E	Instruction Set in Numerical Order
Appendix C	Appendix F	ALPHA LSI Execution Times

Section 1

GENERAL INFORMATION

1.1 INTRODUCTION

This supplement describes Computer Automation's LSI-2 Processor Extended instruction set. The instruction set consists of 20 new instructions for the NAKED MINI / ALPHA LSI Type 2 Computer only. Other versions of Computer Automation's family of computers cannot execute these new instructions.

1.2 STACK INSTRUCTIONS

The stack processing instructions extend Memory Reference operations to operands maintained in "stacks" in memory.

The number, size, and location of stacks in use at any time are unlimited as are the number of stacks in use by any code module and the number of code modules using any given stack.

All stack accesses are controlled by a stack pointer. Stacks may be accessed in the conventional "PUSH" and "POP" fashion utilizing automatic hardware pre-decrement and post-increment respectively, of the stack pointer. Stack contents can also be accessed directly or with indexing thru the stack pointer without altering the stack pointer value.

Stack processing instructions greatly facilitate the generation of reentrant code modules by allowing the reentrant module to operate only on variables contained in stacks. Simple manipulation of the one-word stack pointer, by exchanging one parameter for another, can then cause the re-entrant module to operate on any of a number of stacks.

Since access to stacks is unrestricted, no hardware prevention or detection of stack overflow, underflow, or overlap is provided.

1.3 REGISTER CHANGE INSTRUCTIONS

This class of instructions has been expanded to include the ability to complement one register and either AND or OR the result with the other register. The EIX instruction provides the ability to execute a single instruction that is not part of the inline program sequence.

Section 2

INSTRUCTIONS AND DIRECTIVES

2.1 INTRODUCTION

This section defines the new LSI-2 instructions. Also defined is one new directive that is processed by the various assemblers used with the LSI-2.

2.2 STACK, DOUBLE WORD INSTRUCTIONS (Figure 2-1)

Stack instructions permit the Programmer to enter or retrieve a full 16-bit word from a stack. A stack is a group of contiguous memory locations whose length is variable up to 32,768 words. A stack is organized on a last-in-first-out basis whereby the last word entered into the stack will be the first word retrieved from the stack.

A stack can start at any address and fills from upper memory toward lower memory (decreasing addresses). The stack instructions themselves do not provide any stack boundary limit testing features. The user must provide boundary limit testing as overhead associated with using stack instructions.

All stack accesses are controlled by a stack pointer for each stack. The stack pointer is a 15-bit word address which points to the most recently accessed location in the stack (this address is referred to as the stack element address -- SEA). The stack pointer may be located anywhere in memory.

Stack instructions occupy two consecutive words in memory. The first word contains the instruction while the second word contains the address of the stack pointer.

2.2.1 Addressing Modes

To provide flexibility in stack management, four addressing modes are provided with stack instructions.

2.2.1.1 Direct Access to Stack

In the Direct Access Mode, the second word of the instruction (the stack pointer address -- SPA) is used to fetch the stack pointer from memory. The stack pointer, in turn, is used to access the stack element for entry, retrieval, or testing of data.

2.2.1.2 Indexed Access to Stack

In the Indexed Access Mode, the stack pointer address in the second word of the instruction is used to fetch the stack pointer from memory. The contents of the X regi-

STACK INSTRUCTION

| OP CODE |
| STACK POINTER ADDRESS |

P

P+1

SPA

STACK POINTER*

SEA

*STACK POINTER ALWAYS
POINTS TO MOST RECENT
ENTRY IN STACK

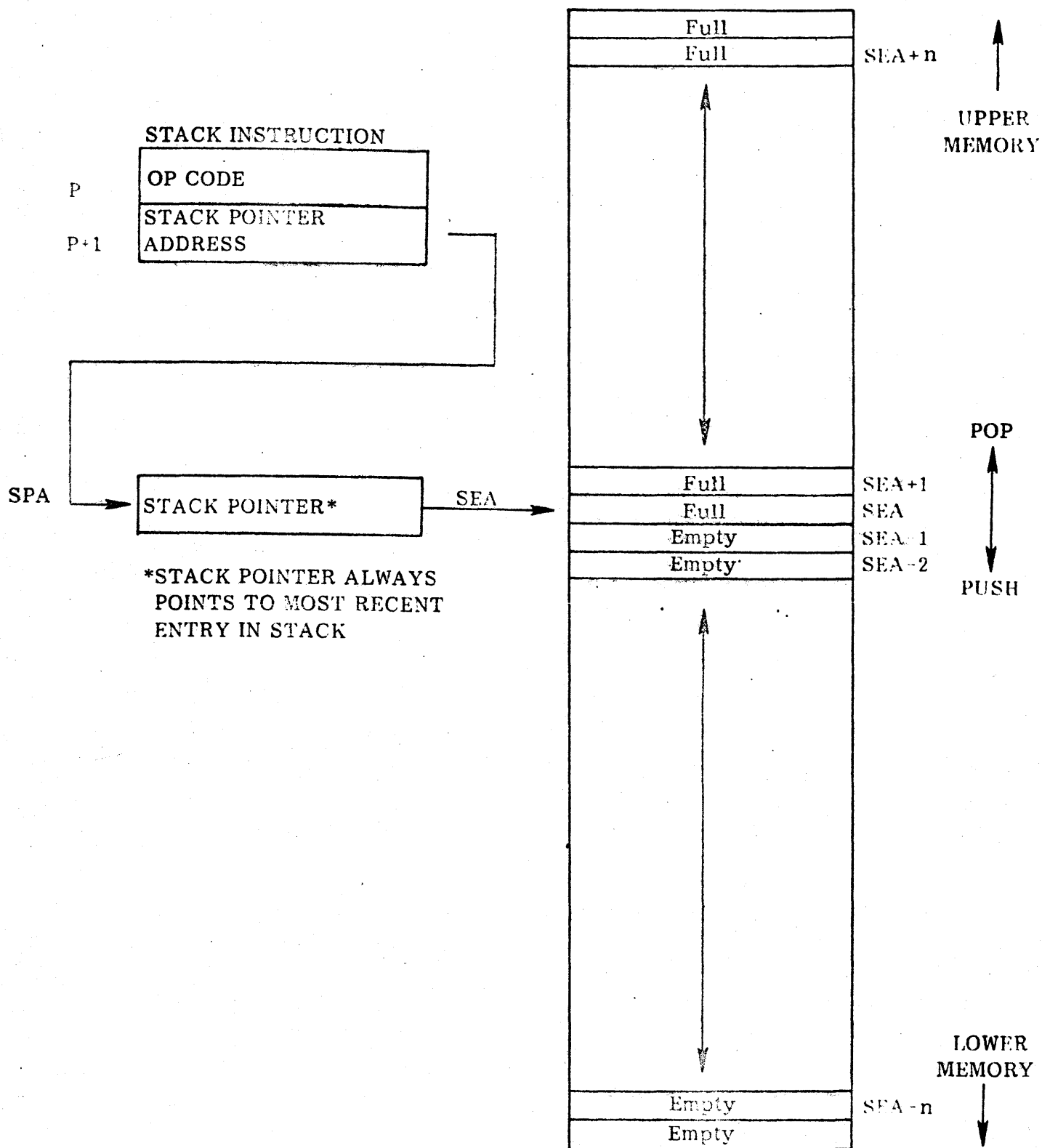| Full | |
Full	SEA + n
Full	SEA+1
Full	SEA
Empty	SEA-1
Empty	SEA-2
Empty	SEA-n
Empty	

UPPER
MEMORY

POP

PUSH

LOWER
MEMORY

Figure 2-1. Stack Organization and Management

2-2

ster (Index register) is then summed with the stack pointer to form the stack element address. This allows access to the nth element in the stack relative to the last stack entry when the X register contains n. For example, if X=0 the most recent stack entry is accessed while if X=1 the next most recent entry is accessed.

2.2.1.3 Auto-Increment Access to Stack (POP)

In the Auto-increment Mode, the stack pointer address is used to obtain the stack pointer. The stack pointer, in turn, provides access to the stack element. Upon completion of the stack access, the stack pointer is incremented and restored to memory. This mode of addressing appears to remove (POP) the most recent entry from the stack when used with a load type instruction.

2.2.1.4 Auto-Decrement Access to Stack (PUSH)

In this mode, the stack pointer is accessed via the stack pointer address, decremented by one and restored. The stack element is then accessed using the decremented stack pointer. This mode of addressing appears to insert (PUSH) a new entry onto the stack when used with a store type instruction.

2.2.2 Assembler Format

The assembler format for Stack instructions is as follows:

[LABEL]   OPCODE   OPERAND [,AM]   [COMMENTS]

The label and comment fields are optional with this class of instruction.

The op code must be present. The legal op codes for Stack instructions are defined in paragraphs 2.2.3 thru 2.2.7, inclusive.

The operand field consists of one or two expressions. The first expression represents a memory word address and must be present. The second expression (AM) is optional and, when included, must be separated from the first by a comma. This expression represents the addressing mode of the Stack instruction. The following is a list of valid expression characters and their associated addressing modes.

These instructions generate two 16-bit words. The first word is the stack instruction op code. The second word is the absolute address of the stack pointer.

Character	Address Mode
No second expression	DIRECT. Stack element is accessed through Stack Pointer. The Stack Pointer is unchanged.
−	PUSH. Stack Pointer is DECREMENTED. Stack element is then accessed through Stack Pointer.
+	POP. Stack element is accessed through Stack Pointer; Stack Pointer is then INCREMENTED.
@	INDEXED. The sum of the Stack Pointer and index register form the effective address of the Stack element to be accessed.

2.2.3 Arithmetic Stack Instructions

ADDS   ADD STACK ELEMENT TO A. Adds the contents of the stack element to the A register. OV is set if arithmetic overflow occurs.

SUBS   SUBTRACT STACK ELEMENT FROM A. Subtracts the contents of the stack element from the A register. OV is set if arithmetic overflow occurs.

2.2.4 Logical Stack Instructions

ANDS   AND STACK ELEMENT TO A. Logically AND's the contents of the stack element with the A register.

IORS   INCLUSIVE OR STACK ELEMENT TO A. Inclusively OR's the contents of the stack element with the A register.

XORS   EXCLUSIVE OR STACK ELEMENT TO A. Exclusively OR's the contents of the stack element with the A register.

2.2.5 Data Transfer Stack Instructions

EMAS   EXCHANGE STACK ELEMENT AND A. Simultaneously stores the contents of the A register in the stack element location and loads the contents of the stack element location into the A register.

LDAS   LOAD STACK ELEMENT INTO A. Loads the contents of the stack element into the A register.

LDXS   LOAD STACK ELEMENT INTO X. Loads the contents of the stack element into the X register.

STAS   STORE A IN STACK ELEMENT. Stores the contents of the A register in the stack element location.

STXS   STORE X IN STACK ELEMENT. Stores the contents of the X register in the stack element location.

2.2.6   Program Transfer Stack Instructions

CMSS   COMPARE AND SKIP IF HIGH OR EQUAL. Compares the contents of the stack element with the A register. If the A register is greater than the contents of the stack element, a one-word skip occurs. If the A register is equal to the contents of the stack element, a two-word skip occurs. If the A register is less than the contents of the stack element, the next sequential instruction is executed.

IMSS   INCREMENT STACK ELEMENT AND SKIP ON ZERO RESULT. The contents of the stack element is incremented by one. If the increment causes the result to become zero, a one-word skip occurs. OV is set if arithmetic overflow occurs.

JMPS   JUMP UNCONDITIONAL. The P counter is loaded with the value of the stack pointer, causing an unconditional branch to the addressed stack element location (SEA). The next instruction is executed from location SEA.

JSTS   JUMP AND STORE. The contents of the P counter (P+2) are stored in the addressed stack element location and the P counter is then loaded with the address of the addressed stack element plus one (SEA+1). The next instruction is accessed from location SEA+1.

2.2.7 Stack Control Instruction

SLAS   STACK LOCATION TO A. Loads the contents of the stack pointer into the A register.

2.3 REGISTER CHANGE INSTRUCTIONS

Five new Register Change instructions have been added to the LSI-2 Processor. All five instructions fall into the Multi-Register Change Subclass (refer to paragraph 3.7.5 in the NAKED MINI/ALPHA LSI SERIES PROGRAMMING REFERENCE MANUAL -- Document 10077-00B1).

2.3.1   Assembler Format

These instructions use the same format and syntax as used by the Register Change instructions discussed in the Programming Reference Manual.

2.3.2   Multi-Register Change Instructions

BCA
BIT CLEAR A. The contents of the X register are ones complemented and then logically ANDed with the contents of the A register. The result replaces A and the original value of X is left unchanged.

BCX
BIT CLEAR X. The contents of the X register are ones complemented and then logically ANDed with the contents of the A register. The result replaces X and the original value of A is left unchanged.

BSA
BIT SET A. The contents of the X register are logically ORed with the contents of A. The result replaces A and X is left unchanged.

BSX
BIT SET X. The contents of the A register are logically ORed with the contents of X. The result replaces X and A is left unchanged.

EIX
EXECUTE INSTRUCTION POINTED TO BY X. The instruction whose address is contained in the X register is executed as though it occupied the location following the EIX instruction. The location following the EIX instruction is skipped during execution of the EIX instruction.

If the executed instruction:

1.   Is a multi-word instruction, the second and succeeding words of the instruction must be located at the second location after the EIX instruction (EIX+2).

2.   Modifies the program counter, the modification is relative to location EIX+1.

3.   Is a SCM or conditional I/O instruction, the location following the EIX instruction (EIX+1) should be coded with a JMP $-1. This is required for recovery purposes in the event of an interrupt or the lack of a true sense response.

Note that EIX is not interruptable.

2.4 ASSEMBLER DIRECTIVES

2.4.1 Machine Directive (MACH)

[LABEL]   MACH   Expression   [COMMENTS]

The MACH directive allows the user to specify which CAI 16-bit computer's instruction set is to be considered valid during this assembly. This allows the assembly, and/or error detection, of programs written for either (or both) LSI (1 or 2) and ALPHA 16 computers. Instructions declared invalid by the MACH directive will be flagged with an "O" error, but will be assembled correctly.

The expression in the operand field must be present, absolute (not relocatable or external), and must be previously defined. The value of the expression will replace the current value in the MACH flag word, remaining in effect until the end of the current assembly or until another MACH directive is encountered. The acceptable values of the MACH directive are shown in table 2-1, below.

The label, if present, will be given the current location counter value.

Table 2-1. MACH Flag Word Values

MACH Value*	Instruction Set Allowed
0	Common subset of ALPHA 16 and LSI only
1	ALPHA 16
2	LSI
3	ALPHA 16 and LSI
4	Extended LSI-2
5	ALPHA 16 and Extended LSI-2
6	LSI and Extended LSI-2
7	ALPHA 16, LSI and Extended LSI-2

*Default value of 2 is assumed if no MACH directive is entered.

MACH directives should appear prior to program instructions.

The common subset of ALPHA 16 and LSI instructions is always allowed.

Section 3

PROGRAMMING EXAMPLES

3.1   INTRODUCTION

This section provides programming examples of how to use the new Stack and Register Change instructions.

3.2   STACK INSTRUCTIONS

The following are examples of Stack instructions:

Example 1 - This example illustrates a save/restore sequence using the Stack capability, allowing convenient coding of re-entrant or recursive routines. This example assumes interrupts were disabled by the JST instruction which caused control to be passed to this routine.

```
SUBR       ENT
           STAS          PTR, -          Push 'A' on Stack
           STXS          PTR, -          Push 'X' on Stack
           SIA                           Get CPU status
           STAS          PTR, -          Push on Stack
           LDA           SUBR            Get return address
           STAS          PTR, -          Push on Stack
           EIN                           Restore interrupts
             .
             .
             .
             .
           SIN           6               Disable interrupts
           LDAS          PTR, +          Pop return
           STA           SUBR            and save
           LDAS          PTR, +          Pop save
           SOA                           and restore
           LDXS          PTR, +          Pop 'X'
           LDAS          PTR, +          Pop 'A'
           JMP           *RTN            Return
```

Example 2 - This example illustrates an indexed stack move of 100 entries from Stack 1 to Stack 2, while simultaneoulsy zeroing Stack 1.

```
                LXP     100             Count to move
LOOP            ZAR                     Zero out buffer 1
                EMAS    PTR1, @         Get data (indexed)
                STAS    PTR2, @         Put data (indexed)
                DXR                     Decrement count and Pointer
                JXN     LOOP            Loop back 99 more
                .
                .
                .
PTR1            DATA    STACK1 - 1      Pointer to Stack 1
PTR2            DATA    STACK2 - 1      Pointer to Stack 2
STACK1          RES     100             Stack 1
STACK2          RES     100             Stack 2
```

3.3 REGISTER CHANGE INSTRUCTIONS

The following are examples of Register Change instructions:

Example 1 - This example shows how a single mask word can be used to set or clear one or more flag bits in a flag word.

Setting Bits

```
                .
                .
        LDX     MASK            Mask bits to X
        LDA     FLAG            Flag word to A
        BSA                     Set bits in flag word
        STA     FLAG            Store new flag word
                .
                .
                .
```

Clearing Bits

```
                .
                .
        LDX     MASK            Mask bits to X
        LDA     FLAG            Flag word to A
        BCA                     Clear bits in flag word
        STA     FLAG            Store new flag word
                .
                .
                .
```

MASK word - Contains "1"s in those bit poistions which are to be set or cleared.

Example 2 - This example illustrates how the EIX instruction could be used in a universal output driver, where the I/O commands of each particular device are contained in tabulor form, i.e., in tables ordered by logical unit number.

```
                                        X contains the character to be output
          .                             A contains the logical unit number
          ADD       IOINST              Add table address
          EAX                           Address to X, character to A
          EIX                           Execute OTA instruction
          JMP       $-1                 Required for conditional I/O
          .
          .

IOINST    DATA      $+1                 I/O Table, ordered by logical unit
          OTA       DAX0, FCX0  ⎫
          OTA       DAX1, FCX1  ⎪
          OTA       DAX2, FCX2  ⎬       Device address and
          .             .    . ⎪       function code for
          .             .    . ⎪       each logical unit.
          .             .    . ⎪
          OTA       DAXn, FCXn  ⎭
```

Appendix A

INSTRUCTION SET IN ALPHABETICAL ORDER

This appendix contains the ALPHA LSI-2 Extended instruction set in alphabetical order by instruction mnemonic. Instructions with variable fields have been appended with an asterisk (*).

Instruction Mnemonic	Instruction Skeleton in Hex	Description
ADDS	1438*	Add Stack Element to A.
ANDS	1418*	AND Stack Element to A.
BCA	06CA	Bit Clear A.
BCX	06C8	Bit Clear X.
BSA	068A	Bit Set A.
BSX	0688	Bit Set X.
CMSS	1658*	Compare and Skip if High or Equal
EIX	0218	Execute Instruction Pointed to by X.
EMAS	14F8*	Exchange Stack Element and A.
IMSS	1678*	Increment Stack Element and Skip on Zero
IORS	1478*	Inclusive OR Stack Element to A.
JMPS	16D8*	Jump Unconditional.
JSTS	16F8*	Jump and Store.
LDAS	14D8*	Load Stack Element into A.
LDXS	1698*	Load Stack Element into X.
SLAS	1618*	Stack Location to A.
STAS	1478*	Store A in Stack Element.
STXS	16B8*	Store X in Stack Element.
SUBS	1458*	Subtract Stack Element from A.
XORS	14B8*	Exclusive OR Stack Element to A.

Appendix B

INSTRUCTION SET IN NUMERICAL ORDER

This appendix contains the ALPHA LSI-2 Extended instruction set in machine code numerical order. For each instruction, reference is made to one of the machine code formats listed below. Instructions with variable fields have been appended with asterisks (*).
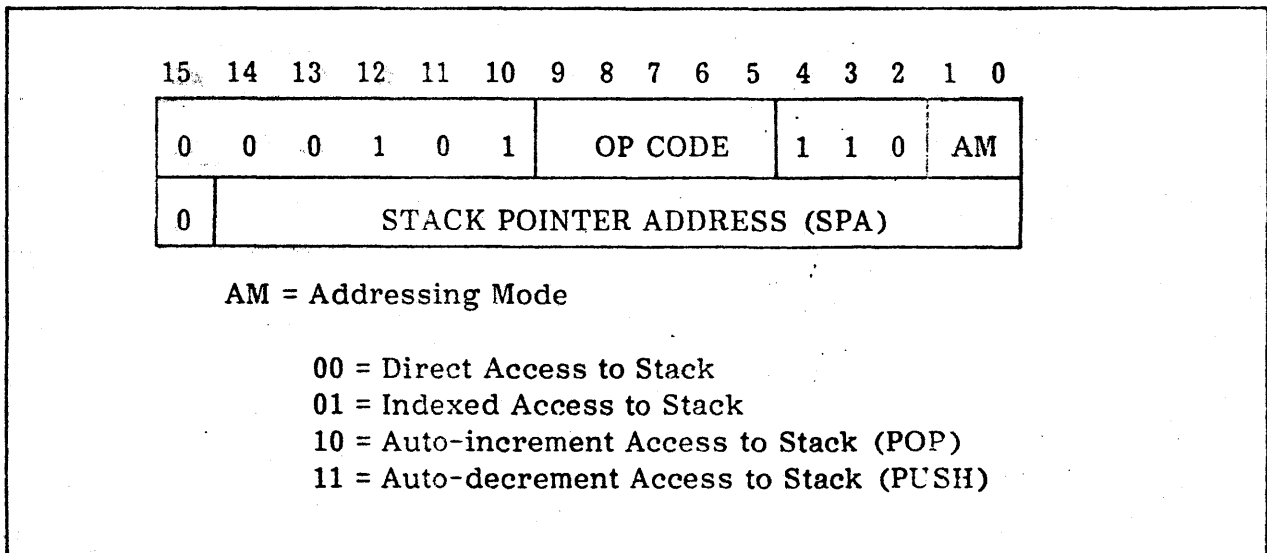
15	14	13	12	11	10	9 8 7 6 5	4	3	2	1 0
0	0	0	1	0	1	OP CODE	1	1	0	AM

0	STACK POINTER ADDRESS (SPA)

AM = Addressing Mode

00 = Direct Access to Stack
01 = Indexed Access to Stack
10 = Auto-increment Access to Stack (POP)
11 = Auto-decrement Access to Stack (PUSH)

Figure B-1. Stack Instruction Machine Code Format

15	14	13	12	11	10 9 8 7 6 5 4 3 2 1 0
0	0	0	0	0	OP CODE

OP CODE = The Register Change Control Code which specifies the source, operation, and location of results.
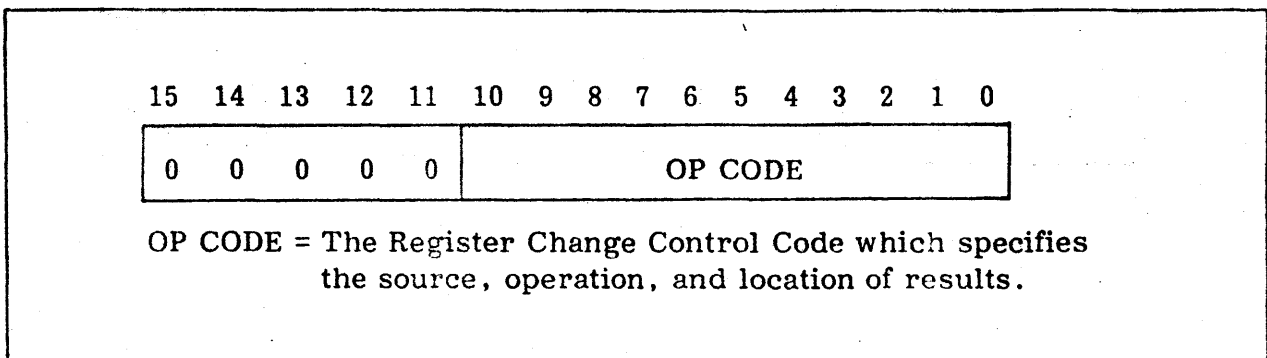
Figure B-2. Register Change Instruction Machine Code Format

Instruction Skeleton in Hex	Instruction Mnemonic	Description	Machine Code Format
0218	EIX	Execute Instruction Pointed to by X.	2
0688	BSX	Bit Set X.	2
068A	BSA	Bit Set A.	2
06C8	BCX	Bit Clear X.	2
06CA	BCA	Bit Clear A.	2
1418*	ANDS	AND Stack Element to A.	1
1438*	ADDS	Add Stack Element to A.	1
1458*	SUBS	Subtract Stack Element from A.	1
1478*	STAS	Store A in Stack Element	1
1498*	IORS	Inclusive OR Stack Element to A.	1
14B8*	XORS	Exclusive OR Stack Element to A.	1
14D8*	LDAS	Load Stack Element into A.	1
14F8*	EMAS	Exchange Stack Element and A.	1
1618*	SLAS	Stack Location to A.	1
1658*	CMSS	Compare and Skip if High or Equal.	1
1678*	IMSS	Increment Stack Element and Skip on Zero.	1
1698*	LDXS	Load Stack Element into X.	1
16B8*	STXS	Store X in Stack Element.	1
16D8*	JMPS	Jump Unconditional.	1
16F8*	JSTS	Jump and Store.	1

Appendix C

ALPHA LSI EXECUTION TIMES

C.1 GENERAL

This appendix defines the execution time of each instruction in the ALPHA LSI-2 extended instruction set. A variety of memories, with varying access times, are offered with the ALPHA LSI-2. The variation in memory access time makes a tabulation of execution times difficult. For this reason time calculation algorithms are provided. These algorithms are useful with any memory access time by making the appropriate memory parameter substitution.

C.2 MEMORY PARAMETERS

Currently, four memories are offered in the ALPHA LSI family, three of these are core memories, while the fourth is a semiconductor memory. Table C-1 lists the parameters of these memories. All times listed are in nanoseconds.

Table C-1. LSI Family Memory Parameters

Memory Type	Configuration	C	RA	RO	WA	WO	M	M'	ROI	WOI
Core 980	Add on 4K, 8K	980	380	600	180	800	600	400	220	420
Core 1200	Add on 16K	1200	400	800	200	1000	600	400	300	500
Core 1600	Add on or integral 4K, 8K	1600	450	1150	250	1350	600	400	0	0
SC 1200	Add on 2K, 4K, 8K Integral 2K, 4K	1200	500	700	200	1000	600	400	0	0

Parameters in nanoseconds are:

C =   Cycle Time
RA =   Read Access
RO =   Read Overhead
WA =   Write Access
WO =   Write Overhead
M =   LSI-1 Effective Read Access
M' =   LSI-1 Effective Write Access
ROI =   Interleaved Effective Read Overhead
WOI =   Interleaved Effective Write Overhead

C.3 LSI-2 EXECUTION TIME ALGORITHMS

The LSI-2 execution time algorithms are listed in table C-2. The algorithms are partitioned by class and subclass. The Stack instruction address calculation times precede the Stack instruction execution algorithms. Note that three different sets of address calculations are provided. The list of Stack instructions have algorithms which list S_1, S_2, or S_3. The appropriate address calculation variable should be used as indicated.

All memories may be overlapped to achieve higher transfer rates. Core 1600 and SC1200 may be overlapped 100 percent to achieve twice the data transfer rate of a single memory module. Core 1200 and Core 980 may be overlapped to achieve a maximum transfer rate of 171 and 163 percent, respectively, of a single memory. Overlapping is always effective for DMA operation.

Overlapping is effective for LSI-2 as indicated by the execution time equations. Terms of the form n/RO or m/WO mean that the larger of the two times indicated are to be used. When overlapping is achieved by alternate memory accesses in different memory modules, the overhead times are masked and the effective RO and WO become zero except for Core 980 and Core 1200 which have an overhead time even when interleaved.

Numerous instructions have several times listed to define variations of an instruction.

Table C-2. LSI-2 Execution Time Algorithms

STACK CLASS			
ADDRESSING MODE	S_1	S_2	S_3
direct access	$3RA + 2(400/RO) + 550/RO$	$S_1 + 100$	$S_1 + 300$
indexed access	$3RA + 2(400/RO) + 850/RO$	$S_1 + 100$	$S_1 + 300$
auto-increment (POP) or auto-decrement (PUSH)	$3RA + 2(400/RO) + 500/RO*$ $+ WA + 400/WO$	$S_1 + 100$	$S_1 + 300$

*not effected by overlap

S_1 is used with ADDS, SUBS, ANDS, IORS, XORS, EMAS, LDAS, LDXS, CMSS and IMSS.

S_2 is used with STAS, STXS, and JSTS.

S_3 is used by JMPS and SLAS.

Table C-2. LSI-2 Execution Time Algorithms (Cont'd)

ARITHMETIC
    ADDS
    SUBS

LOGICAL
    ANDS
    IORS
    XORS

$S_1 + RA + 400/RO$

DATA TRANSFER

LDAS	$S_1 + RA + 400/RO$
LDXS	$S_1 + RA + 400/RO$
STAS	$S_2 + WA + 250/RO$
STXS	$S_2 + WA + 250/RO$
EMAS	$S_1 + RA + 500/RO* + WA + 550/WO$

PROGRAM TRANSFER

JMPS	S_3	
JSTS	$S_2 + WA + 550/WO$	
IMSS	$S_1 + RA + 500/RO* + WA$	
	$+ 700/RO$	$\neq 0$ in line, no skip
	or $+ 1450/RO$	$= 0$ in line, skip
	or $+ 850/RO$	$\neq 0$ interrupt, no echo
	or $+ 1600/RO$	$=$ interrupt, echo
CMSS	$S_1 + RA$	
	$+ 550/RO$	$A < Y$
	or $+ 850/RO$	$A = Y$
	or $+ 1150/RO$	$A > Y$

STACK CONTROL
    SLAS S_3

REGISTER CHANGE CLASS

MULTI-REGISTER CHANGE
    BCA
    BCX
    BSA
    BSX

$RA + 1300/RO$

    EIX $RA + 500/RO +$ normal time of instruction
                                                  executed

C.4 LSI-2 EXTENDED INSTRUCTION SET EXECUTION TIMES

The execution times of the LSI-2 extended instruction set are listed in table C-4. The Stack instruction address calculation times are listed in table C-3.

Table C-3. Stack Instruction Address Calculation Times

MEMORY TYPE	ADDRESSING MODE	S_1	S_2	S_3
CORE 1600	direct access	4.8	4.9	5.1
	indexed access	4.8	4.9	5.1
	auto-increment (POP) or auto-decrement (PUSH)	6.4	6.5	6.7
CORE 1200	direct access	3.6	3.7	3.9
	indexed access	3.65	3.75	3.95
	auto-increment (POP) or auto-decrement (PUSH)	4.8	4.9	5.1
CORE 980	direct access	2.94	3.04	3.24
	indexed access	3.19	3.29	3.49
	auto-increment (POP) or auto-decrement (PUSH)	3.92	4.02	4.22
SC 1200	direct access	3.6	3.7	3.9
	indexed access	3.75	3.85	4.05
	auto-increment (POP) or auto-decrement (PUSH)	4.8	4.9	5.1

S_1 is used with ADDS, SUBS, ANDS, IORS, XORS, EMAS, LDAS, LDXS, DMSS and IMSS.

S_2 is used with STAS, STXS, and JSTS.

S_3 is used by JMPS and SLAS.

Table C-4. LSI-2 Extended Instruction Set Execution Times

MNEMONIC	LSI-2			
	C1600	C1200	C980	SC1200
STACK				

Arithmetic
  ADDS
  SUBS
 	
Logic
  ANDS
  IORS
  XORS
 	
Data Transfer
  LDAS
  LDXS
	$S_1 + 1.6$	$S_1 + 1.2$	$S_1 + 0.98$	$S_1 + 1.2$

STAS STXS	$S_2 + 1.6$	$S_2 + 1.2$	$S_2 + 0.98$	$S_2 + 1.2$
EMAS	$S_1 + 3.2$	$S_1 + 2.4$	$S_1 + 1.96$	$S_1 + 2.4$
Program Transfer CMSS	$S_1 + 1.6$	$S_1 + (1.2 \text{ or } 1.55)$	$S_1 + (0.98 \text{ or } 1.53)$	$S_1 + (1.2 \text{ or } 1.65)$
IMSS	$S_1 + (3.0 \text{ or } 3.3)$	$S_1 + (2.2 \text{ or } 2.85)$	$S_1 + (1.86 \text{ or } 2.61)$	$S_1 + (2.1 \text{ or } 2.85)$
JMPS	S_3	S_3	S_3	S_3
JSTS	$S_2 + 1.6$	$S_2 + 1.2$	$S_2 + 0.98$	$S_2 + 1.2$
Stack Control SLAS	S_3	S_3	S_3	S_3
REGISTER CHANGE				

Multi-Register
  BCA
  BCX
  BSA
  BSX
	1.75	1.7	1.68	1.8

EIX	1.6	1.2	0.98	1.2

BETA ASSEMBLER REFERENCE MANUAL
(Supplement)

LSI-2 EXTENDED INSTRUCTIONS

1.0   INTRODUCTION

The NAKED MINI/ALPHA LSI-2 supports an extended set of instructions not found in the LSI-1 or ALPHA 16 computers. This document describes their operation in the syntax of the BETA assemblers (BETA 4/8 and OMEGA), and assumes the user is familiar with the BETA 4 Assembler Reference Manual (document 96018-00).

These instructions are supported in all BETA Assemblers beginning with version -D0 and are made available through use of the MACH directive (described in the BETA 4 Assembler Reference Manual) as follows:

MACH Value*	Instruction Set Allowed
0	Common subset of ALPHA 16 and LSI only
1	ALPHA 16
2	LSI
3	ALPHA 16 and LSI
4	Extended LSI-2
5	ALPHA 16 and Extended LSI-2
6	LSI and Extended LSI-2
7	ALPHA 16, LSI and Extended LSI-2

* Default value of 2 is assumed if no MACH directive is entered.

MACH directives should appear prior to program instructions.
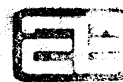
The common subset of ALPHA 16 and LSI instructions is always allowed.

2.0   LSI-2 EXTENDED INSTRUCTIONS

2.1   Instruction Syntax

2.1.1   General

For assembly purposes the LSI-2 extended instructions are divided into two classes.

2.1.2   Instruction Classes

The LSI-2 extended instructions are discussed in this section in a logical
sequence rather than numerical sequence. The instruction classes and
their sequence of discussion are as follows:

Class 10:   Stack, Double Word
Class 5:   Register Change and Control

2.1.3   Syntax Description

This section describes the syntax for each instruction class. In the
following descriptions brackets are used to indicate optional fields.

2.2   Class 10 - Stack, Double Word

2.2.1   General

The combination of post-autoincrement addressing in which the stack
pointer is stepped toward higher memory after the operand address is
determined, and pre-autodecrement addressing in which the stack pointer
is stepped toward lower memory before the operand address is determined,
is the basic requirement for convenient low.overhead stack operations.

The LSI-2 has extensive stack processing capabilities allowing, for
example, the nested handling of interrupts and/or subroutine calls.
Elements in the stacks may be accessed through indexed addressing.
This provides for convenient access of dynamically assigned temporary
storage, especially useful in nested procedures. The stack pointer may
be manipulated without accessing the stack to allow convenient boundary
condition testing.

Stack instructions require two consecutive words of memory, a word
for stack pointer, and one or more consecutive words for the stack
itself. Addressing modes include direct, indexed, pre-autodecrement
(push), and post-autoincrement (pop).

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Word 1	0	0	0	1	0	1	X	0	X	X	X	1	1	0	AM	

Word 2	0	Stack Pointer Address (SPA)

Stack Pointer	0	Stack Element Address (SEA)

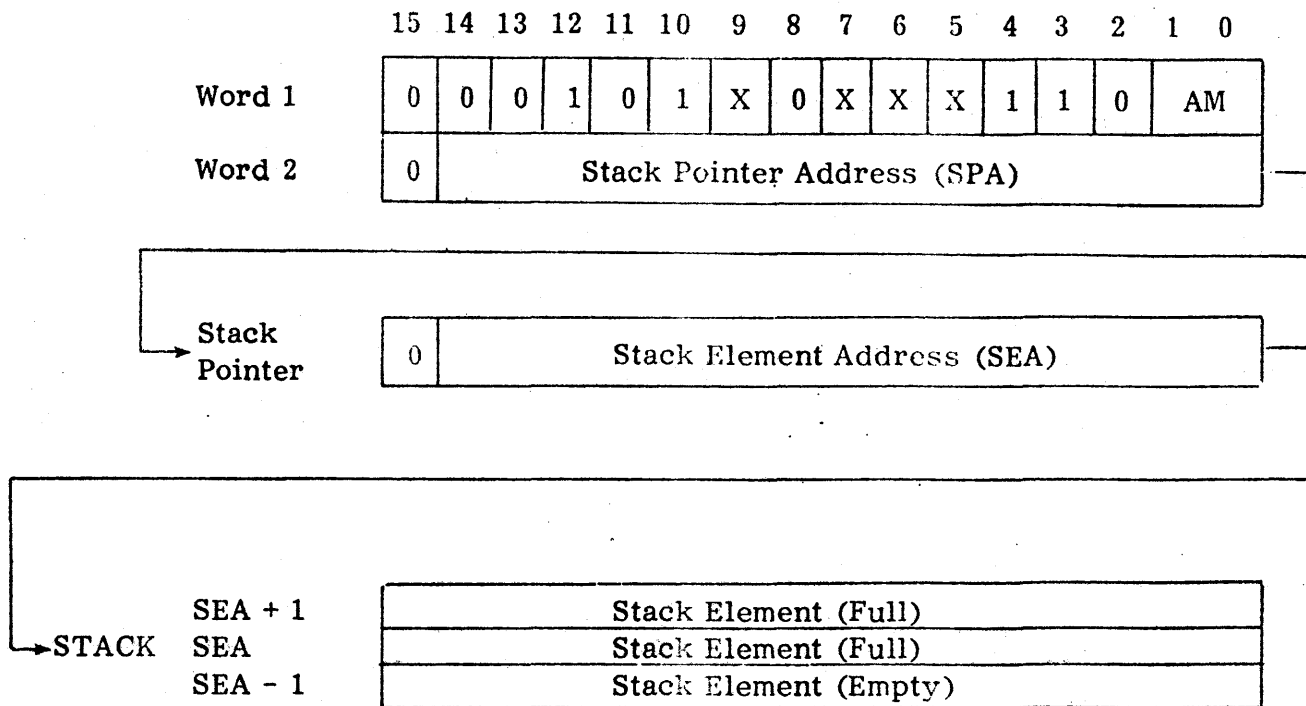|	SEA + 1	Stack Element (Full)
STACK	SEA	Stack Element (Full)
	SEA - 1	Stack Element (Empty)

Figure 1 - Class 10 Machine Language Format

Where: X - Op Code
AM - Addressing Mode
SPA - Stack Pointer Word Address; indirection not allowed.
SEA - Stack Element Word Address; indirection not allowed.

Applicable Addressing Modes (AM) are:

00 - Direct Access to Stack
01 - Indexed Access to Stack
10 - Post-Autoincrement Access to Stack (POP)
11 - Pre-Autodecrement Access to Stack (PUSH)

2.2.2   Assembly Format

Class 10 instruction format for assembly purposes is as follows:

[LABEL]   OP CODE   OPERAND[,AM]   [COMMENT]

2.2.2.1   Label Field. The Label field is optional with Class 10 instructions.

3

2.2.2.2   Operation Code Field. The Operation Code (Op Code) must be present. Legal Op Codes for Class 10 instructions are:

ADDS	Add Stack Element to A
ANDS	Logical AND Stack element with A
CMSS	Compare Stack element and A
EMAS	Exchange Stack element and A
IMSS	Increment Stack element and Skip on zero
IORS	Inclusive OR Stack element with A
JMPS	Jump to Stack element
JSTS	Jump and Store in Stack element
LDAS	Load A from Stack element
LDXS	Load X from Stack element
SLAS*	Stack Pointer to A
STAS	Store A to Stack element
STXS	Store X to Stack element
SUBS	Subtract Stack element from A
XORS	Exclusive OR Stack element with A

*Note: SLAS does not access a Stack element, but the STACK POINTER (SPA)

2.2.2.3   Operand Field. The operand field consists of one or two expressions, the first of which must be present. The first expression represents a memory word address.

The second expression (AM) is optional and, when included, must be separated from the first by a comma. This expression represents the addressing mode of the Stack instruction. The following is a list of valid expression characters and their associated addressing modes.

Character	Address Mode
No second expression	DIRECT. Stack element is accessed through Stack Pointer. The Stack Pointer is unchanged.
-	PUSH. Stack Pointer is DECREMENTED; Stack element is then accessed through Stack Pointer.
+	POP. Stack element is accessed through Stack Pointer; Stack Pointer is then INCREMENTED.
@	INDEXED. The sum of the Stack Pointer and index register form the effective address of the Stack element to be accessed.

4

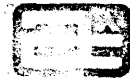2.2.2.4   Comments Field. The comments field is optional.

2.2.2.5   Class 10 Examples. The following are examples of Class 10 instructions:

Example 1 - This example illustrates a save/restore sequence using the
Stack capability, allowing convenient coding of re-entrant
or recursive routines. This example assumes interrupts
were disabled by the JST instruction which caused control
to be passed to this routine.

```
SUBR     ENT
         STAS     PTR,-          Push 'A' on Stack
         STXS     PTR,-          Push 'X' on Stack
         SIA                     Get CPU status
         STAS     PTR,-          Push on Stack
         LDA      SUBR           Get Return Address
         STAS     PTR,-          Push on Stack
         EIN                     Restore Interrupts
          .

          .

         SIN      6              Disable interrupts
                                 during restore
         LDAS     PTR,+          Pop return
         STA      SUBR           and save.
         LDAS     PTR,+          Pop status
         SOA                     and restore.
         LDXS     PTR,+          Pop 'X'
         LDAS     PTR,+          Pop 'A'
         JMP      *RTN           Return
```

Example 2 - This example illustrates an indexed Stack move of 100
entries from Stack 1 to Stack 2, while simultaneously
zeroing Stack 1.

```
         LXP      100            Count to move
LOOP     ZAR                     Zero out buffer 1
         EMAS     PTR1,@         Get data (indexed)
         STAS     PTR2,@         Put data (indexed)
         DXR                     Decrement count and Pointer
         JXN      LOOP           Loop back 99 more
          .

          .
PTR1     DATA     STACK1 - 1     Pointer to Stack 1
PTR2     DATA     STACK2 - 1     Pointer to Stack 2
STACK 1  RES      100            Stack 1
STACK 2  RES      100            Stack 2
```

2.3   Class 5 - Register Change and Control

2.3.1   General

The functions of register change instructions and control instructions
are explained in subsections 3.7 and 3.8 of the BETA 4 Assembler
Reference Manual (document 96018-00).

2.3.2   Assembly Format

The only mandatory field for Class 5 instructions is the Operation Code
Field. The Label and comment fields are optional.

[LABEL]   OP CODE   [COMMENTS]

2.3.2.1   Register Change and Control Op Codes. The following are the extended
Class 5 register change and control instructions which are supported on
LSI-2.

BCA   Bit Clear A. The contents of the X register
               are logically complemented and then ANDed
               with A. The original value of X is left un-
               changed and the result is left in A.

BCX   Bit Clear X. The contents of the X register
               are logically complemented and then ANDed
               with A. The original value of A is left un-
               changed and the result is left in X.

BSA   Bit Set A. The contents of the X register
               are logically ORed with A. The original
               value of X is left unchanged and the result
               is left in A.

BSX   Bit Set X. The contents of the A register
               are logically ORed with X. A is left un-
               changed and the result is left in X.

EIX   Execute Instruction pointed to by X. The instruction whose address is contained in the X register is executed as though it occupied the location following the EIX instruction. The location following the EIX instruction is skipped during execution of the EIX instruction.

If the executed instruction:

1. Is multi-word instruction, the second and succeeding words of the instruction must be located at the second location after the EIX instruction (EIX+2).

2. Modifies the program counter, the modification is relative to location EIX+2.

3. Is a SCM or conditional I/O instruction, the location following the EIX instruction (EIX+1) should be coded with a JMP $-1. This is required for recovery purposes in the event of an interrupt or the lack of a true sense response.

Note that EIX is not interruptable.

2.3.2.2   **Class 5 Examples.** The following are examples of Class 5 instructions:

Example 1 - This example shows how a single mask word can be used to set or clear one or more flag bits in a flag word.

Setting Bits

.
.
.

LDX	MASK	Mask bits to X
LDA	FLAG	Flag word to A
BSA		Set bits in flag word
STA	FLAG	Store new flag word

.
.
.

Clearing Bits

```
        .
        .
    LDX     MASK        Mask bits to X
    LDA     FLAG        Flag word to A
    BCA                 Clear bits in flag word
    STA     FLAG        Store new flag word
        .
        .
```

MASK word - Contains "1"s in those bit positions which are to be set
or cleared.

Example 2 - This example illustrates how the EIX instruction
could be used in a universal output driver, where the I/O
commands of each particular device are contained in tabular
form, i.e., in tables ordered by logical unit number.

```
                            X contains the character to be output
                            A contains the logical unit no.
    ADD     IOINST          Add table address
    EAX                     Address to X, character to A
    EIX                     Execute OTA instruction
    JMP     $-1             Required for conditional I/O
        .
        .
        .

IOINST  DATA    $+1             I/O Table, ordered by logical unit
        OTA     DAX0, FCX0
        OTA     DAX1, FCX1
        OTA     DAX2, FCX2
        .        .   .
        .        .   .
        .        .   .
        OTA     DAXn, FCXn
```