AD-A060 256
BOLT BERANEK AND NEWMAN INC CAMBRIDGE MASS
PLURIBUS DOCUMENT 4: BASIC SOFTWARE.(U)
SEP 78 M F KRALEY
F/G 9/2
UNCLASSIFIED
BBN-3001
DCA200-77-C-0616
NL
1 OF 2
AD
A060256

AD A060256

(12)

LEVEL II

basic-software

DDC FILE COPY

D D C
RECEIVED
OCT 23 1978
F

This document has been approved for public release and sale; its distribution is unlimited.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) PLURIBUS DOCUMENT 4: BASIC SOFTWARE		5. TYPE OF REPORT & PERIOD COVERED Technical rept.
		6. PERFORMING ORG. REPORT NUMBER BBN Report No. 3001
7. AUTHOR(s) M. F. Kraley et al.		8. CONTRACT OR GRANT NUMBER(s) DCA200-C-616
9. PERFORMING ORGANIZATION NAME AND ADDRESS Bolt Beranek and Newman Inc. 50 Moulton Street Cambridge, Massachusetts 02138		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS DCA200-77-C-0616
11. CONTROLLING OFFICE NAME AND ADDRESS Defense Communications Agency Washington, D.C. 20305		12. REPORT DATE September 1978
		13. NUMBER OF PAGES 148
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) Defense Commercial Communications Office Scott Air Force Base Illinois 62225		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Distribution Unlimited

146p.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

- multiprocessor
- Pluribus
- reliable computer
- parallel processor
- computer architecture
- fault tolerant computation
- multiprocessor design

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The Pluribus is a reliable, expandable, high bandwidth line of multi-resource computers originally developed for use as a switching node in the ARPA computer network. It can be configured with arbitrary amounts of memory and I/O tailored to suit the application; it is designed to survive failures and continue operation without human intervention even while repairs are in progress. This report, one of a set of nine volumes documenting the Pluribus line, specifies the instruction set of the processor, introduces the assembly language currently used, and provides a manual for the system debugging program.

DD FORM 1473 1 JAN 73 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

060 100 LB

Report No. 3001 Bolt Beranek and Newman Inc.

PLURIBUS DOCUMENT 4: BASIC SOFTWARE

December 1975

Update edition of September 1978

Sponsored by:

Defense Communications Agency
Contract No. DCA200-C-616

ACCESSION for
NTIS White Section ☑
DDC Buff Section ☐
UNANNOUNCED ☐
JUSTIFICATION
BY
DISTRIBUTION/AVAILABILITY CODES
Dist. AVAIL. and/or SPECIAL
A

78 10 12 034

PLURIBUS DOCUMENT 4: BASIC SOFTWARE

PREFACE

"Pluribus Document 4: Basic Software" is one of a series which taken together provides complete documentation of the Pluribus line of computer systems. In the present document, Part 1 specifies the instruction set of the processor. Part 2, entitled "Introduction to Assembly Language," introduces the assembly language currently used. Part 3 is a manual for DDT, the system debugging program.

PRECEDING PAGE NOT FILMED BLANK

TABLE OF CONTENTS

PREFACE

Part 1: Processor Instruction Set . . . . Instruction Set

Part 2: Introduction to Assembly Language . . . . Intro to Assembly Language

Part 3: DDT . . . . DDT

PRECEDING PAGE NOT FILMED BLANK

PLURIBUS DOCUMENT 4: BASIC SOFTWARE

PART 1: PROCESSOR INSTRUCTION SET

Instruction Set

PRECEDING PAGE NOT FILMED BLANK

Update History:

Reproduced with the permission of the Lockheed Electronics Company from the _SUE Processor Instruction Set_, edition of May 1973, copyright Lockheed Electronics Co., Inc.

SUE is a trademark of Lockheed Electronics Company.

Instruction Set

SUE PROCESSOR INSTRUCTION SET

GENERAL SYSTEM BULLETIN G3

Third Edition

This bulletin supercedes
SUE Processor Instruction Set
General System Bulletin G3, Rev. A
dated June 1972

Bulletin GB13020009103
May 1973

©Copyright 1973 by Lockheed Electronics Company
Los Angeles, California All rights reserved

821

EFFECTIVE PAGES

New pages introduced in this third edition include Processor Instruction Sets for SUE 1110A/B, 1111A/B, 1112A/B, and Appendices D and E.

Changes in the second edition, which included Processor Instruction Set SUE 1110, and Appendices A through C, are indicated by a heavy line in the outer margin of the changed page.

CONTENTS

Title Page

SUE 1110 INSTRUCTION SET

Introduction 1
Word Formats 2
Data Words 2
Address Words 3
Instruction Words 4
Fields 5
Addressing 6
Byte-Word Addressing 6
Absolute and Relative Addressing 6
Extended Addressing 6
Indexing 7
Auto Incrementing and Decrementing 7
Indirect Addressing 7
Register, Immediate and Literal Operands 8
Combination Addressing Modes 8
Special Addresses 10
Status Indicators 11
Instruction Descriptions 12
General Register Instructions 12
General Operations 12
General Register Instruction Word Formats 13
General Register Instruction Times 18
Branch Conditional Instructions 20
Branch Conditions 21
Branch Instruction Word Formats 22
Branch Instruction Times 23

Instruction Set

CONTENTS (continued)

Title	Page
Shift Instructions	24
Shift Instruction Word Formats	25
Shift Instruction Timing	27
Control Instructions	28
Control Instruction Word Formats	28
Control Instruction Times	33
Unimplemented Instructions	34
Input/Output Instructions	34
SUE 1110A INSTRUCTION SET	
Introduction	35
Store Key Instruction	35
SUE 1110B INSTRUCTION SET	
Introduction	37
Fetch and Clear Instructions	37
Fetch and Clear Operation	38
SUE 1111A INSTRUCTION SET	
Introduction	39
Temporary Storage	39
Instruction Format	40
Decimal Data Format	41
Character Data Format	41
Symbolic Coding for Operands	42
Instructions	42
SUE 1111B INSTRUCTION SET	
Introduction	46
Fetch and Clear Instructions	46
Fetch and Clear Operations	46

CONTENTS (continued)

Title	Page
SUE 1112A INSTRUCTION SET	
Introduction	49
Double Precision Data Format	49
Instruction Times	50
Bit Manipulation Instructions	50
Bit Manipulation Instruction Formats	51
Bit Manipulation Operations	52
Move Instructions	52
Move Instruction Format	52
Move Operations	53
Normalize and Count Instructions	53
Normalize and Count Instruction Format	53
Normalize and Count Operations	53
Double Length Shift Instructions	56
Double Length Shift Instruction Format	57
Double-Length Shift Operations	57
Class B Instruction Set	60
Class B Instruction Format	60
Accumulator Registers	60
Single-Precision Fixed Point Instructions	60
Addressing Modes	60
One-Word Operand Format	61
Single Precision Fixed- Point Operations	61
Double Precision Fixed-Point Instructions	62
Addressing Modes	62
Double Precision Fixed Point Operations	63
Control Instructions	63
Control Instruction Formats	64

Instruction Set

CONTENTS (continued)

Title	Page
SUE 1112B INSTRUCTION SET	
Introduction	65
Fetch and Clear Instructions	65
Fetch and Clear Operation	66
APPENDIX A, INSTRUCTION TIMES	
Single Shift Instruction Timing for SUE 1110 (Basic), 1110A/B, 1111A/B, 1112A/B	A-3
SUE 1111A/B CLASS C INSTRUCTION TIMES DECIMAL AND CHARACTER INSTRUCTIONS	
Decimal Add and Subtract Timing	A-5
Decimal Shift Timing	A-10
Move Timing	A-11
Compare-Field Timing	A-11
Decimal Compare Timing	A-13
APPENDIX B, INSTRUCTION SUMMARY AND INDEX	
SUE 1110 (Basic) Instructions Summary	B-1
SUE 1110 (Basic) Instruction Index	B-2
APPENDIX C, INPUT/OUTPUT ADDRESSES	
APPENDIX D, SELF-INTERRUPT AND SYSTEM INTERRUPT EXECUTIVE SPACE	
APPENDIX E, USASCII CHARACTER SET AND HEXADECIMAL CODES	

LIST OF TABLES

Table	Title	Page
1	Combination Addressing Modes	9
2	Special Addresses	10
3	General Register Instruction Word Formats	14
4	SUE 1110 (Basic) General Register Instruction Times	19
5	Branch Instruction Times	24
6	SUE 1110 (Basic) Control Instruction Times	33
A-1	SUE 1110 (Basic) General Register Instruction Times	A-1
A-2	SUE 1110A/B, 1111A/B, and 1112A/B General Register Instruction Times	A-2
A-3	SUE 1110 (Basic), 1110A/B, 1111A/B and 1112A/B Control Instruction Times	A-3
A-4	SUE 1110 (Basic) Branch Instruction Times	A-4
A-5	SUE 1110A/B, 1111A/B and 1112A/B Branch Instruction Times	A-4
A-6	Decimal Shift Timing Chart	A-10
A-7	SUE 1112A/B Instruction Times	A-18
A-8	SUE 1112A/B Single- and Double-Precision Fixed-Point Instruction Times	A-20
C-1	Input-Output Device Addresses	C-1

Instruction Set

PREFACE

This bulletin contains instructions to program seven types of SUE processors:

	Number of Instructions
SUE 1110 (basic)	108
SUE 1110A	109
SUE 1110B	111
SUE 1111A, Decimal Arithmetic	118
SUE 1111B, Decimal Arithmetic	120
SUE 1112A, Scientific Double Precision	144
SUE 1112B, Scientific Double Precision	146

SUE 1110 basic is the first instruction set described in this bulletin. SUE 1110A performs the basic instruction set and one additional instruction, Store Key (SKEY). Both SUE 1111A and 1112A processors have the speed and capabilities of SUE 1110A, and each has an extended instruction set. Descriptions of these extended instructions follow the description of the SUE 1110B. Instruction times for all instructions are summarized in Appendix A.

Processors SUE 1110B, 1111B, and 1112B perform the same instructions as the respective A-series processors, and two additional instructions Fetch and Clear Word (FCLW), and Fetch and Clear Byte (FCLB). These two instructions can be used in multiprocessor systems as a synchronizing mechanism.

Instructions in this bulletin are described in machine language for the system user possessing a background in digital computer terminology and operation. Additional information on the basic instruction set is contained in the LAP-2 Assembler manual. Operation and maintenance of SUE processors is contained in the respective reference and maintenance bulletins designated by the processor model number.

SUE 1110 INSTRUCTION SET

INTRODUCTION

SUE 1110 instruction set includes 108 basic instructions exclusive of 16 addressing modes. Many of these instructions operate on either 16-bit data or 8-bit byte formats. Other instructions test one or more of the 16 status indicator bits. This bulletin presents a detailed description of word formats, addressing modes, and status indicators followed by a definition of each instruction operation.

The 108 instructions are divided into eleven classes according to type of instruction function. Seven of these classes are grouped as general register instructions. They contain arithmetic, logical, move, compare and test functions that involve the eight general registers of the processor. Two classes represent the branch instructions. They contain unconditional and conditional branch functions on the true or false condition of status indicators. The shift class contains full 15-bit shift capabilities with eight different operations and two address modes. The control class contains system control functions such as load/store of all general registers, load/store of status indicators and control of interrrupt operations.

The eleven instruction classes are:

Class Code	Description
1	Accumulator to Memory with Auto Decrement
2	Accumulator to Memory with Auto Increment
3	Accumulator to Memory
4	Data to Accumulator, Jump to Subroutine, Jump, and Register to Register
5	Memory to Accumulator with Auto Decrement
6	Memory to Accumulator with Auto Increment
7	Memory to Accumulator

Instruction Set

8	Branch False and No Operation
9	Branch True and Unconditional
A	Shift
0	Control

Class codes are specified in the instruction word format by the four-bit C field. (Fields are defined later under instruction words in this bulletin). Five class codes are not defined for the basic instruction set. They have been reserved for specification of additional general purpose instructions in the SUE 1111A, B and 1112A, B Processors; or, for special purpose instructions in future SUE processors with expanded ROM control memories.

SUE 1110 Processor contains eight, 16-bit general registers including the program counter. Seven of these registers may be used as accumulators or index registers. The arithmetic-logic unit processes 16-bit operands but memory data may be 8-bit bytes or 16-bit words.

Memory addresses are 16-bit numbers that select up to 60k (k = 1024) bytes. Addresses 60k to 64k are used to directly address registers within system modules other than program memory modules.

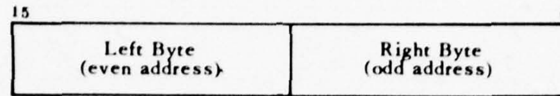
WORD FORMATS

Bit positions within a word are numbered right to left starting with 0. Bit 0 is the least significant bit of the word and bit 15 is the most significant.

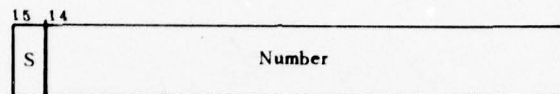
DATA WORDS

Two data word formats can be processed, an 8-bit byte and a 16-bit word. The most significant bit (15) represents the algebraic sign of numeric data. A ONE in bit position 15 represents a negative number, and a ZERO represents a positive number. Negative numbers are in twos complement form.

Byte Format



Word Format



S	Sign
1	negative (-)
0	positive (+)

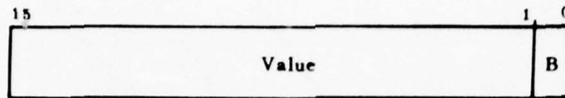
Instruction Set

In byte operations, the entire selected 16-bit register is used in the operation with the byte operand. In register-to-memory instructions (byte mode), the right byte of the register operates on the designated byte in memory. In memory-to-register instructions (byte mode), the designated byte in memory operates on the full 16-bit register as though the memory operand has a left byte equal to ZERO attached to it. In either type of operation, arithmetic operations occur in a 16-bit register and carry and overflow are detected out of a 16-bit register.

ADDRESS WORDS

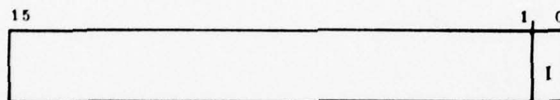
The 16-bit address represents a byte address. Bit zero selects the left or right byte of a 16-bit word. On word addresses, bit zero is used to specify more than one level of indirect addressing.

Byte Address



Bit 0 -	Byte
0	left
1	right

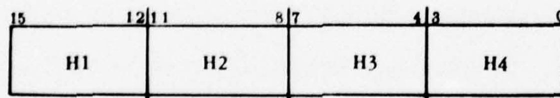
Word Address



Bit 0 -	Addressing
0	address direct
1	indirect

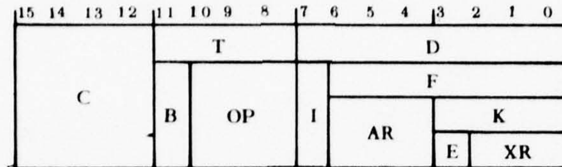
INSTRUCTION WORDS

Instruction words are constructed to facilitate encoding and decoding of the machine language code. The words are defined so that the fields of the instruction do not overlap the four hexadecimal digits represented by H1, H2, H3, and H4. Those fields that are subsets of a hexadecimal digit are right-justified, with the high-order bit used to indicate the less common condition.



Digit	
H1	Class Designation 0 through 15
H2	Operation Designator (usually)
H3	Accumulator Designator (usually)
H4	Index Designator (usually)

FIELDS. - A variety of word formats are interpreted by the processor. All of the fields used, and their positions, are defined below in a composite drawing. Functions of a given field may vary according to the instruction.



NOTE: Several fields have more than one function depending on the instruction that contains them. In the field definitions below, any function common to several instructions is defined. For descriptions of other functions, refer to corresponding instruction descriptions.

Instruction Set

Field Definitions

Symbol	Description
C	Class Indicator (4 bits) - Specifies 1 of 16 classes or divisions of the instruction set. Classes indicate the type of function.
T	Test Operation (4 bits) - Defines operation codes for the Control and Branch classes.
D	Displacement Address (8 bits) - Direct address (+ or -) to words relative to the address of the instruction. May be expressed as P (Program Counter) +D, where D is the range, -128 through +127. An exception, if the absolute-address mode of a Control instruction is specified, then D directly addresses the first 256 words in memory.
B	Byte Indicator (1 bit) - Specifies whether the memory operand is a word (B=0) or a byte (B=1). Field of a Control instruction specifies the Relative (B=1) or Absolute (B=0) address mode.
OP	Operation Indicator (3 bits) - Defines 1 of 8 operations available to certain classes. Several classes use the same set of operations, as explained in greater detail under Instruction Descriptions.
I	Indirect Addressing Indicator (1 bit) - Specifies first level of indirect addressing if I=1.
AR	Accumulator Register Designator (3 bits) - Designates 1 of 8 general registers as an A-Register during instruction execution.
E	Extended-Address Indicator (1 bit) - Indicates (when 1) that the word following the instruction will be accessed as an extended-address part of the instruction.
XR	Index Register Designator (3 bits) - Designates 1 of 7 general registers as an X-Register during instruction execution.
K	Constant (4 bits) - Designates length of a Shift command, or an immediate constant. Also used to enable interrupts.
F	Status Bit-Pattern (7 bits) - Comprises the bit pattern for changing control states for certain Control instructions.

ADDRESSING

SUE 1110 Processor develops a 16-bit operand address based on the mode that is selected by the instruction class code and other fields of the instruction word format.

BYTE-WORD ADDRESSING

A bit (B) in the instruction word specifies if the operand is to be a byte (8 bits) or a word (16 bits) in general register instructions. If B = 1 and bit zero of the effective operand address is ZERO, the left byte (bits 15 through 8) is used; the right byte (bits 7 through 0) is used if bit zero is ONE. If B = 0, a word operand is requested and the address of the word is treated as an even-numbered byte address.

ABSOLUTE AND RELATIVE ADDRESSING

Branch instructions use the relative displacement method to develop the branch address. The D field of the instruction is an 8-bit (7 bits plus sign) number that specifies a branch within +127 or -128 words (not bytes) from the current location. Negative numbers are represented in twos complement form.

Control instructions use the relative displacement as well as the absolute addressing modes. Bit B of the instruction word, when set to a ONE, selects the relative mode and, when ZERO, selects the absolute mode. In the absolute address mode the D field of the instruction is an 8 bit number that specifies direct address of the first 256 words (not bytes) of memory.

EXTENDED ADDRESSING

When the E bit of the instruction is a ONE, the word following the instruction becomes the base address and is used to develop the operand address. When E = 0 the base address is ZERO.

INDEXING

Content of one of the seven general registers may be selected as an index register. The XR field of the instruction selects the register. When the XR field is all zeros, no indexing is specified. If neither extended addressing nor indexing is called for (i.e. bits 3-0 all ZEROs) then no address is specified and an unimplemented instruction trap is generated.

Two types of indexing are used:

> Base Relative Indexing (indexing relative to the base address of the computer or user program). – In this type of indexing, the index register contains the complete address of the desired memory location. Base relative indexing together with autoincrement or autodecrement provide generalized push down and pop up stack processing capabilities.
>
> Table Indexing (indexing relative to the base address of a table). – In this type of indexing the index register contains the variable n to fetch the quantity located at TABLE + n.

AUTO INCREMENTING AND DECREMENTING

Within the general register instructions, separate class codes are used to provide the option for automatic increment or decrement of the index register selected by the XR field of the instruction. When autodecrement is specified, the content of the index register is decremented before the operand address is generated. When autoincrement is specified, the content of the selected index register is incremented after the operand address is generated.

ONE is subtracted or added to the content of the index register when the instruction specifies a byte operand with autodecrement or autoincrement. TWO is subtracted or added when the instruction specifies a word operand with autodecrement or autoincrement.

INDIRECT ADDRESSING

If indirect bit I of the instruction is set to a ONE, the address developed by the processor points to the address of the operand.

Instruction Set

Multi-level indirect addressing is provided in the word mode only. The processor tests the least significant bit of the indirect address. If this bit is a ONE, and the word mode is specified, the word pointed-to is also treated as an indirect address. If the least significant bit of the address is a ZERO, the processor stops the multi-level indirect addressing for this instruction. If the processor counts up to 16 levels of indirect addressing, an unimplemented instruction self-interrupt is generated and the instruction is trapped.

Only single level indirect addressing is available in the byte mode because the least significant bit of the operand address specifies left or right byte.

REGISTER, IMMEDIATE AND LITERAL OPERANDS

The data-to-accumulator (class code 4) general instruction provides for selection of register, literal or immediate operands. The register operand is the register specified by the XR field, and can be the program counter if XR = 0. The literal operand may be the 16-bit word following the instruction or the 16-bit word following the instruction plus the contents of XR. An immediate operand is the 4-bit value in the instruction's K field.

COMBINATION ADDRESSING MODES

In most general register instructions, combinations of addressing modes may be specified to yield fourteen useful functions for memory operand selection. The processor develops addresses in combinations of the following in the sequence shown:

Extended Address
Autodecrement the Index
Indexed
Indirect
Autoincrement the Index

Autodecrement and autoincrement functions apply to the contents of the general register selected by the XR field of the instruction.

On autodecrement the content of the index register is decremented by one for byte addresses or by two for word addresses before the index register contents is used as an index value. On autoincrement the content of the index register is incremented by one or two after it is used as an index value.

If the XR field of an instructions is all ZEROs, no indexing is specified. However, auto-increment or auto-decrement specified with a ZERO XR field affects the program counter.

Table 1 contains a summary of the fourteen combinational addressing modes.

Instruction Set

Table 1. Combination Addressing Modes

Address Mode	M Effective Address	XR Index Register	Assembler Mnemonic
Extended	A	-	A
Extended, Indexed	A + X	-	A(R)
Extended, Indexed, Autoincrement	A + X	X + e	A(R+)
Extended, Autodecrement, Indexed	A + X - e	X - e	A(-R)
Indexed	X	-	(R)
Indexed, Autoincrement	X	X + e	(R+)
Autodecrement, Indexed	X - e	X - e	(-R)
Extended, Indirect	[A]	-	*A
Extended, Indexed, Indirect	[A + X]	-	*A(R)
Extended, Indexed, Autoincrement, Indirect	[A + X]	X + e	*A(R+)
Extended, Autodecrement, Indexed, Indirect	[A + X - e]	X - e	*A(-R)
Indexed, Indirect	[X]	-	*(R)
Indexed, Autoincrement, Indirect	[X]	X + e	*(R+)
Autodecrement, Indexed, Indirect	[X - e]	X - e	*(-R)

NOTES: A - 16-bit word following instruction
X - Content of General register selected by XR field
e - A ONE if byte address, a TWO if word address
[ ] - 16-bit word at address specified in brackets.

SPECIAL ADDRESSES

Even addresses 61,440 to 65,534 (hexadecimal F000 to FFFE) are reserved for addressing of system hardware registers within SUE system modules. The odd numbered addresses in this range are not used. Each system module is assigned a set of even (word) addresses as shown in table 2.

Addressing a system register for either a read or write function is allowed by master modules. The slave module always transmits or receives 16 data bits. If the selected register is less than 16 bits in length, the data is transmitted in the least significant bit positions and the most significant, unused, bit positions are ZEROs.

Table 2. Special Addresses

Addresses (Hexadecimal)	Module Assignment
F000-F7FE	Reserved for special memory assignments
F800	I/O Device Controller #1, Status Register
F802	I/O Device Controller #1, BTA Address Register
F804	I/O Device Controller #1, BTA Block Length Register
F806	I/O Device Controller #1, Control Register
F808	I/O Device Controller #1, Data Register
F80A-F80E	Reserved for I/O Device Controller #1
F810-F81E	I/O Device Controller #2 as in #1
F820-FAFE	Reserved for I/O Device Controllers as in #1. (see Appendix C)
FB00-FBFE	Auto Load Memory
FC00-FEFE	Reserved for Auto Load
FF00	Central Processor (#0) Register 0, (Program Counter)
FF02-FF0E	Central Processor (#0), General Registers 1-7
FF10	Central Processor (#0), Status Indicators
FF12	Central Processor (#0), Instruction Register
FF14-FF1C	Reserved for Central Processor #0
FF1E	Central Processor (#0), Control Flip-Flops
FF20-FF3E	Processor #1, same set as #0
FF40-FF5E	Processor #2, same set as #0
FF60-FF7E	Processor #3, same set as #0
FF80	Control Panel #1 Address Register-Attention Interrupt
FF82	Control Panel #1 Data Register
FF84-FF86	Control Panel #2 as in #1
FF88-FF8A	Control Panel #3 as in #1
FF8C-FF8E	Control Panel #4 as in #1
FF90-FFFF	Reserved for other System Modules to be assigned.

STATUS INDICATORS

SUE 1110 Processor has a 16-bit status indicator register. Status indicators may be affected by execution of general register and shift instructions. This is indicated by their symbol in INSTRUCTION DESCRIPTIONS. The status indicators may also be set or reset with special control instructions.

The status bit position within the status register, symbol, name, and description are as follows:

Bit	Symbol	Name and Description
0	EQ	Equal – In a compare operation, the source operand equals the target operand.
1	GT	Greater-Than – In a compare operation, the source operand is greater than the target operand.
2	OV	Overflow – Set during Add, Subtract, or Arithmetic Left Shift if the Carry out of bit 15 is different than the Carry in to bit 15. If the set condition is not caused, V remains unchanged.
3	CY	Carry – Receives the Carry out of bit 15 during an Add, Subtract, Arithmetic Left Shift, or Left Linked Shift. Reset during an Arithmetic Right Shift. Receives bit 0 shifted out from a Right Linked Shift.
4	F1	Flags 1, 2, or 3 – Programmable flag bits.
5	F2	
6	F3	
7	LP	Loop Complete – Set if content of register selected by XR field equals ZERO at the completion of an Autoincrement or Autodecrement instruction. Reset if content of XR is NOT ZERO.
8	OD	Odd – For all general register instructions except Compare, the Odd indicator receives the least significant bit of the result.
9	ZE	Zero – For all general register instructions except Compare, set if the result is ZERO and reset if NOT ZERO.
10	NG	Negative – Receives the most significant bit of the result of any general register instruction except Compare.
11	A	Active – Indicates that the processor is executing instructions. A is set unless the processor is quiescent.
12	M1	Interrupt Mask – Bits M1 through M4 correspond to system interrupts 1 through 4. When any bit is set or reset, respectively, the Bus Controller is requested to ignore or allow interrupt requests for the corresponding vector.
13	M2	
14	M3	
15	M4	

INSTRUCTION DESCRIPTIONS

GENERAL REGISTER INSTRUCTIONS

Class codes 1 through 7 specify the general register instructions. They are all two-operand instructions with one set of eight general operations. In the definitions of these operations, the terms target (T) and source (S) are used. The target is the register or memory cell to be modified, the source is the register or memory cell used as an operand that is to remain unchanged.

GENERAL OPERATIONS. – The OP field of the instruction selects the operation for each class of general register instruction as follows:

OP Code (Hexadecimal	Operation	Description	Status Indicators Affected
0	MOVe	Transfer the source operand to the target operand. (S) → (T)	NG, ZE, OD
1	SUBtract	Subtract the source operand from the target operand and store the result in the target operand. -(S) + (T) → (T)	CY, OV, NG, ZE, OD
2	ADD	Form the sum of the source (S) and target (T) operands and store in (T). (S) + (T) → (T)	CY, OV, NG, ZE, OD
3	AND	Form the logical product of the source and target operands and store the result in the target operand. (S).AND. (T) → (T)	NG, ZE, OD
4	Inclusive OR	Form the logical sum of the source and target operands and store in the target operand. (S).OR. (T) → (T)	NG, ZE, OD

5	Exclusive OR	Form the logical difference of the source and target operands and store in the target operand. (S). EOR. (T) → (T)	NG, ZE, OD
6	CoMPare	Compare logical, the source operand to the target operand. Register contents and memory contents are not affected.	GT, EQ

	GT	EQ
If (S) < (T)	0	0
If (S) = (T)	0	1
If (S) > (T)	1	0

NOTE

Bit 15 of each word is considered a magnitude bit, not a sign bit. The compare result is unsigned based on the 16-bit magnitude.

7	TeST	Form the logical product of the source and target operands. Register and memory contents are not affected.	NG, ZE, OD

If (S) . AND. (T) = 0, SET ZE, RESET NG, OD

If (S) . AND. (T) ≠ 0, RESET ZE

If (S) . AND. (T) is odd, SET OD (odd implies bit 0 is set)

If (S) . AND. (T) is negative, SET NG (negative implies bit 15 is set)

GENERAL REGISTER INSTRUCTION WORD FORMATS. – The instruction word formats used for the general register instructions is shown in table 3.

Table 3. General Register Instruction Word Formats

General Register Classes*		H_1 15 14 13 12	H_2 11	10 9 8	H_3 7	6 5 4	H_4 3	2 1 0
Accumulator To Memory	Auto Decrement	C=1	B	OP	I	AR	E	XR
	Auto Increment	C=2	B	OP	I	AR	E	XR
		C=3	B	OP	I	AR	E	XR
Jump to Subroutine		C=4	0	0	I	AR	E	XR
Jump		C=4	0	0	I	0	E	XR
Data to Accumulator	Literal/Register	C=4	1	OP	0	AR	E	XR
	Immediate Data	C=4	1	OP	1	AR	K	
Memory To Accumulator	Auto Decrement	C=5	B	OP	I	AR	E	XR
	Auto Increment	C=6	B	OP	I	AR	E	XR
		C=7	B	OP	I	AR	E	XR

NOTES:

C — Class Codes 1-7

OP — Operation Code:

0	MOV	Move
1	SUB	Subtraction
2	ADD	Addition
3	AND	Logical Product
4	IOR	Logical Inclusive OR
5	EOR	Logical Exclusive OR
6	CMP	Compare
7	TST	Test

B — Word when 0, Byte when 1

I — Indirect when 1

AR — Accumulator Register designator (0-7)

E — Extended or two-word instruction when 1

XR — Index Register designator (0-7), no indexing when 0

K — 4-bit Immediate data constant

For E=0 and XR≠0, XR provides the entire operand address. If no index register is selected (XR=0), and E=0, an instruction trap occurs, except for class 4, Register, where the PC is the source operand.

For E=1, XR=0, the next word provides the entire operand address. If E=1 and XR ≠0, indexing operation is specified. In this case, the content of (XR) is added to the next word to produce the effective address of the memory operand or an indirect address.

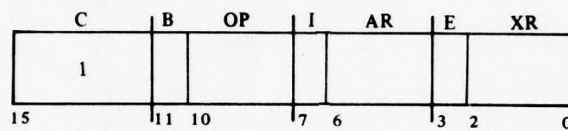
*SUE 1112 Instruction Set contains more instructions in class code 4.

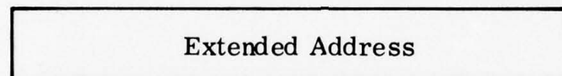
The following additional symbols are used in the instruction definitions:

( )	Contents of
M	effective operand address
PC	Program Counter, general register 0.
P	Current instruction address

General Register Instruction Definitions

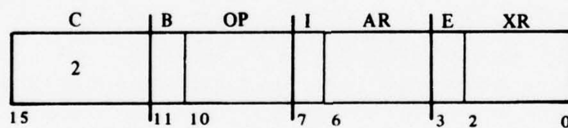
ACCUMULATOR TO MEMORY, AUTO DECREMENT

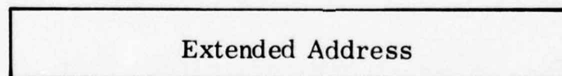




(AR) operates on (M). (XR) is decremented before use.

ACCUMULATOR TO MEMORY, AUTO INCREMENT

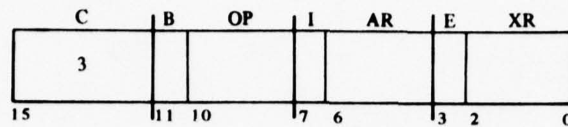


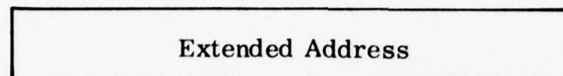


(AR) operates on (M). (XR) is incremented after use.

Instruction Set

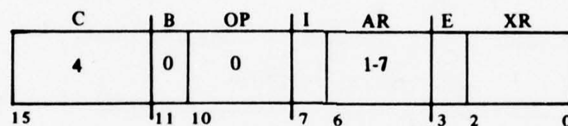
ACCUMULATOR TO MEMORY

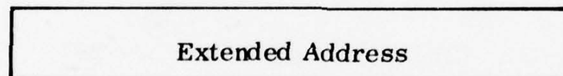




(AR) operates on (M). (XR) is not affected.

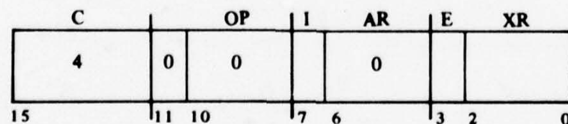
JUMP TO SUBROUTINE JSBR

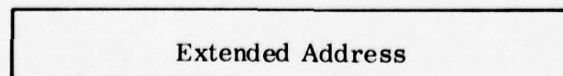




P + 2 replaces (AR) for E = 0 and P +4 replaces (AR) for E = 1. (M) operates on PC (content of general register 0). Thus, the return address is stored in AR and PC is set to the jump-location address.

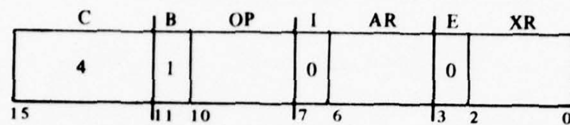
JUMP JUMP





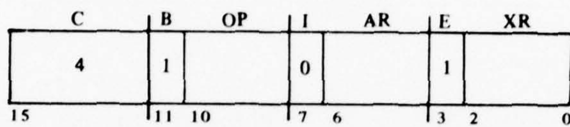
(M) operates on PC, setting it to the jump-location address. The jump function is the same as a MOV (M) to PC, but does not affect status indicators.

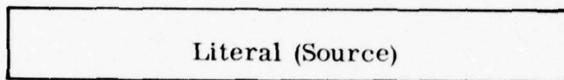
DATA TO ACCUMULATOR, INDEX REGISTER



A register-to-register instruction. (XR) operates on (AR).

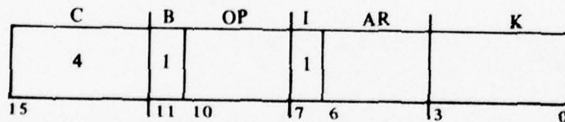
DATA TO ACCUMULATOR, LITERAL





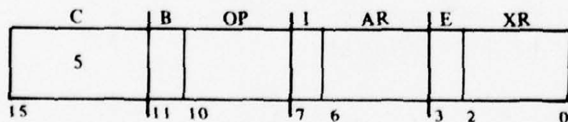
The word following the instruction is the literal source operand. It operates on (AR). If XR is not 0, then (XR) is added to the literal before operating on (AR).

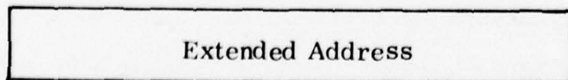
DATA TO ACCUMULATOR, IMMEDIATE



K operates on (AR). K is the 4-bit immediate constant operand.

MEMORY TO ACCUMULATOR, AUTO DECREMENT

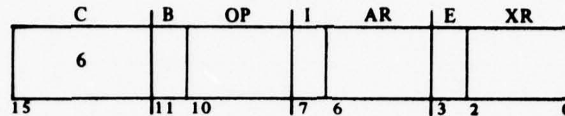


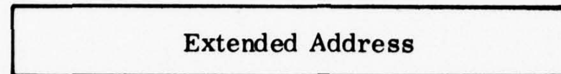


(M) operates on (AR). (XR) is decremented before use.

Instruction Set

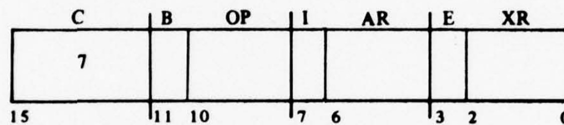
MEMORY TO ACCUMULATOR, AUTO INCREMENT

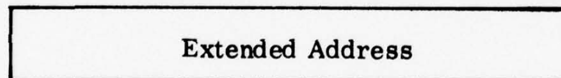




(M) operates on (AR). (XR) is incremented after use.

MEMORY TO ACCUMULATOR





(M) operates on (AR). (XR) is not affected.

GENERAL REGISTER INSTRUCTION TIMES. - Instruction execution times depend on:

- Operand addressing modes
- General operation code
- Program memory access and cycles
- INFIBUS availability

Table 4 contains a summary of typical general register instruction times assuming the INFIBUS is available to the processor and a SUE 3311 Core Memory is used for instruction and data storage. A memory cycle time of 850 nanoseconds, read access time of 750 nanoseconds, and a write access time of 550 nanoseconds is used. Access is the total time to access both the bus scheduler and memory. Microprogram steps of 160 nanoseconds are used for arithmetic operations and 130 nanoseconds for non-arithmetic operations.

Table 4. SUE 1110 (Basic) General Register Instruction Times

General Instruction		Time (Microseconds)		
		Indexed	Auto-Increment	Auto-Decrement
ACCUMULATOR TO MEMORY	Class Codes	3	2	1
Logical: MOV, AND, IOR, EOR Op Codes: 0 3 4 5		3.94	4.81	4.81
Arithmetic: SUB, ADD Op Codes: 1 2		4.03	4.90	4.90
Compare: CMP Op Code: 6		3.70	4.57	4.57
Test: TST Op Code: 7		3.35	4.22	4.22
Address Modes: For Extended, add 0.13 For Indirect, add 1.14 for first level, add 1.01 for each additional level For Extended, Indirect, add 1.40 for first level, add 1.01 for each additional level				
JUMP, JUMP TO SUBROUTINE	Class Code	4	-	-
Instruction: JUMP, JSBR Op Code: 0, AR = 0, AR ≠ 0		2.79	-	-
Address Modes:				
For Extended, add 0.06		2.85	-	-
For Indirect, add 1.14 for first level, add 1.01 for each additional level		3.93	-	-
For Extended, Indirect add 1.33 for first level, add 1.01 for each additional level		4.12	-	-
DATA TO ACCUMULATOR	Class Code	4	-	-
Logical: MOV, AND, IOR, EOR Op Codes: 0 3 4 5	Register to Register or Immediate	2.50	-	-
Arithmetic: SUB, ADD Op Codes: 1 2		2.79	-	-
Compare: CMP Op Code: 6		2.69	-	-
Test: TST Op Code: 7		2.50	-	-
Address Modes: For Literal add 0.68 For Literal Indexed add 0.84				
MEMORY TO ACCUMULATOR	Class Codes	7	6	5
Logical: MOV, AND, IOR, EOR Op Codes: 0 3 4 5		3.35	4.09	4.09
Arithmetic: SUB, ADD Op Codes: 1 2		3.64	4.38	4.38
Compare: CMP Op Code: 6		3.67	4.41	4.41
Test: TST Op Code: 7		3.35	4.09	4.09
Address Modes: For Extended add 0.13 For Indirect add 1.14 for first level, add 1.01 for each additional level For Extended, Indirect add 1.40 for first level, 1.01 for each additional level				

NOTE: All times are in microseconds.

Instruction Set

To compute the actual instruction execution time, it is necessary to add the time increments shown in Table 4 for each selected addressing mode. The minimum times shown in the table assume an indexed addressing mode. A more complete table of general instruction times is given in Appendix A.

For example, an ADD register-to-register instruction requires 2.79 microseconds with the SUE core memory. An ADD memory-to-accumulator instruction requires 3.64 microseconds when the operand address is held in an index register. If the address is located in the next word location (extended instruction mode), the time is 3.77 microseconds. Indexing the extended address does not add time to the instruction. Indirect addressing adds 1.14 microseconds for the first level and 1.01 for each subsequent level.

BRANCH CONDITIONAL INSTRUCTIONS

Thirteen conditions can be tested by branch conditional (TRUE or FALSE) instructions. Each condition can be tested to produce a branch or a fall-through to the next instruction for either state (TRUE for class code 9 and FALSE for class code 8). The condition status is determined by testing the status indicators and programmable flags affected by the last operation.

BRANCH CONDITIONS. – Following is a list of the 13 branch conditions and their meaning when TRUE.

T Field	Condition	Symbol	Meaning (TRUE Condition)
0	Unconditional	UN	The branch is made unconditionally.
1	Equal	EQ	The latest compare operation found the two operands to be equal to each other.
2	Greater-Than	GT	The latest compare operation found the source operand to be greater than the target operand.
3	Overflow	OV	An add, subtract, or shift operation produced a result outside of the range $-2^{15} \leq R \leq +(2^{15} - 1)$ since overflow was last reset.
4	Carry	CY	The latest add, subtract, or shift operation produced a carry out of the most significant end of the arithmetic unit.
5	Flag 1	F1	These three programmable flags can be set or reset by a set or reset status indicator instruction.
6	Flag 2	F2	
7	Flag 3	F3	
8	Loop Complete	LP	This indicator is set if the result of the latest autoincrement or autodecrement of any index register equals zero; otherwise it is reset.
9	Odd	OD	The result of the latest general operation (except compare), or shift operation is an odd number (Bit 0 = 1).
A	Zero	ZE	The latest general operation (except compare), or shift operation results in all zeros.
B	Negative	NG	Result of the latest general operation (except compare), or shift operation is a negative number (Bit 15 = 1).
C	Less-Than	LT	In the latest compare operation, the source operand was less than the target operand.
D, E, F			Cause an unimplemented instruction trap.

Instruction Set

BRANCH INSTRUCTION WORD FORMATS. -

	H1 (15 14 13 12)	H2 (11 10 9 8)	H3 (7 6 5 4)	H4 (3 2 1 0)
No Operation	C= 8	0		
Branch Unconditional	C= 9	0	D	
Branch False	C= 8	T	D	
Branch True	C= 9	T	D	

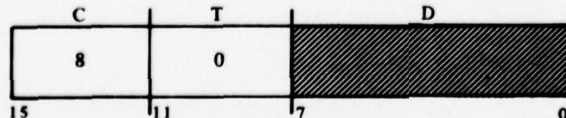
D Displacement word address in twos complement form.

T The T-Field specifies each Branch test. That is, which processor status indicator (if any) is to be tested. A list of each indicator, the corresponding value for T, and the operator assembler-mnemonics follows (z = T for true and F for false):

T (hexadecimal)	Indicator	Assembler Mnemonic (xx)
1	Equal	BEQz
2	Greater Than	BGTz
3	Overflow	BOVz
4	Carry	BCYz
5	Flag 1	BF1z
6	Flag 2	BF2z
7	Flag 3	BF3z
8	Loop Complete	BLPz
9	Odd	BODz
A	Zero	BZEz
B	Negative	BNGz
C	Less Than	BLTz
D, E, F,	(Instruction is trapped)	

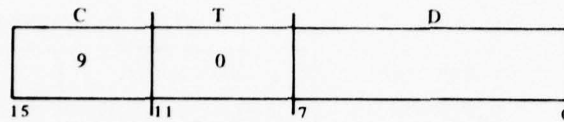
Branch Instruction Definitions

NO OPERATION NOPR



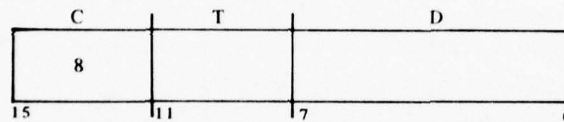
A one-word NO-OP which does not affect status indicators.

BRANCH UNCONDITIONAL BRUN



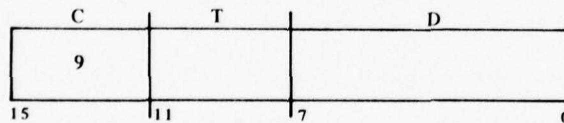
An unconditional (no testing) branch is made to the relative address specified by D. PC + 2 x D replaces PC. D is in twos complement form with sign extended to represent a 16 bit number.

BRANCH FALSE BxxF



A Branch is made to the relative address specified by D if the indicator specified by T is false, or 0; otherwise, the next instruction in sequence is accessed.

BRANCH TRUE BxxT



A Branch is made to the relative address specified by D if the indicator specified by T is true, or 1; otherwise, the next instruction in sequence is accessed.

BRANCH INSTRUCTION TIMES. - Branch instruction execution times depend on whether or not the branch occurs, or the next instruction in sequence is executed. The branch-on-less-than operation (T = C) has different timing than the branch on other status bits. The branch instruction times are shown in Table 5.

Table 5. Branch Instruction Times

Instruction	Assembler Mnemonic	Time (microseconds)			
		Next Word		Branch	
		1110 Basic	All Others**	1110 Basic	All Others**
No Operation	NOPR	1.78	1.75	-	-
Branch Unconditional	BRUN	-	-	2.72	2.82
Branch True	BxxT*	1.78	1.75	2.72	2.82
Branch False	BxxF*	1.78	1.75	2.72	2.82
Branch Less Than True	BLTT	1.75	1.75	3.08	3.21
Branch Less Than False	BLTF	1.88	1.88	3.08	3.21

*where xx = EQ, GT, OV, CY, F1, F2, F3, LP, OD, ZE, NG.
**includes Processors 1110A and B, 1111A and B, and 1112A and B.

SHIFT INSTRUCTIONS

Class code A (hexadecimal) specifies a shift instruction. Up to 15 bit-position shifts may be specified in a single shift instruction. Two formats are provided to allow an option on the location of the shift count. When bit 7 of the instruction is a ZERO, the least significant four bits of the general register selected by XR, contains the shift count. When bit 7 is a ONE, the K field of the instruction word specifies the shift count.

SHIFT INSTRUCTION WORD FORMATS. -

Two single-word formats are used. The formats illustrated are for the shift count defined by (XR) or K, respectively.

	H_1				H_2				H_3				H_4			
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Shift Indexed	A_{16}				0	OP			0	AR			0	XR		
Shift Immediate	A_{16}				0	OP			1	AR			K			

AR Accumulator Register designator (to be shifted).
K Shift Count
XR Shift Count Source Register.
OP Shift Operation Code:

OP	Bits 10	9	8	Operation
0	0	0	0	Single Left Arithmetic Open
1	0	0	1	Single Left Logical Linked
2	0	1	0	Single Left Logical Open
3	0	1	1	Single Left Logical Closed
4	1	0	0	Single Right Arithmetic Open
5	1	0	1	Single Right Logical Linked
6	1	1	0	Single Right Logical Open
7	1	1	1	Single Right Logical Closed

NOTE

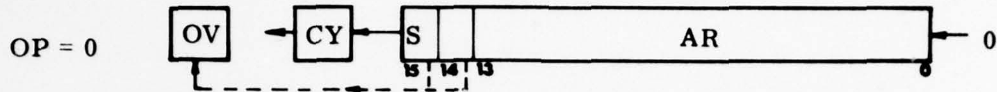
SUE 1112 Processor provides double length shifts and also normalize instructions in addition to these basic single shifts.

Instruction Set

Shift Instruction Definitions

SINGLE LEFT ARITHMETIC OPEN SLAO

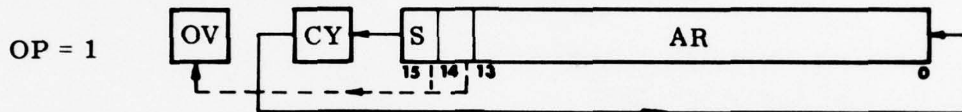
OP = 0



(AR) bits are shifted left out of $(AR)_{15}$ to the carry (CY) and zeros are shifted to $(AR)_0$. If any $(AR)_{14}$ bit is different than $(AR)_{15}$ preceding a shift, then the overflow indicator, OV, is set. Operation affects status indicators: CY, OV, NG, ZE, OD.

SINGLE LEFT LOGICAL LINKED SLLL

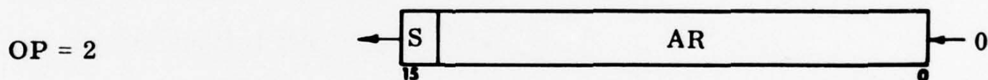
OP = 1



Carry (CY) is shifted into $(AR)_0$ and $(AR)_{15}$ is shifted into CY. If any $(AR)_{14}$ bit is different than $(AR)_{15}$ preceding a shift, then the overflow indicator, OV, is set. Operation affects status indicators: CY, OV, NG, ZE, OD.

SINGLE LEFT LOGICAL OPEN SLLO

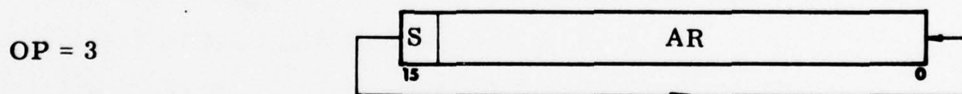
OP = 2



(AR) is shifted left. For each bit shifted, $(AR)_{15}$ is lost and $(AR)_0$ equals 0. Operation affects status indicators: NG, ZE, OD.

SINGLE LEFT LOGICAL CLOSED SLLC

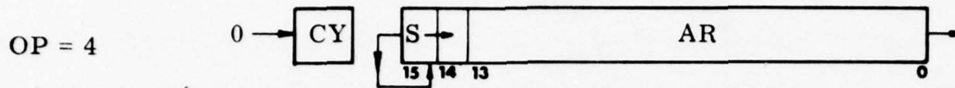
OP = 3



(AR) is shifted left. $(AR)_{15}$ is shifted into $(AR)_0$. Operation affects status indicators; NG, ZE, OD.

SINGLE RIGHT ARITHMETIC OPEN SRAO

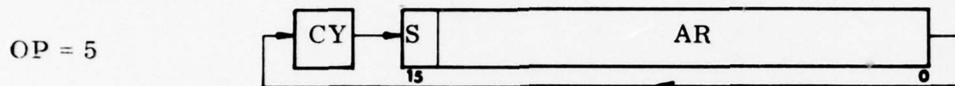
OP = 4



(AR) is shifted right. $(AR)_{15}$, the sign bit, remains the same and is shifted into $(AR)_{14}$. $(AR)_0$ bits shifted out are lost. Carry (CY) is reset. Operation affects status indicators: CY, NG, ZE, OD.

SINGLE RIGHT LOGICAL LINKED SRLL

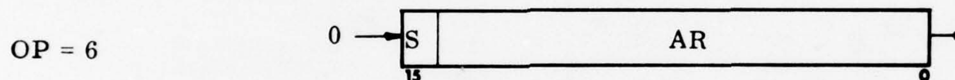
OP = 5



Carry (CY) is shifted into $(AR)_{15}$, and $(AR)_0$ is shifted into CY. Operation affects status indicators: CY, NG, ZE, OD.

SINGLE RIGHT LOGICAL OPEN SRLO

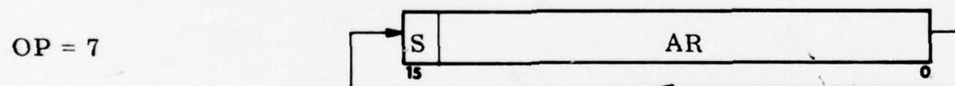
OP = 6



(AR) is shifted right. For each bit shifted, $(AR)_0$ is lost and $(AR)_{15}$ equals 0. Operation affects status indicators: NG, ZE, OD.

SINGLE RIGHT LOGICAL CLOSED SRLC

OP = 7



(AR) is shifted right. $(AR)_0$ is shifted into $(AR)_{15}$. Operation affects status indicators: NG, ZE, OD.

SHIFT INSTRUCTION TIMING. - Shift instruction execution times depend on the number of single bit shifts (N) specified in either the K field (immediate) or the selected register, XR. The time is calculated by the formula:

$$T_s = 2.76 + (0.26)N$$

where N = 0, 1, ..., 15.

Instruction Set

CONTROL INSTRUCTIONS

Class code 0[1] specifies a group of instructions that provide control of processor operation in a system. The instructions provide control of system interrupts, and storing and restoring status indicators and general registers.

CONTROL INSTRUCTION WORD FORMATS. -

Formats: H1 (15 14 13 12)	H2 (11 10 9 8)	H3 (7 6 5 4)	H4 (3 2 1 0)	Assembler Mnemonic	Instructions
C= 0	0			HALT	Halt
C= 0	2	0 F		RSTS	Reset Programmable Status Indicators
C= 0	2	1 F		SETS	Set Programmable Status Indicators
C= 0	8	0	K	ENBL	Enable Interrupts
C= 0	8	4	K	ENBW	Enable and Wait
C= 0	8	8	K	DSBL	Disable Interrupt
C= 0	8	C	K	DSBW	Disable and Wait
C= 0	8	4/C	0	WAIT	Wait
C= 0	B 1	D		STSM	Status to Memory
C= 0	B 3	D		REGM	Registers to Memory
C= 0	B 4	D		RETN	Return from Interrupt
C= 0	B 5	D		MSTS	Memory to Status
C= 0	B 7	D		MREG	Memory to Registers

Notes:

Shaded areas are ignored.

F - Programmable status bits to be reset (RST) or set (SET) (bit positions correspond with status register).

K - Interrupt mask bits to be reset (ENBL) or set (DSBL) (corresponding mask bits in status register).

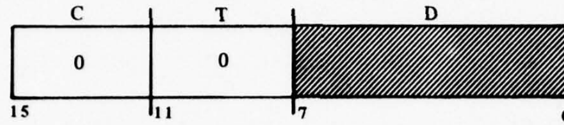
B - Address mode, absolute when 0 or relative when 1.

D - Address field (words), twos complement form for relative.

[1]More instructions in class code 0 are described under SUE 1110A and B, and 1112A and B Instruction Sets.

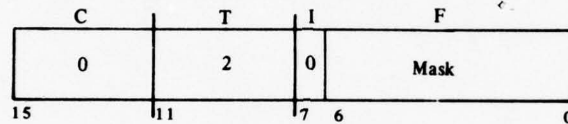
Control Instruction Definitions

HALT HALT



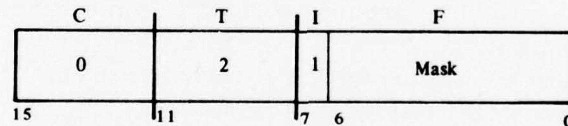
Further instruction execution ceases. Execution resumes if the RUN switch on the control panel is pressed or if RUN is enabled by another processor. Return from a HALT is to the next instruction in sequence. Interrupts cause no resumption.

RESET PROGRAMMABLE STATUS INDICATORS RSTS



F is a mask for resetting status indicators. For each corresponding bit of F and the least significant seven bits of the status indicator word, if F is ONE, the indicator is reset to ZERO.

SET PROGRAMMABLE STATUS INDICATORS SETS



F is a mask for setting status indicators. For each corresponding bit of F, and the least significant seven bits of the status indicator word, if F is ONE, the indicator is set to ONE.

F Bits for RSTS and SETS

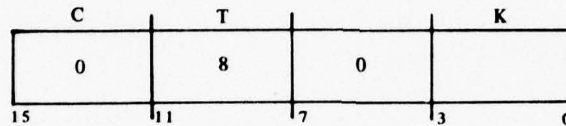
F bits set	Status Indicators
0	EQ EQUAL
1	GT GREATER THAN
2	OV OVERFLOW
3	CY CARRY
4	F1 FLAG 1
5	F2 FLAG 2
6	F3 FLAG 3

Instruction Set

NOTE

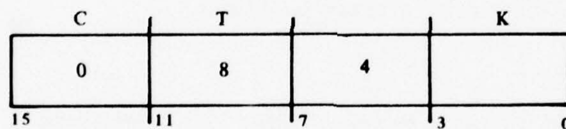
In the following five interrupt control instructions, the K field (bits 3 through 0) corresponds to status register bits 15 through 12 that enable or disable interrupts 4 through 1, respectively.

ENABLE INTERRUPTS ENBL



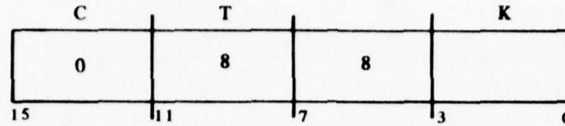
Each ZERO in the K field is ignored, each ONE in the K field enables the corresponding interrupt.

ENABLE and WAIT ENBW



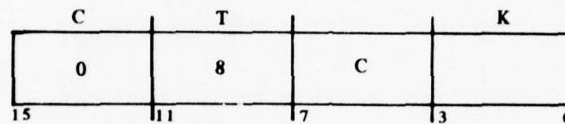
Each ZERO in the K field is ignored, each ONE in the K field enables the corresponding interrupt. The processor enters the WAIT state until an enabled interrrupt occurs. If the enabled interrupt is 4, the interrupt is processed in the normal manner. If the enabled interrupt is 1, 2, or 3, execution continues at the next instruction in sequence. If no interrupts are enabled an instruction trap occurs.

DISABLE INTERRUPTS DSBL



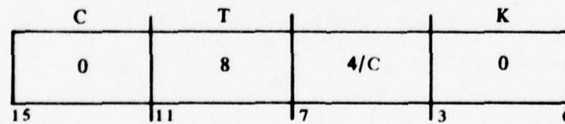
Each ZERO in the K field is ignored, each ONE causes an interrupt disable for the corresponding interrupt.

DISABLE and WAIT DSBW



Each ZERO in the K field is ignored, each ONE in the K field disables the corresponding interrupt. The processor enters the WAIT state until a non-disabled interrupt occurs. If the enabled interrupt is 4, the interrupt will be processed in the normal manner. If the interrupt is 1, 2, or 3, execution continues at the next instruction in sequence. If no interrupts are enabled, an instruction trap occurs.

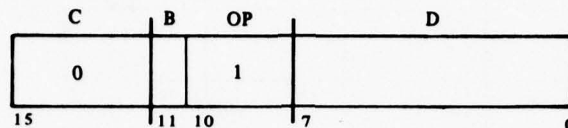
WAIT WAIT



When the K-field is zero, no interrupt masking takes place, and the processor enters the WAIT state until an interrupt occurs. If the interrupt is level 4, it will be taken. The normal return after a level 4 interrupt is to the WAIT instruction.

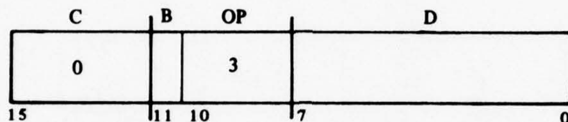
If the interrupt level is 1, 2, or 3, execution continues at the next instruction in sequence. If no interrupts are enabled, an instruction trap occurs.

STATUS TO MEMORY STSM



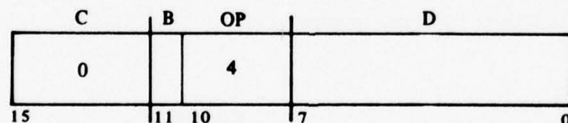
The content of the status indicator register replaces M. Relative (B = 1) or absolute (B = 0) addressing is used to determine M.

REGISTERS TO MEMORY REGM



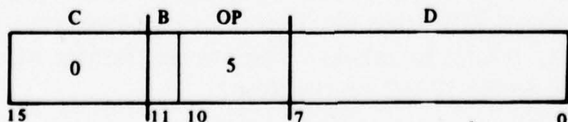
The general registers 1, 2, 3, 4, 5, 6, and 7 are stored into memory in words M, M + 2, ..., M + 12. Relative (B = 1) or absolute (B = 0) addressing is used to determine M.

RETURN FROM INTERRUPT RETN



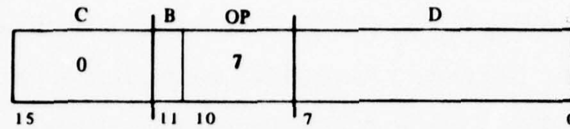
(M) replaces the status indicator register, and M + 2 replaces the Program Counter, PC. Relative (B = 1) or Absolute (B = 0) addressing is used to determine M.

MEMORY TO STATUS MSTS



(M) replaces the content of the status indicator register. Relative (B = 1) or absolute (B = 0) addressing is used to determine M.

MEMORY TO REGISTERS MREG



General registers 1, 2, 3, 4, 5, 6, and 7 are loaded from memory, from words M, M + 2, ..., M + 12. Relative (B = 1) or absolute (B = 0) addressing is used to determine M.

CONTROL INSTRUCTION TIMES. – Table 6 contains a list of the instruction execution times for control instructions.

Table 6. SUE 1110 (basic) Control Instruction Times

Instruction	Assembler Mnemonic	Time (microseconds)
Halt	HALT	1.01 + time to restart
Reset Programmable Status Indicators	RSTS	1.59
Set Programmable Status Indicators	SETS	1.72
Enable Interrupts	ENBL	1.85
Enable and Wait	ENBW	2.80 + time to interrupt
Disable Interrupts	DSBL	1.98
Disable and Wait	DSBW	2.80 + time to interrupt
Wait	WAIT	2.80 + time to interrupt
Status to Memory	STSM	2.14 Absolute, 2.46 Relative
Registers to Memory	REGM	7.24 Absolute, 7.56 Relative
Return from Interrupt	RETN	4.26 Absolute, 4.58 Relative
Memory to Status	MSTS	2.47 Absolute, 2.79 Relative
Memory to Registers	MREG	7.93 Absolute, 8.25 Relative

Instruction Set

UNIMPLEMENTED INSTRUCTIONS

Instruction class codes B_{16} to F_{16} (11-15) have not been implemented for SUE 1110 Processors. They are reserved for instruction set expansion, and some have been expanded to accommodate Processors SUE 1111A and B, and 1112A and B, as described in the last four sections of this bulletin. Use of these class codes for SUE 1110 causes the instruction to be trapped for software interpretation of the instruction. Trapping an instruction refers to the action taken on unimplemented instructions. When an unimplemented bit combination is detected, a transfer is made to an interpretive subroutine that can either simulate the instruction execution or perform some specialized system functions.

INPUT/OUTPUT INSTRUCTIONS

There are no dedicated input-output instructions. The upper 4K addresses (out of a total 64K) are reserved for device addresses, control words, status words, etc. Input/output functions may be accomplished by ordinary general instructions, and status checking by test instructions.

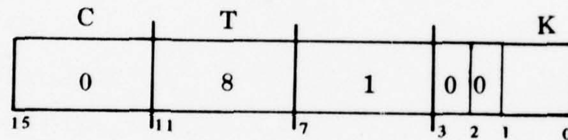
SUE 1110A INSTRUCTION SET

INTRODUCTION

Processor SUE 1110A, an improved design of SUE 1110, provides the capability to set the key bits of the address bus. The SUE 1110A instruction set includes all of the instructions for SUE 1110 (basic) plus the Store Key, SKEY, instruction.

STORE KEY INSTRUCTION

A subclass of class code 0, the Store Key instruction has the following format:



K - A two bit value to be stored into the key bits.

Refer to Appendix A for instruction times.

SUE 1110B INSTRUCTION SET

INTRODUCTION

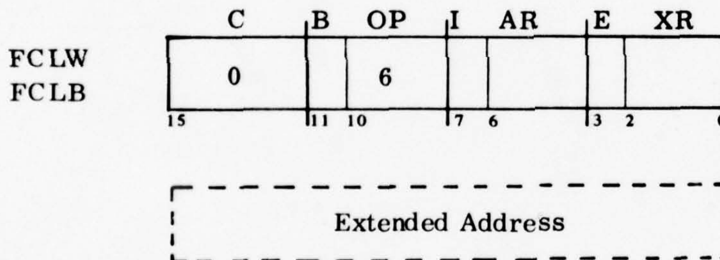
SUE 1110B Instruction Set includes all of the instructions performed by Processors SUE 1110 (basic), and 1110A, and the two Fetch and Clear (word or byte) instructions described below. Refer to Appendix A for instruction times.

Instruction Set

FETCH AND CLEAR INSTRUCTIONS

A subclass of class code 0, Fetch and Clear allows implementation of multi-processor systems with shared resources.

FETCH AND CLEAR INSTRUCTION WORD FORMAT



B - Word when 0 (FCLW), byte when 1 (FCLB)

I - Indirect when 1

AR - Accumulator register designator (0-7)

E - Extended or two-word instruction when 1

XR - Index register designator (0-7), no indexing when 0

FETCH AND CLEAR OPERATION

This instruction reads and clears the designated memory word or byte and places the previous contents into the designated register. In particular, it allows a processor to read a memory operand without allowing another processor to read the same memory operand before it has been cleared by the first processor.

NOTE

Both the memory cell and the designated register are cleared by these instructions when performed by SUE 1110A, 1111A and 1112A processors. SUE 1110 (basic) processor traps on this instruction.

Instruction Set

SUE 1111A INSTRUCTION SET

INTRODUCTION

The SUE 1111A Instruction Set includes all instructions described for processors SUE 1110 (basic), SUE 1110A, and instructions described in this section under SUE 1111A. Processor SUE 1111A can perform the following decimal arithmetic instructions:

Instruction	Mnemonic	Operation Code	Description
Zero and Add	ZADD	2	Move decimal field
Add Decimal	ADDD	3	Add decimal fields
Subtract Decimal	SUBD	4	Subtract decimal fields
Compare Decimal	CMPD	5	Compare decimal fields
Shift Right	SFTR	8	Shift decimal field right
Move Right	MOVR	9	Move field right to left
Shift Left	SFTL	A	Shift decimal field left
Move Left	MOVL	B	Move field left to right
Compare Field	COMP	C	Compare fields

TEMPORARY STORAGE

The decimal instructions implemented in SUE 1111 use the storage locations associated with the unimplemented instruction trap as temporary storage. For example, locations 20, 22, and 24 for the CPU have the contents of registers 5, 6, and 7 during and after completion of a decimal instruction. If an unimplemented instruction routine is to use a decimal instruction, it must save and restore these locations.

INSTRUCTION FORMAT

All decimal instructions except Shift Right (SFTR) and Shift Left (SFTL) are accommodated by one standard format as follows:

	H1				H2				H3				H4			
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Word 0	C				OP				0	R1			0	R2		
Word 1	L1				L2				0	X1			0	X2		

A modification of this format accommodates SFTR and SFTL instructions:

	H1				H2				H3				H4			
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Word 0	C				OP				0	R1			0			
Word 1	L1				S				0	X1			0	X2		

- C — Class Code - All instructions use class code C_{16}
- OP — Operation Codes 2_{16} through 5_{16} and 8_{16} through C_{16} (see instruction description)
- R1 — Register designator (1-7) for address of the most significant byte of the source field
- R2 — Register designator (1-7) for the address of the most significant byte of the destination field
- L1 — Source field length, minus 1 (bytes)
- L2 — Destination field length, minus 1 (bytes)
- S — Number of decimal digit positions to be shifted.
- X1 — Index register, content of which is appended to the L1 field, (only if X1 is non-zero). Length is then L1 + (X1) + 1.
- X2 — Index register, content of which is appended to the L2 or S field, (only if X2 is non-zero). Length is then L2 + (X2) + 1. For Shift instructions the content of X2 is appended to the S field. The total digit positions shifted is then S + (X2).

NOTES

All seven general registers may be used as the source for parameters. If zero is specified as the register for X1 or X2, then only the L or S field is used. R1 and R2 cannot be zero.

The target field must be large enough to hold each operation result, else the result is truncated without overflow status being set.

Instruction Set

DECIMAL DATA FORMAT

Decimal data is formatted with two digits (4 bit fields or nibbles) per byte with the right-most nibble taken as a sign. The valid decimal digits are 0 to 9_{16} with A_{16} to F_{16} giving invalid results. Valid signs are C for plus and D for minus.

Two examples of decimal data format required for operation codes 2, 3, 4, 5, 8, and A follow:

Example 1

0 1	3 8	5 C
Byte 1	Byte 2	Byte 3

Example 1 represents the number +1385.

Example 2

3 5	5 7	3 D

Example 2 represents the number -35573.

CHARACTER DATA FORMAT

The character data format used with operation codes 9, B, and C follow:

A	B	C
Byte 1	Byte 2	Byte 3

SYMBOLIC CODING FOR OPERANDS

Symbolic coding of the operand field for all instructions except Shift is: (R1), L1(X1), (R2), L2(X2). For Shift instructions, SFTR and SFTL, the coding is: (R1), L(X1), S(X2).

INSTRUCTIONS

ZERO AND ADD ZADD

OP = 2

Zero And Add moves the source field to the destination field. If the destination field is longer than the source field, the excess high-order bytes are filled with zeros. If the destination field is shorter than the source field, the excess data is truncated. The move takes place from the right end first so that truncated data is in the most significant portion of the field.

ADD ADDD

OP = 3

Add Decimal numerically adds the signed source field to the signed destination field with the sum placed into the destination field. If the sum is larger than the destination field, the excess high-order bytes are truncated. If the sum is smaller than the destination field, the excess high-order destination field bytes are filled with zeros.

Addition occurs from right to left one byte at a time. The sum contains the correct sign following the normal rules of algebra. If the result of the addition is zero, the result has the same sign as the original destination field.

SUBTRACT SUBD

OP = 4

Subtract Decimal numerically subtracts the signed source field from the destination field with the difference placed into the destination field. If the difference is larger than the destination field, the excess high-order bytes are truncated. If the difference is smaller than the destination field, the excess high-order destination field bytes are filled with zeros.

Subtraction occurs from right to left one byte at a time. The difference contains the correct sign following the normal rules of algebra. If zero, the difference has the same sign as the original destination field.

COMPARE DECIMAL CMPD

OP = 5

Compare Decimal numerically compares the signed source field with the signed destination field. Status bits EQ (bit 0) and GT (bit 1) indicate the results of the compare. The status bits are affected as follows:

	GT	EQ
Source = Destination	0	1
Source > Destination	1	0
Source < Destination	0	0

Commensurate with the rules of order, positive and negative zeros are equal by the Decimal Compare instruction, and large negative numbers are smaller than small negative numbers.

The source and destination fields need not be the same length. The numeric values of the two fields are compared and leading zeros are ignored. Neither the source nor the destination field is altered by this instruction. To provide a faster decision algorithm, comparison is made first on the sign and units digits, then on digits in order of most significance.

SHIFT RIGHT SFTR

OP = 8

Shift Right performs a decimal digit shift to the right. Digit positions on the left end of the field are filled with zeros as shifting proceeds. Digits on the right end are shifted out around the sign and lost, leaving the sign unchanged. Note that the number of shifts refers to the digit positions shifted and not the number of bytes. Also note that the true number of shifts is given and not the number minus 1. Therefore, zero shifts can be specified causing no operation to be performed. Shift Right provides a fast way to divide by a power of ten, and can be used for decimal point alignment, etc.

Instruction Set

MOVE RIGHT MOVR

OP = 9

Move Right transfers the source field to the destination field. The right-most byte (units/sign byte in decimal field) is moved first. Then, the move proceeds to the left one byte at a time.

If the source field is smaller than the destination field, the remaining left end of the destination field is unchanged. If the destination field is smaller than the source field, the move proceeds to the left until the destination field is full; then the move aborts so that all right-end bytes are transferred correctly. The source field is left unchanged unless it overlaps the destination field.

SHIFT LEFT SFTL

OP = A

Shift Left performs a decimal digit shift to the left. Digit positions on the right end of the field, except the sign position, are filled with zeros as shifting proceeds. The sign is not shifted and left unaltered. Digits on the left end are shifted out and lost.

Note that the number of shifts refers to the digit positions shifted and not the number of bytes. Also note that the true number of shifts is given and not the number minus 1. Therefore, zero shifts can be specified causing no operation to be performed. Shift Left provides a fast way to multiply by a power of ten, and can be used for decimal point alignment, etc.

MOVE LEFT MOVL

OP = B

Move Left transfers the source field to the destination field. The left-most byte is moved first, then the move proceeds to the right one byte at a time.

If the source field is smaller than the destination field, the remaining right end of the destination field is left unchanged. If the destination field is smaller than the source field, the move proceeds to the right until the destination field is

full; then the move aborts so that all left-end bytes are transferred correctly. The source field is left unchanged unless it overlaps the destination field.

COMPARE FIELD COMP

OP = C

Compare Field compares the source field with the destination field. The compare operates from left to right. If either field is shorter than the other, the shorter field is considered to be extended to the right with ASCII blanks (A0) during the compare operation. Status bits EQ (bit 0) and GT (bit 1) indicate the compare results and are affected as follows:

	GT	EQ
Source = Destination	0	1
Source > Destination	1	0
Source < Destination	0	0

The compare assumes the collating sequence of ASCII (i.e. binary values of 8-bit characters as stored internally in core).

Instruction Set

SUE 1111B INSTRUCTION SET

INTRODUCTION

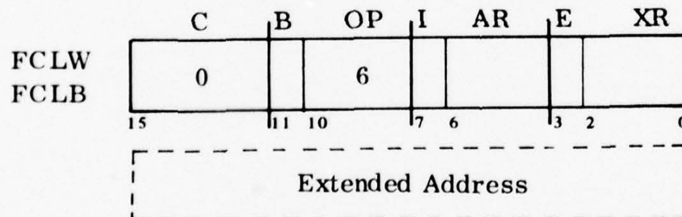
The SUE 1111B Instruction Set includes all instructions performed by processors SUE 1110 (basic), 1110A, and 1111A, and the two Fetch and Clear instructions described below. (Fetch and Clear, described for SUE 1110B and 1112B processors, is repeated here for programmer convenience.)

Refer to Appendix A for instruction times.

FETCH AND CLEAR INSTRUCTIONS

A subclass of class code 0, Fetch and Clear allows implementation of multiprocessor systems with shared resources.

FETCH AND CLEAR INSTRUCTION WORD FORMAT



B - Word when 0 (FCLW), byte when 1 (FCLB)

I - Indirect when 1

AR - Accumulator register designator (0-7)

E - Extended or two-word instruction when 1

XR - Index register designator (0-7), no indexing when 0

FETCH AND CLEAR OPERATIONS

This instruction reads and clears the designated memory word or byte and places the previous contents into the designated register. In particular it

allows a processor to read a memory operand without allowing another processor to read the same memory operand before it has been cleared by the first processor.

NOTE

Both the memory cell and the designated register are cleared by this instruction when performed with SUE 1110A, 1111A, and 1112A processors. SUE 1110 (basic) processor traps on this instruction.

Instruction Set

SUE 1112A INSTRUCTION SET

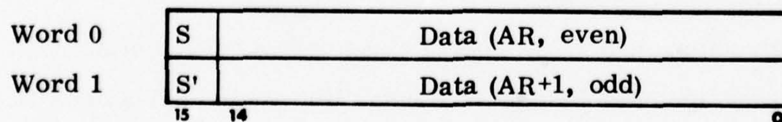
INTRODUCTION

The SUE 1112A Double Precision Processor performs all instructions described in this bulletin for SUE 1110 (basic), 1110A, and the extended instructions described in this section under SUE 1112A. Following is a list of the 1112A extended instructions and their class codes:

Instruction	Class Codes (Hexadecimal)
Bit Manipulation	4
Move	4
Normalize and Count	A
Double-Length Shift	A
Single Precision Fixed Point	B
Double Precision Fixed Point	B
Control	0

DOUBLE PRECISION DATA FORMAT

Double-precision data operations are accommodated by the following format:



Word 0 - Most significant fifteen bits of the fixed-point number

Word 1 - Least significant fifteen bits of the fixed-point number

Range - $-(2^{30}) \leq \text{Number} < +(2^{30})$

S - Sign bit of the 32-bit twos complement fixed-point number

S' - Sign bit extension in the least significant word

Upon termination of all

double-length arithmetic normalize,

double-length arithmetic shift, and

double-precision add, subtract, and multiply

instructions, sign S' is adjusted to reflect sign S unless bits 14 through 0 of word 1 are all zeros. In that event, sign S' also is zero. Also, on these double-length instructions, and double-length Load And Store: the zero indicator (ZE) reflects the condition of both words, the sign indicator (NG) reflects the sign of the most significant word, and the odd indicator (OD) reflects the value of the least significant bit of the least significant word.

NOTE

To take the two's complement of a double precision number, use the following procedure: If the least significant word is not zero, take the two's complement of the least significant word and the one's complement of the most significant word. If the least significant word is zero, take the two's complement of the most significant word.

INSTRUCTION TIMES

Refer to Appendix A for timing of SUE 1112A arithmetic instructions.

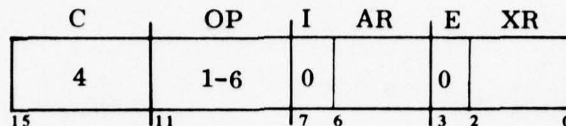
BIT MANIPULATION INSTRUCTIONS

Bit Manipulation Instructions use a subclass of class code 4. Each of six operations in the bit manipulation instructions may alter status indicators NG, ZE, OD, and CY, for subsequent testing by Branch Conditional instructions. OV is unaffected. CY = 0 indicates the designated bit or bits are all ZEROs.

BIT MANIPULATION INSTRUCTION FORMATS

Three instruction word formats are used by bit manipulation instructions:

Single Bit Addressed by (XR)

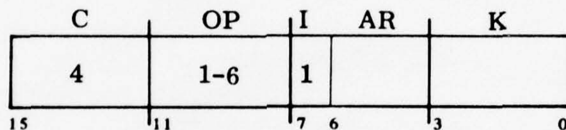


OP - Operation code 1-6

AR - Accumulator register (0-7) that contains the operation result

XR - Index register of which the least significant four bits contain the bit number to be tested and modified.

Single Bit Explicitly Designated by K

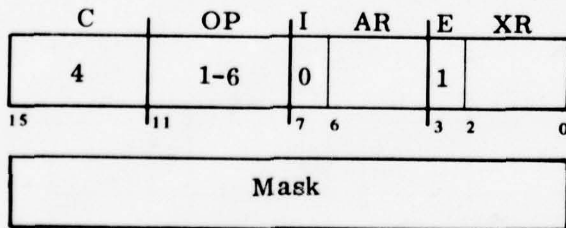


OP - Operation code 1-6

AR - Designator (0-7) for the accumulator register that contains the operation result

K - A four-bit value that specifies the bit number to be tested.

Multiple Bits Selected by the Mask and (XR)



OP - Operation code 1-6

AR - Designator (0-7) for the accumulator register that contains the operation result

Mask - Second word of extended instruction to select the bits for testing and modification.

XR - If 0, only the mask field is used to determine the bits to be tested; if 1-7, the mask is ANDed with the content of XR to select the bits to be tested and modified.

BIT MANIPULATION OPERATIONS

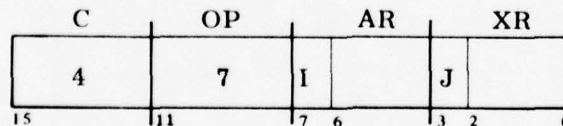
Following are bit manipulation operations. Each of the operations (OP codes 1-6) tests the designated bit or bits and sets CY if any are not zero; resets CY if all are zero.

Operation Code	Mnemonic	Description
1	RBIT	Make the designated bit or bits a 0, all others unchanged
2	SBIT	Make the designated bit or bits a 1, all others unchanged
3	CBIT	Change the designated bit or bits, all others unchanged
4	IBIT	Isolate the designated bit or bits, all others reset to 0
5	TSBT	Test the designated bit or bits and shift (AR) left one. The bit shifted out of $(AR)_{15}$ is lost and a zero is shifted into $(AR)_0$
6	TBIT	Only test the designated bit or bits

MOVE INSTRUCTIONS

Another subclass of class code 4 is used by the Move instructions. There are four operations, each of which may alter status indicators NG, ZE, and OD for subsequent testing by Branch Conditional instructions. CY and OV are unaffected.

MOVE INSTRUCTION FORMAT



AR - Destination register designator

XR - Source register designator

MOVE OPERATIONS

Following are Move instruction operations:

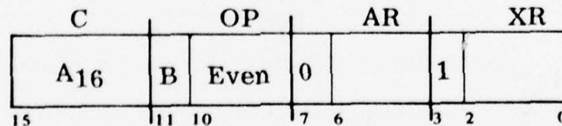
I-J Codes	Mnemonic	Description
I=0, J=0	NEGT	Move the twos complement value of register XR to register AR. The content of XR is unchanged unless XR=AR.
I=0, J=1	CPLM	Move the ones complement value of register XR to register AR. The content of XR is unchanged.
I=1, J=0	MOVP	Move the positive magnitude of register XR to register AR. The content of XR is unchanged.
I=1, J=1	MOVM	Move the negative magnitude of register XR to register AR. The content of XR is unchanged.

Instruction Set

NORMALIZE AND COUNT INSTRUCTIONS

A subclass of class A (Shift) instructions is used by the Normalize And Count instructions. There are 8 instructions, 4 single-, and 4 double-length.

NORMALIZE AND COUNT INSTRUCTION FORMAT



B - 0 = single length shift, 1 = double length shift

AR - Designator of register to be shifted

XR - Designator of register to be incremented by shift count

NORMALIZE AND COUNT OPERATIONS

Following are Normalize And Count operations:

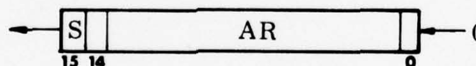
Operation Code	Mnemonic	Description
Single Length		
0	SLAN	Single left arithmetic normalize
2	SLLN	Single left logical normalize
4	SRAN	Single right arithmetic normalize
6	SRLN	Single right logical normalize

Operation Code	Mnemonic	Description
Double Length		
0	DLAN	Double left arithmetic normalize
2	DLLN	Double left logical normalize
4	DRAN	Double right arithmetic normalize
6	DRLN	Double right logical normalize

The register (AR), or pair of registers, (AR and AR+1, where AR is even) is shifted in the direction indicated until the requested condition is met. The count of positions shifted is added to (XR). A maximum shift of 15 (31 for double) may occur. If the register or registers indicated for normalize contain zero, the instruction sets only the status bits; AR and XR are not altered. Also, on double arithmetic normalize, the sign position of the odd register is ignored by the zero test; but the position is set to the sign of the even register, or it is cleared if the odd register bits 14 to 0 are all ZERO. Double-length arithmetic format is described under INTRODUCTION at the beginning of this section.

SINGLE LEFT ARITHMETIC NORMALIZE SLAN

OP = 0

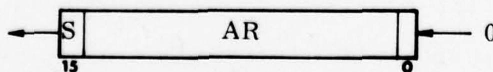


[Shift stops when bits $(AR)_{15}$ and $(AR)_{14}$ are different]

If any $(AR)_{14}$ is the same as $(AR)_{15}$, shift until these bits differ. Then add the shift count to (XR). ZEROs are shifted into $(AR)_0$ and bits shifted out of $(AR)_{15}$ are lost. Status bits NG, ZE, OD are affected accordingly. CY and OV are unaffected.

SINGLE LEFT LOGICAL NORMALIZE SLLN

OP = 2

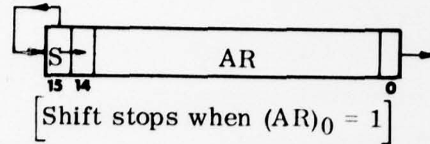


[Shift stops when $(AR)_{15} = 1$]

$(AR)_{15}$ is tested for a set condition; if not set, (AR) is shifted left. Zero is shifted into $(AR)_0$. When $(AR)_{15}$ is set, the operation terminates and the shift count is added to (XR). Status bits NG, ZE, and OD reflect the shift result. CY and OV are unaffected.

SINGLE RIGHT ARITHMETIC NORMALIZE SRAN

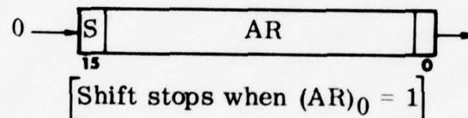
OP = 4



$(AR)_0$ is tested for a set condition. If not set, (AR) is shifted right. $(AR)_{15}$, the sign bit, remains the same and is shifted into $(AR)_{14}$. When $(AR)_0$ is set, the operation terminates and the shift count is added to (XR). Status bits NG, ZE, OD reflect the shift result. CY and OV are unaffected.

SINGLE RIGHT LOGICAL NORMALIZE SRLN

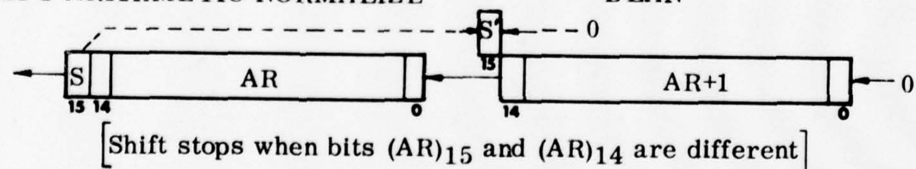
OP = 6



$(AR)_0$ is tested for a set condition. If not set, (AR) is shifted right. Zero bits are shifted into $(AR)_{15}$. When $(AR)_0$ is set, the operation terminates and the shift count is added to (XR). Status bits NG, ZE, and OD reflect the shift result. CY and OV are unaffected.

DOUBLE LEFT ARITHMETIC NORMALIZE DLAN

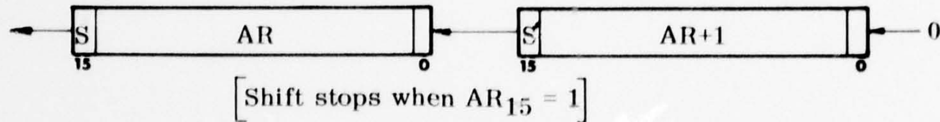
OP = 2



If any $(AR)_{14}$ is the same as $(AR)_{15}$, shift until these bits differ. Then add the shift count to (XR). ZEROs are shifted into $(AR+1)_0$ and $(AR+1)_{14}$ is shifted into $(AR)_0$. When shift stops: $(AR+1)_{15} = 0$ if $(AR+1)_{14-0} = 0$; if not, $(AR+1)_{15} = (AR)_{15}$. Status bits NG, ZE, and OD are affected accordingly. CY and OV are unaffected.

DOUBLE LEFT LOGICAL NORMALIZE DLLN

OP = 2

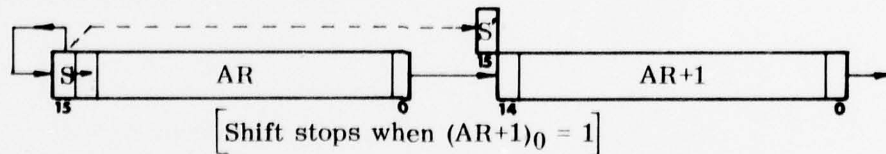


[Shift stops when $AR_{15} = 1$]

$(AR)_{15}$ is tested for a set condition; if not set, (AR) is shifted left. ZEROs are shifted into $(AR+1)_0$. $(AR+1)_{15}$ is shifted into $(AR)_0$. When $(AR)_{15}$ is set, the operation terminates and the shift count is added to (XR). Status bits NG, ZE, and OD reflect the shift result. CY and OV are unaffected.

DOUBLE RIGHT ARITHMETIC NORMALIZE DRAN

OP = 4

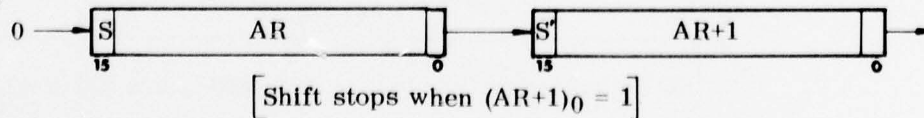


[Shift stops when $(AR+1)_0 = 1$]

$(AR+1)_0$ is tested for a set condition. If not set, (AR) and (AR+1) are shifted right. $(AR)_{15}$, the sign bit, remains the same and is shifted into $(AR)_{14}$. $(AR)_0$ is shifted into $(AR+1)_{14}$. When $(AR+1)_0$ is set, the operation terminates. Upon termination, $(AR+1)_{15}$ is adjusted to reflect the sign of the even register.

DOUBLE RIGHT LOGICAL NORMALIZE DRLN

OP = 6



[Shift stops when $(AR+1)_0 = 1$]

Zero is shifted into $(AR)_{15}$. $(AR+1)_0$ is tested for a set condition. If not set, (AR) and (AR+1) are shifted right and $(AR)_0$ is shifted into $(AR+1)_{15}$. When $(AR+1)_0$ is set, the operation terminates and the shift count is added to XR. Status bits NG, ZE, and OD reflect the shift result. CY and OV are unaffected.

DOUBLE LENGTH SHIFT INSTRUCTIONS

A subclass of class A (Shift) instructions is used by the Double-Length Shift Instructions. There are 8 instructions, corresponding to the 8 single-length shifts.

DOUBLE-LENGTH SHIFT INSTRUCTION FORMAT

Two formats are shown below. The shift count is contained either in the register designated by XR (bit 7 = 0), or in instruction bits 4 through 0, designated by K (bit 7 = 1).

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Shift Indexed	A_{16}				1	OP			0	AR		▨	0	XR		
Shift Immediate	A_{16}				1	OP			1	AR		K				

AR - Designates the register pair (2, 3; 4, 5; or 6, 7) to be shifted

XR - Designates register containing the shift count

K - Shift count for immediate. Note that the field contains 5 bits, extending into bit position 4 (which is not used by AR).

OP - Shift operation code

DOUBLE-LENGTH SHIFT OPERATIONS

The double-length shift operations are similar to the single-length shift subclass, and are defined as follows:

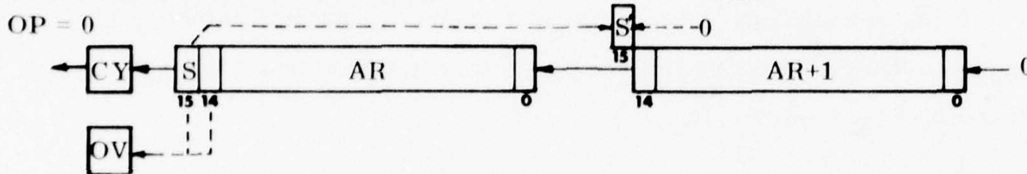
Operation Code	Mnemonic	Operation
0	DLAO	Double Left Arithmetic Open
1	DLLL	Double Left Logical Linked
2	DLLO	Double Left Logical Open
3	DLLC	Double Left Logical Closed
4	DRAO	Double Right Arithmetic Open
5	DRLL	Double Right Logical Linked
6	DRLO	Double Right Logical Open
7	DRLC	Double Right Logical Closed

On arithmetic shifts, the inter-register shift coupling is between bit 0 of the even register and bit 14 of the odd register.

Instruction Set

DOUBLE LEFT ARITHMETIC OPEN DLAO

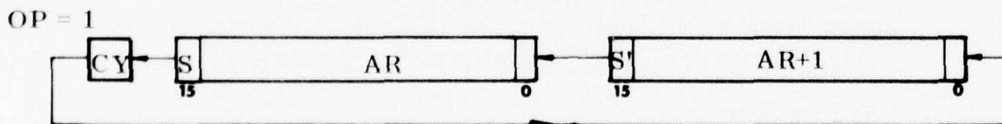
OP = 0



(AR) and (AR+1) are shifted left. $(AR)_{15}$ is shifted to carry (CY), and zeros are shifted into $(AR+1)_0$. If any $(AR)_{14}$ bit shifted is different than $(AR)_{15}$, overflow indicator, OV, is set. When shift stops: $(AR+1)_{15} = 0$ if $(AR+1)_{14-0} = 0$; if not, $(AR+1)_{15} = (AR)_{15}$. Operation affects status indicators CY, OV, NG, ZE, OD.

DOUBLE LEFT LOGICAL LINKED DLLL

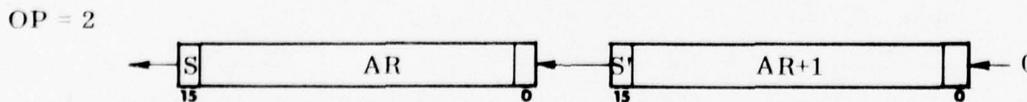
OP = 1



Carry (CY) is shifted into $(AR+1)_0$. $(AR)_{15}$ is shifted into CY. Operation affects status indicators: CY, NG, ZE, OD. (OV cannot set as in SLLL.)

DOUBLE LEFT LOGICAL OPEN DLLO

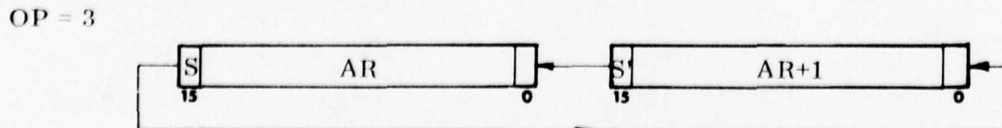
OP = 2



(AR) and (AR + 1) are shifted left. For each bit shifted, $(AR)_{15}$ is lost and $(AR + 1)_0$ equals 0. Operation affects status indicators: NG, ZE, OD.

DOUBLE LEFT LOGICAL CLOSED DLLC

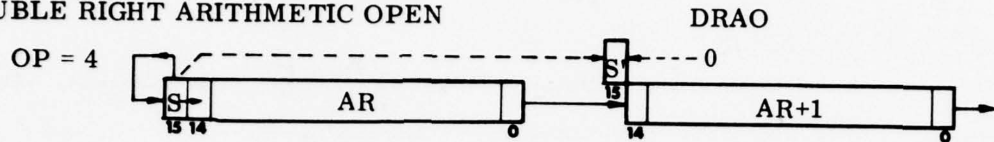
OP = 3



(AR) and (AR + 1) are shifted left. $(AR)_{15}$ is shifted into $(AR + 1)_0$. Operation affects status indicators: NG, ZE, OD.

DOUBLE RIGHT ARITHMETIC OPEN DRAO

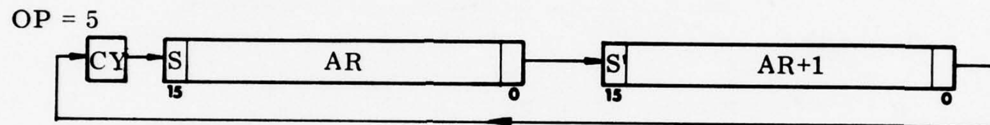
OP = 4



(AR + 1) and (AR) are shifted right. Sign bit $(AR)_{15}$ remains the same and is shifted into $(AR)_{14}$. $(AR+1)_0$ bits shifted out are lost. When shift stops: $(AR+1)_{15} = 0$ if $(AR+1)_{14-0} = 0$; if not, $(AR+1)_{15} = (AR)_{15}$. Operation affects status indicators: NG, ZE, OD.

DOUBLE RIGHT LOGICAL LINKED DRLL

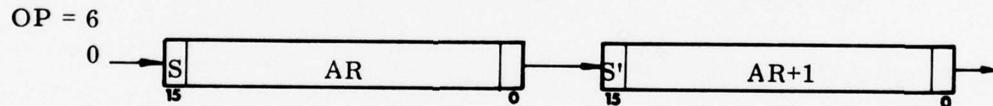
OP = 5



Carry (CY) is shifted into $(AR)_{15}$. $(AR+1)_0$ is shifted into CY. Operation affects status indicators CY, NG, ZE, OD.

DOUBLE RIGHT LOGICAL OPEN DRLO

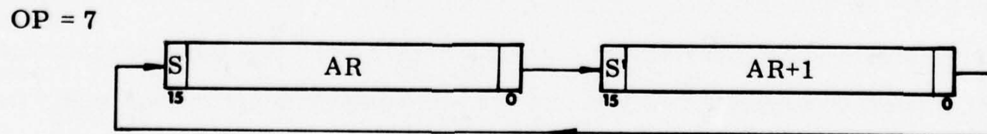
OP = 6



(AR) and (AR + 1) are shifted right. For each bit shifted, $(AR+1)_0$ is lost, and $(AR)_{15}$ equals 0. Operation affects status indicators: NG, ZE, OD.

DOUBLE RIGHT LOGICAL CLOSED DRLC

OP = 7

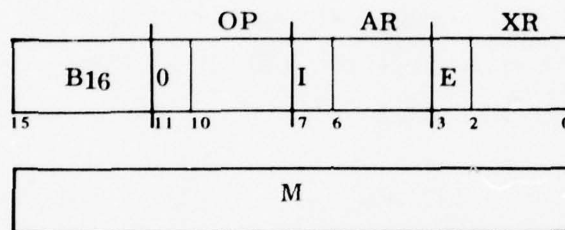


(AR) and (AR + 1) are shifted right. $(AR+1)_0$ is shifted into $(AR)_{15}$. Operation affects status indicators: NG, ZE, OD.

CLASS B INSTRUCTION SET

The Class B instruction set contains two sets of arithmetic instructions: single-precision fixed point and double-precision fixed point.

CLASS B INSTRUCTION FORMAT



OP - Operation code

I - Indirect address indicator

AR - Accumulator register pair for fixed point instructions (See definition below)

E - Extended address indicator

XR - Index register designator

M - Extended address (if required)

ACCUMULATOR REGISTERS

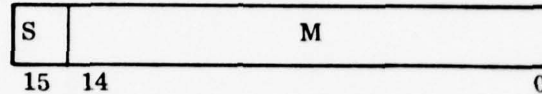
A register pair beginning with an even-numbered register, such as (R2, R3), (R4, R5) or (R6, R7) are defined as one accumulator for some single-precision and all double-precision fixed point instructions. An attempt to use R0 or R1 as an accumulator causes a level-5 interrupt.

SINGLE-PRECISION FIXED POINT INSTRUCTIONS

Three single-precision fixed point instructions supplement the basic instruction set. These extended instructions have a one-word memory operand and a two-register accumulator operand.

ADDRESSING MODES. - Standard memory-to-register addressing is permitted within this subclass. The accumulator registers are designated as a pair of registers addressed by the even-numbered register of the pair. The even-numbered register contains the most significant data.

ONE-WORD OPERAND FORMAT. - Following is the one-word operand format:



S - Sign bit of the 16-bit twos complement fixed point number

M - Remaining 15 bits of the fixed point number

$-(2^{15}) \leq \text{NUMBER} < +(2^{15})$

Format for double-length word is described under INTRODUCTION to this section.

Instruction Set

SINGLE-PRECISION FIXED-POINT OPERATIONS

Three single-precision operations are described as follows:

Operation Code	AR	Mnemonic and Operation	Description	Status Indicators Affected
3	2, 4, 6	MLTA (Multiply, Add)	Multiply the data in the odd-numbered register by the effective operand, and add the contents of the even-numbered register. A two-word product is formed in the combined registers.	NG, ZE, OD, CY
3	3, 5, 7	MULT (Multiply)	Multiply the data in the odd-numbered register by the effective operand to form a two-word product in the even-odd register pair.	NG, ZE, OD
4	2-3, 4-5, 6-7	DIVD (Divide)	Divide the data in the two-register accumulator by the effective operand. A properly-signed quotient results in the odd-numbered register, with the remainder (in the even-numbered register) having the same sign as the original dividend.	OV, NG, ZE, OD

NOTES

a. If register 0 or 1 is specified as AR, the instruction traps as an unimplemented instruction.

b. A multiplier or divisor of 8000 (i.e., - 65,536, the most negative number) has the same effect as if zero; results in setting OV on divide.

c. A multiplicand or addend of 8000 is treated as -65,536.

d. A dividend of 8000 0000 (i.e., most negative double-precision number) causes a divide check result.

DOUBLE PRECISION FIXED-POINT INSTRUCTIONS

Four double-precision, fixed-point instructions are provided in the extended instructions. Each has a two-word memory operand and a two-register accumulator operand. Format for double precision fixed point words is described under INTRODUCTION to this instruction set.

ADDRESSING MODES. - Standard memory-to-register addressing is permitted within the extended class. No other addressing modes are permitted. The effective memory address is the address of two consecutive memory words, the first containing the most significant data. The accumulator registers are designated as a pair of registers, addressed by the even-numbered register of the pair (e.g. R2 of the R2, R3 pair). The even-numbered register contains the most significant data.

DOUBLE PRECISION FIXED POINT OPERATIONS

Four operations are described as follows:

Operation Code	AR	Mnemonic and Operation	Description	Status Indicators Affected
5	2, 4, 6	DLOD (Double Load)	Move the contents of the two consecutive words located at the effective address to the combined registers.	NG, ZE, OD
0	2, 4, 6	DSTA (Double Store)	Move the contents of the two registers to the two consecutive words located at the effective address.	NG, ZE, OD
2	2, 4, 6	DADD (Double Add)	Add the contents of the consecutive words located at the effective address to the two registers.	CY, OV, NG, ZE, OD
1	2, 4, 6	DSUB (Double Subtract)	Subtract the contents of the two consecutive words located at the effective address from the two registers.	CY, OV, NG, ZE, OD

NOTE

If register 0 is specified as AR, the instruction traps as an unimplemented instruction.

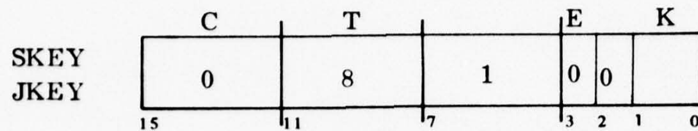
CONTROL INSTRUCTIONS

Four control instructions (mnemonics SKEY, JKEY, LCPU, LKEY) are included in the SUE 1112A Instruction Set. Instruction Store Key (SKEY), described in the SUE 1110A Instruction Set, is repeated here for programming convenience.

Instruction Set

CONTROL INSTRUCTION FORMATS

A subclass of class code 0, the control instructions use the following format:

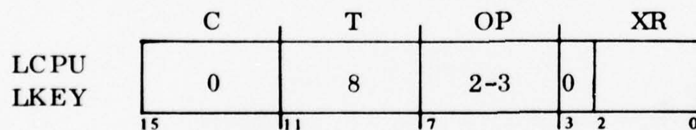


Jump Address

E - See description below

K - A two-bit value to be stored into the key bits of the address bus

Mnemonic	E	Description
SKEY	0	Store the value K into the key bits
JKEY	1	Store the value K into the key bits and the jump address into the program counter (i.e. jump)



OP - Operation code 2-3

XR - Index register designator

Operation Code	Mnemonic	Description
2	LCPU	Load the processor number into (XR) 5, 6; all other bits in (XR) are cleared.
3	LKEY	Load the Lkey value into (XR) right justified.

Instruction Set

SUE 1112B INSTRUCTION SET

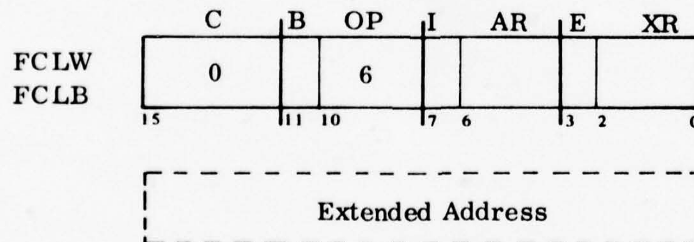
INTRODUCTION

SUE 1112B Instruction Set includes all of the instructions performed by processors SUE 1110 (basic), 1110A, 1112A and the two Fetch and Clear (word or byte) instructions described below. The Fetch and Clear Instructions, described also in the SUE 1110B and 1111B Instruction Sets, are repeated here for programmer convenience.

FETCH AND CLEAR INSTRUCTIONS

A subclass of class code 0, Fetch and Clear allows implementation of multiprocessor systems with shared resources.

FETCH AND CLEAR INSTRUCTION WORD FORMAT



B - Word when 0 (FCLW), byte when 1 (FCLB)

I - Indirect when 1

AR - Accumulator register designator (0-7)

E - Extended or two-word instruction when 1

XR - Index register designator (0-7), no indexing when 0

FETCH AND CLEAR OPERATION

This instruction reads and clears the designated memory word or byte and places the previous contents into the designated register. In particular it allows a processor to read a memory operand without allowing another processor to read the same memory operand before it has been cleared by the first processor.

NOTE

Both the memory cell and the designated register are cleared by this instruction when performed with SUE 1110A, 1111A and 1112A processors. SUE 1110 (basic) processor traps on this instruction.

APPENDIX A
INSTRUCTION TIMES

Table A-1. SUE 1110 (Basic) General Register Instruction Times

General Instruction Class	Class Code	Address Mode	Assembler Mnemonic	Operation Word or Byte, Time in Microseconds							
				MOV OP = 0	SUB OP = 1	ADD OP = 2	AND OP = 3	IOR OP = 4	EOR OP = 5	CMP OP = 6	TST OP = 7
ACCUMULATOR TO MEMORY, AUTO DECREMENT	1	Indexed	R, (-R)	4.81	4.93	4.90	4.81	4.81	4.81	4.57	4.22
		Extended, Indexed	R, A(-R)	4.94	5.06	5.03	4.94	4.94	4.94	4.70	4.35
		Indexed, Indirect	R, *(-R)	5.95	6.07	6.04	5.95	5.95	5.95	5.71	5.36
		Extended, Indexed, Indirect	R, *A(-R)	6.23	6.33	6.30	6.23	6.23	6.23	5.97	6.21
ACCUMULATOR TO MEMORY, AUTO INCREMENT	2	Indexed	R, (R+)	4.81	4.93	4.90	4.81	4.81	4.81	4.57	4.22
		Extended, Indexed	R, A(R+)	4.94	5.06	5.03	4.94	4.94	4.94	4.70	4.35
		Indexed, Indirect	R, *(R+)	5.95	6.07	6.04	5.95	5.95	5.95	5.71	5.36
		Extended, Indexed, Indirect	R, *A(R+)	6.23	6.33	6.30	6.23	6.23	6.23	5.97	6.21
ACCUMULATOR TO MEMORY	3	Indexed	R, (R)	3.94	4.06	4.03	3.94	3.94	3.94	3.70	3.35
		Extended	R, A	4.07	4.19	4.16	4.07	4.07	4.07	3.83	3.48
		Extended, Indexed	R, A(R)	4.07	4.19	4.16	4.07	4.07	4.07	3.83	3.48
		Indexed, Indirect	R, *(R)	5.08	5.20	5.17	5.08	5.08	5.08	4.84	4.49
		Extended, Indirect	R, *A	5.34	5.46	5.43	5.34	5.34	5.34	5.10	5.34
		Extended, Indexed, Indirect	R, *A(R)	5.34	5.46	5.43	5.34	5.34	5.34	5.10	5.34
DATA TO ACCUMULATOR	4	Register to Register	R, R	2.50	2.79	2.79	2.50	2.50	2.50	2.69	2.50
		Immediate to Register	=H)X, R	2.50	2.79	2.79	2.50	2.50	2.50	2.69	2.50
		Literal to Register	=H)XXXX, R	3.18	3.47	3.47	3.18	3.18	3.18	3.37	3.18
		Literal, Indexed to Register	=H)XXXX(R), R	3.34	3.63	3.63	3.34	3.34	3.34	3.53	3.34
MEMORY TO ACCUMULATOR, AUTO DECREMENT	5	Indexed	(-R), R	4.09	4.38	4.38	4.09	4.09	4.09	4.41	4.09
		Extended, Indexed	A(-R), R	4.22	4.51	4.51	4.22	4.22	4.22	4.54	4.22
		Indexed, Indirect	*(-R), R	5.23	5.52	5.52	5.23	5.23	5.23	5.55	5.23
		Extended, Indexed, Indirect	*A(-R), R	5.49	5.78	5.78	5.49	5.49	5.49	5.81	5.49
MEMORY TO ACCUMULATOR, AUTO INCREMENT	6	Indexed	(R+), R	4.09	4.38	4.38	4.09	4.09	4.09	4.41	4.09
		Extended, Indexed	A(R+), R	4.22	4.51	4.51	4.22	4.22	4.22	4.54	4.22
		Indexed, Indirect	*(R+), R	5.23	5.52	5.52	5.23	5.23	5.23	5.55	5.23
		Extended, Indexed, Indirect	*A(R+), R	5.49	5.78	5.78	5.49	5.49	5.49	5.81	5.49
MEMORY TO ACCUMULATOR	7	Indexed	(R), R	3.35	3.64	3.64	3.35	3.35	3.35	3.67	3.35
		Extended	A, R	3.48	3.77	3.77	3.48	3.48	3.48	3.80	3.48
		Extended, Indexed	A(R), R	3.48	3.77	3.77	3.48	3.48	3.48	3.80	3.48
		Indexed, Indirect	*(R), R	4.49	4.78	4.78	4.49	4.49	4.49	4.81	4.49
		Extended, Indirect	*A, R	4.75	5.04	5.04	4.75	4.75	4.75	5.07	4.75
		Extended, Indexed, Indirect	*A(R), R	4.75	5.04	5.04	4.75	4.75	4.75	5.07	4.75

Instruction Set

Table A-2. SUE 1110A/B, 1111A/B, and 1112A/B General Register Instruction Times

General Instruction Class	Class Code	Address Mode	Assembler Mnemonics OP Code	Operands	MOV (W/B) OP = 0	SUB (W/B) OP = 1	ADD (W/B) OP = 2	AND (W/B) OP = 3	IOR (W/B) OP = 4	EOR (W/B) OP = 5	CMP (W/B) OP = 6 Result ≤	CMP (W/B) OP = 6 Result >	TST (W/B) OP = 7
ACCUMULATOR TO MEMORY, AUTO DECREMENT	1	Register Address (Indexed)	----	R, (-R)	4.73	4.86	4.76	4.73	4.73	4.73	4.34	4.43	4.31
		Extended (with/without Indexing)	----	R, A(-R)	4.73	4.89	4.79	4.76	4.76	4.76	4.37	4.46	4.34
		Register Address, Indirect (Word)[1]	---W	R, *(-R)	6.71	6.87	6.77	6.74	6.74	6.74	6.35	6.44	6.32
		(Byte)	---B	R, *(-R)	5.96	6.12	6.02	5.99	5.99	5.99	5.60	5.69	5.57
		Extended, Indirect (Word)[1]	---W	R, *A(-R)	6.32	6.48	6.38	6.35	6.35	6.35	5.96	6.05	5.93
		(Byte)	---B	R, *A(-R)	5.70	5.86	5.76	5.73	5.73	5.73	5.34	5.43	5.31
ACCUMULATOR TO MEMORY, AUTO INCREMENT	2	Register Address (Indexed)	----	R, (R+)	4.73	4.86	5.76	4.73	4.73	4.73	4.34	4.43	4.31
		Extended (with/without Indexing)	----	R, A(R+)	4.80	4.96	4.86	4.86	4.86	4.86	4.44	4.53	4.41
		Register Address, Indirect (Word)[1]	---W	R, *(R+)	6.71	6.87	6.77	6.74	6.74	6.74	6.35	6.44	6.32
		(Byte)	---B	R, *(R+)	5.96	6.12	6.02	5.99	5.99	5.99	5.60	5.69	5.57
		Extended, Indirect (Word)[1]	---W	R, *A(R+)	6.39	6.55	6.45	6.42	6.42	6.42	6.03	6.12	6.00
		(Byte)	---B	R, *A(R+)	5.77	5.93	5.83	5.80	5.80	5.80	5.41	5.50	5.38
ACCUMULATOR TO MEMORY	3	Register Address (Indexed)	----	R, (R)	3.76	4.22	4.12	4.09	4.09	4.09	3.70	3.79	3.67
		Extended (with/without Indexing)	----	R, A(R) or R, A	3.83	4.16	4.06	4.03	4.03	4.03	3.64	3.73	3.61
		Register Address, Indirect (Word)[1]	---W	R, *(R)	5.74	6.07	5.97	5.94	5.94	5.94	5.55	5.64	5.52
		(Byte)	---B	R, *(R)	4.99	5.32	5.22	5.19	5.19	5.19	4.80	4.89	4.77
		Extended, Indirect, (Word)[1]	---W	R, *A(R) or R, *A	5.42	5.75	5.65	5.62	5.62	5.62	5.23	5.32	5.20
		(w/wo Indexing) (Byte)	---B	R, *A(R) or R, *A	4.80	5.13	5.03	5.00	5.00	5.00	4.61	4.70	4.58
ACCUMULATOR/EXPLICIT DATA TO ACCUMULATOR	4	Register to Register	----	R, R	2.24	2.66	2.53	2.24	2.24	2.24	2.40	2.49	2.24
		Indexed, Literal	----	=H)XXXX, R	3.12	3.54	3.41	3.12	3.12	3.12	3.28	3.37	3.12
		Literal (Full Word)	----	=H)XXXX(R), R	2.96	3.38	3.25	2.96	2.96	2.96	3.12	3.21	2.96
		Immediate (Four-bits)	----	=H)X, R	2.24	2.66	2.53	2.24	2.24	2.24	2.40	2.49	2.24
MEMORY TO ACCUMULATOR, AUTO DECREMENT	5	Register Address (Indexed)	----	(-R), R	4.05	4.47	4.34	4.05	4.05	4.05	4.21	4.30	4.05
		Extended (with/without Indexing)	----	A(-R), R	4.08	4.50	4.37	4.08	4.08	4.08	4.24	4.33	4.08
		Register Address, Indirect (Word)[1]	---W	*(-R), R	6.06	6.48	6.35	6.06	6.06	6.06	6.22	6.31	6.06
		(Byte)	---B	*(-R), R	5.31	5.73	5.60	5.31	5.31	5.31	5.47	5.56	5.31
		Extended, Indirect (Word)[1]	---W	*A(-R), R	5.67	6.09	5.96	5.67	5.67	5.67	5.83	5.92	5.67
		(Byte)	---B	*A(-R), R	5.05	5.47	5.34	5.05	5.05	5.05	5.21	5.30	5.05
MEMORY TO ACCUMULATOR, AUTO INCREMENT	6	Register Address (Indexed)	----	(R+), R	4.05	4.47	4.34	4.05	4.05	4.05	4.21	4.30	4.05
		Extended (with/without Indexing)	----	A(R+), R	4.15	4.57	4.44	4.15	4.15	4.15	4.31	4.40	4.15
		Register Address, Indirect (Word)[1]	---W	*(R+), R	6.06	5.48	6.35	6.06	6.06	6.06	6.22	6.31	6.06
		(Byte)	---B	*(R+), R	5.31	5.73	5.60	5.31	5.31	5.31	5.47	5.56	5.31
		Extended, Indirect (Word)[1]	---W	*A(R+), R	5.74	6.16	6.03	5.74	5.74	5.74	5.90	5.99	5.74
		(Byte)	---B	*A(R+), R	5.12	5.54	5.41	5.12	5.12	5.12	5.28	5.37	5.12
MEMORY TO ACCUMULATOR	7	Register Address (indexed)	----	(R), R	3.54	3.96	3.83	3.54	3.54	3.54	3.70	3.79	3.54
		Extended (with/without Indexing)	----	A(R), R or A, R	3.48	3.90	3.77	3.48	3.48	3.48	3.64	3.73	3.48
		Register Address, Indirect (Word)[1]	---W	*(R), R	5.39	5.81	5.68	5.39	5.39	5.39	5.55	5.64	5.39
		(Byte)	---B	*(R), R	4.64	5.06	4.93	4.64	4.64	4.64	4.80	4.89	4.64
		Extended, Indirect, (Word)[1]	---W	*A(R), R or *A, R	5.07	5.49	5.36	5.07	5.07	5.07	5.23	5.32	5.07
		(w/wo Indexing) (Byte)	---B	*A(R), R or *A, R	4.45	4.87	4.74	4.45	4.45	4.45	4.61	4.70	4.45
JUMP OR CALL SUBROUTINE	4	Register Address (Indexed)	JUMP JSBR	R R, R	3.01								
		Extended (with/without Indexing)	JUMP JSBR	A or A(R) A, R or A(R), R	3.04								
		Register Address, Indirect[1]	JUMP JSBR	*R *R, R	5.02								
		Extended, Indirect[1] (w/wo Indexing)	JUMP JSBR	*A or *A(R) *A, R or *A(R), R	4.63								
FETCH AND CLEAR	0	Register Address (Indexed)	----	(R), R	5.02								
		Extended (with/without Indexing)	----	A(R), R or A, R	4.96								
		Register Address, Indirect (Word)[1]	FCLW	*(R), R	6.87								
		(Byte)	FCLB	*(R), R	6.12								
		Extended, Indirect (Word)[1]	FCLW	*A(R), R or *A, R	6.55								
		(Byte)	FCLB	*A(R), R or *A, R	5.93								

NOTE

[1] Add 1.01 microseconds for each additional level of indirect.

Table A-3. SUE 1110 (Basic), 1110A/B, 1111A/B and 1112A/B
Control Instruction Times

Instruction	Assembler Mnemonic	Time (microseconds)
Halt	HALT	1.01 + time to restart
Reset Programmable Status Indicators	RSTS	1.59
Set Programmable Status Indicators	SETS	1.72
Enable Interrupts	ENBL	1.85
Enable and Wait	ENBW	2.80 + time to interrupt
Disable Interrupts	DSBL	1.98
Disable and Wait	DSBW	2.80 + time to interrupt
Wait	WAIT	2.80 + time to interrupt
Status to Memory	STSM	2.14 Absolute, 2.46 Relative
Registers to Memory	REGM	7.24 Absolute, 7.56 Relative
Return from Interrupt	RETN	4.26 Absolute, 4.58 Relative
Memory to Status	MSTS	2.47 Absolute, 2.79 Relative
Memory to Registers	MREG	7.93 Absolute, 8.25 Relative
Store Key	SKEY*	2.6

*Not available in SUE 1110 (Basic) Processor

SINGLE SHIFT INSTRUCTION TIMING FOR SUE 1110 (Basic), 1110A/B, 1111A/B, 1112A/B

Shift instruction execution times depend on the number of single bit shifts specified in either the K field (immediate) or the selected register, XR. The time is calculated by the formula:

$$T_S = 2.76 + (0.26)N$$

where N = 0, 1, ..., 15.

Instruction Set

Table A-4. SUE 1110 (Basic) Branch Instruction Times

Instruction	Assembler Mnemonic	Time (microseconds)	
		Next Word	Branch
No Operation	NOPR	1.78	-
Branch Unconditional	BRUN	-	2.72
Branch True	BxxT*	1.78	2.72
Branch False	BxxF*	1.78	2.72
Branch Less Than True	BLTT	1.75	3.08
Branch Less Than False	BLTF	1.88	3.08

*where xx = EQ, GT, OV, CY, F1, F2, F3, LP, OD, ZE, NG.

Table A-5. SUE 1110A/B, 1111A/B and 1112A/B Branch Instruction Times

Instruction	Assembler Mnemonic	Time (microseconds)	
		Next Word	Branch
No Operation	NOPR	1.75	-
Branch Unconditional	BRUN	-	2.82
Branch True	BxxT*	1.75	2.82
Branch False	BxxF*	1.75	2.82
Branch Less Than True	BLTT	1.75	3.21
Branch Less Than False	BLTF	1.88	3.21

*where xx = EQ, GT, OV, CY, F1, F2, F3, LP, OD, ZE, NG.

SUE 1111A/B CLASS C INSTRUCTION TIMES

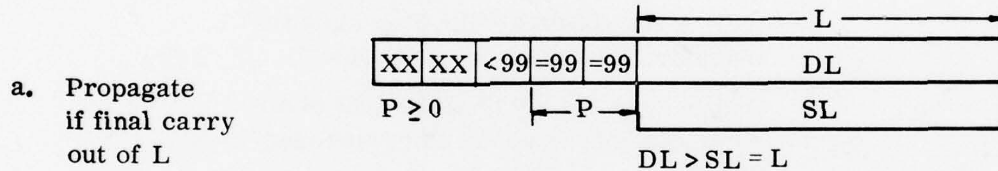
DECIMAL AND CHARACTER INSTRUCTIONS

DECIMAL ADD AND SUBTRACT TIMING

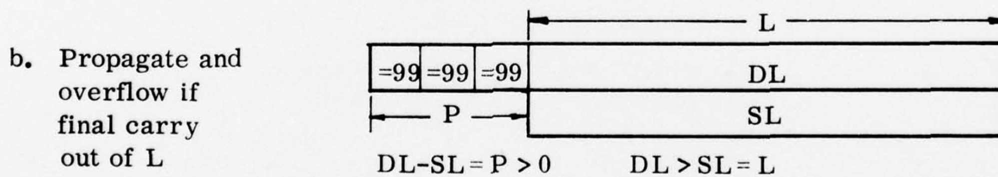
Both the decimal add and subtract operations are started with a sign analysis to determine whether true addition (ADDD with like signs or SUBD with unlike signs), or true subtraction (ADDD with unlike signs or SUBD with like signs) is to occur. Timing for both situations can be calculated using the following case formats and procedures:

TRUE ADDITION CASE FORMATS

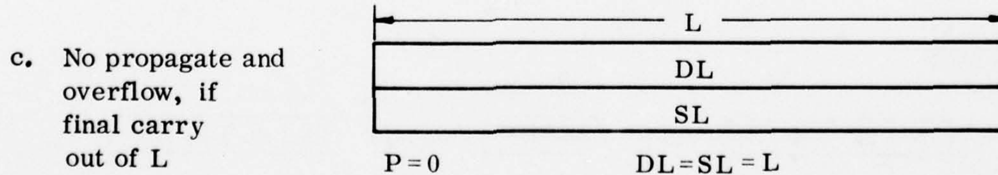
a. Propagate if final carry out of L



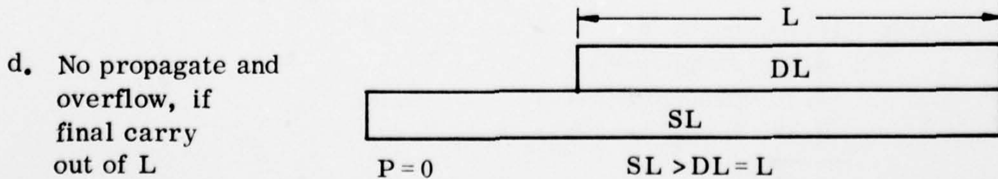
b. Propagate and overflow if final carry out of L



c. No propagate and overflow, if final carry out of L



d. No propagate and overflow, if final carry out of L



TRUE ADDITION TIMING PROCEDURES

Step	For Case Formats	Conditions (Times in Microseconds)	Times (Microseconds)
1	a, b, c, d	Start with the basic time of 10.70 and go to Step 2.	10.70
2	a, b, c, d	For each digit pair to be added (including units and signs) add in 3.21 and go to Step 3.	3.21L
3	a, b, c, d	STOP if no final carry from the summation; otherwise take Step 4.	+0
4	c, d	Add in 0.26 and STOP if SL ≥ DL; otherwise take Step 5.	+0.26
5	a, b	For each digit pair in the extension field (i.e. where DL > SL or P > 0) that equals 99, add in 2.7 until a non-99 pair is encountered (take Step 6), or the extension field runs out (take Step 8).	2.70P
6	a	Add in 2.30 and STOP if the LSD of an extension pair is not 9; otherwise take Step 7.	+2.30
7	a	Add in 2.33 and STOP when the MSD of an extension pair is not 9.	+2.33
8	b	Add in 0.26 and STOP when the extension field runs out (i.e. only 99 is encountered).	+0.26

Example

07	99	74	3+

minus

42	6-

equals

08	00	16	9+

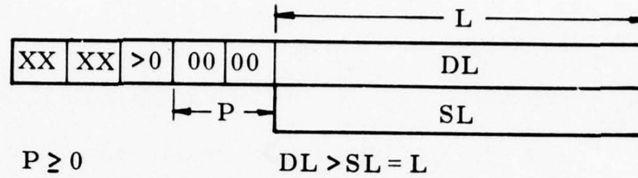
P=1, L=2, Case format: a

time = 10.70 basic
3 x 3.21 summation
2.70 propagate carry
\+ 2.30 terminate carry

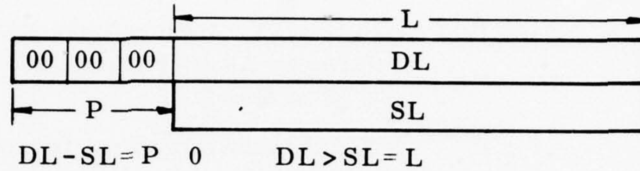
22.12 microseconds total

TRUE SUBTRACTION CASE FORMATS

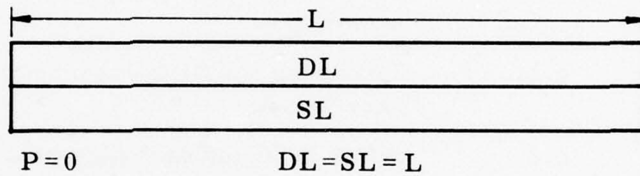
e. Propagate if final borrow out of L



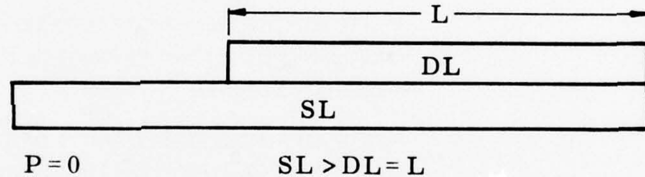
f. Propagate and recomplement if final borrow out of L



g. No propagate and recomplement if final borrow out of L

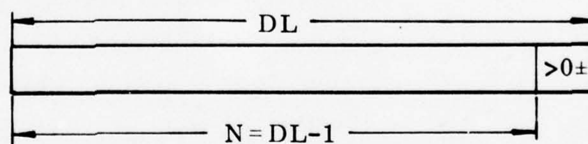


h. No propagate and recomplement if final borrow out of L

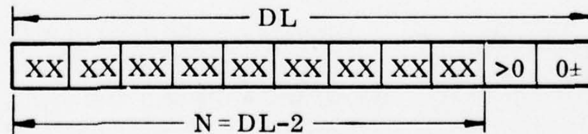


RECOMPLEMENT CASE FORMATS

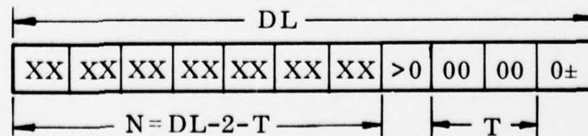
j. Units digit ≠ 0, start nine's complement



k. Short ten's complement



l. Extended ten's complement

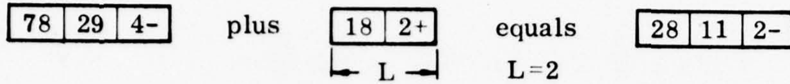


TRUE SUBTRACTION TIMING PROCEDURE

Step	For Case Formats	Conditions (Times in Microseconds)	Times (Microseconds)
1	e, f, g, h	Start with the basic time of 10.99 and go to Step 2.	10.99
2	e, f, g, h	Add in 3.21 for each digit pair to be subtracted (including units and signs) and go to Step 3.	3.21L
3	e, f, g, h	STOP if there is no final borrow from the subtraction process; otherwise take Step 4.	+0
4	g, h	Go to Step 7 if SL ≥ DL; otherwise take Step 5.	
5	e, f	Add in 2.01 for each digit pair in the extension field (i.e. where DL > SL or P > 0) until a non-zero is encountered (take Step 6), or the extension field runs out (take Step 7).	2.01P
6	e	Add 2.01 and STOP when a non-zero digit is encountered in the extension field.	+2.01
7	j, k, l	Add 2.43 (for first step of recomplement); go to Step 10 if the units digit is not zero; otherwise go to Step 8.	2.43
8	l	Add 1.33 for each digit pair in the destination field that is zero until a non-zero pair is encountered, then go to Step 9.	+1.33T
9	k, l	Add 2.14 if the LSD of a digit pair is not zero; add 2.40 if the LSD is zero and the MSD of a digit pair is not zero. Go to Step 10.	2.14 2.40
10	j, k, l	Add 1.75 for each remaining digit pair in the destination field and STOP.	+1.75N

Four Examples of Subtract Timing.

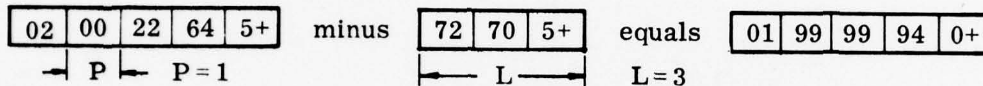
1. No Borrow:



Case format: e, f, g, or h

time = 10.99 basic
2 x 3.21 subtraction
+ 0.00 no borrow
17.41 microseconds total

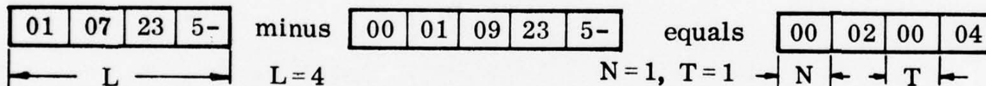
2. Borrow Without Recomplement:



Case format: e

time = 10.99 basic
3 x 3.21 subtraction
2.01 propagate borrow
+ 2.01 terminate borrow
24.64 microseconds total

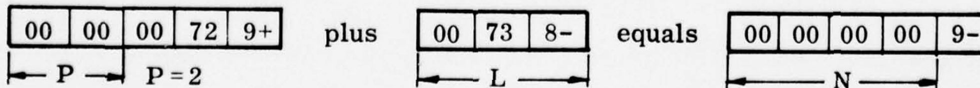
3. Recomplement:



Case format: g or h

time = 10.99 basic
4 x 3.21 subtraction
2.43 tens complement units and sign
1.33 tens complement zero
2.14 tens complement
+ 1.75 nines complement
31.48 microseconds total

4. Borrow and Recomplement:



L=3, N=4, T=0
Case format: f

time = 10.99 basic
3 x 3.21 subtraction
2 x 2.01 propagate borrow
2.43 tens complement units and sign
+ 4 x 1.75 nines complement
34.07 microseconds total

Instruction Set

DECIMAL SHIFT TIMING

Table A-6 lists times for Shift Left (SFTL) and Shift Right (SFTR) instructions.

Table A-6. Decimal Shift Timing Chart

<table>
<tr><th colspan="8">Shift Left Time in Microseconds</th><th colspan="8">Shift Right Time in Microseconds</th></tr>
<tr><th rowspan="2">Shift Count</th><th colspan="7">Field Length</th><th rowspan="2">Shift Count</th><th colspan="7">Field Length</th></tr>
<tr><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th></tr>
<tr><td>0</td><td colspan="7">11.14</td><td>0</td><td colspan="7">11.64</td></tr>
<tr><td>1</td><td rowspan="14">13.43</td><td>16.51</td><td>18.27</td><td>20.03</td><td>21.79</td><td>23.55</td><td>25.31</td><td>1</td><td rowspan="14">13.14</td><td>17.06</td><td>19.17</td><td>21.28</td><td>23.39</td><td>25.50</td><td>27.61</td></tr>
<tr><td>2</td><td>15.30</td><td>19.34</td><td>21.10</td><td>22.86</td><td>24.62</td><td>26.38</td><td>2</td><td>16.38</td><td>18.37</td><td>20.13</td><td>21.89</td><td>23.65</td><td>25.41</td></tr>
<tr><td>3</td><td rowspan="12">14.28</td><td>17.36</td><td>19.12</td><td>20.88</td><td>22.64</td><td>24.40</td><td>3</td><td rowspan="12">13.99</td><td>17.91</td><td>20.02</td><td>22.13</td><td>24.24</td><td>26.35</td></tr>
<tr><td>4</td><td>16.15</td><td>20.19</td><td>21.95</td><td>23.71</td><td>25.47</td><td>4</td><td>17.23</td><td>19.22</td><td>20.98</td><td>22.74</td><td>24.50</td></tr>
<tr><td>5</td><td rowspan="10">15.13</td><td>18.21</td><td>19.97</td><td>21.73</td><td>23.49</td><td>5</td><td rowspan="10">14.84</td><td>18.76</td><td>20.87</td><td>22.98</td><td>25.09</td></tr>
<tr><td>6</td><td>17.00</td><td>21.04</td><td>22.80</td><td>24.56</td><td>6</td><td>18.08</td><td>20.07</td><td>21.83</td><td>23.59</td></tr>
<tr><td>7</td><td rowspan="8">15.98</td><td>19.06</td><td>20.82</td><td>22.58</td><td>7</td><td rowspan="8">15.69</td><td>19.61</td><td>21.72</td><td>23.83</td></tr>
<tr><td>8</td><td>17.85</td><td>21.89</td><td>23.65</td><td>8</td><td>18.93</td><td>20.92</td><td>22.68</td></tr>
<tr><td>9</td><td rowspan="6">16.83</td><td>19.91</td><td>21.67</td><td>9</td><td rowspan="6">16.54</td><td>20.46</td><td>22.57</td></tr>
<tr><td>10</td><td>18.70</td><td>22.74</td><td>10</td><td>19.78</td><td>21.77</td></tr>
<tr><td>11</td><td rowspan="4">17.68</td><td>20.76</td><td>11</td><td rowspan="4">17.39</td><td>21.31</td></tr>
<tr><td>12</td><td>19.55</td><td>12</td><td>20.63</td></tr>
<tr><td>13</td><td>18.53</td><td>13</td><td>18.24</td></tr>
<tr><td>14</td><td>18.53</td><td>14</td><td>18.24</td></tr>
</table>

Extrapolation Formulae:

- Add 0.85 for each additional shift in zero character.
- Add 1.76 for each additional left-shifted character.
- Add 1.76 for each additional right-shifted character when shift count is even.
- Add 2.11 for each additional right-shifted character when shift count is odd.

Note: Field length (SL) is in actual number of characters (i.e. digit pairs) and will be 1 greater than the length specified value in the instruction.

CLASS C NON-IMPLEMENTED OP-CODES

All SUE 1111A and 1111B, class C instructions (Operation Codes 0, 1, 6, 7, D, E and F) trap. Time to trap takes 16.28 microseconds.

MOVE TIMING

Time (T) for MOVR, MOVL, and ZADD is calculated using the following formulae:

Mnemonic	Formula
MOVR or MOVL	T = 11.36 + 1.76N
ZADD	T = 11.36 + 1.76N + 0.85Z

where:

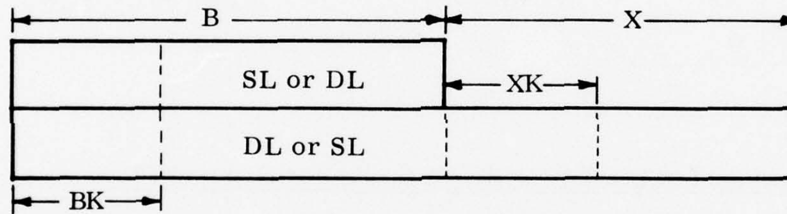
N is the number of characters (i. e. DL or SL, whichever is smaller)

Z is the number of zeros (Z = DL - SL if DL > SL).

Instruction Set

COMPARE-FIELD TIMING

The following general format applies to Compare-Field timing calculations.



where:

SL = Source Length

DL = Destination Length

X = Difference in Length in characters, SL - DL

B = Length of the shorter field, SL or DL

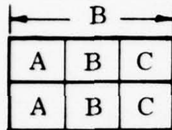
XK = Number of leading blank characters (b̸) in the extension field (X) before a non-blank character is encountered.

BK = Number of character pairs in the two body fields that are in corresponding positions, before a non-equal pair is encountered.

COMPARE FIELD TIME CALCULATIONS

Time (T) in microseconds can be calculated for the six distinct cases as follows:

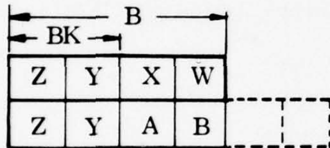
Body Fields Equal, SL = DL



$X = XK = 0$
$B = BK > 0$

$T = 11.35 + 2.4B$

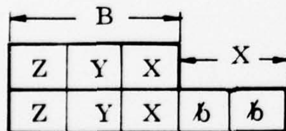
Unequal Pair in Body Scan



$B > BK \geq 0$

$T = 13.72 + 2.4BK$

Body Fields Equal, SL >DL, or SL< DL, Extension Field Blank

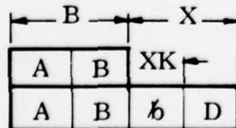


$B = BK > 0$
$X = XK > 0$

SL > DL:
$T = 11.64 + 2.4B + 1.2X$

SL < DL:
$T = 11.51 + 2.4B + 1.2X$

Body Fields Equal, SL >DL or SL< DL, Non-blank in Extension Field



$B = BK > 0$
$X > XK \geq 0$

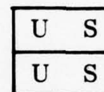
SL > DL:
$T = 13.39 + 2.4B + 1.2XK$

SL < DL:
$T = 13.26 + 2.4B + 1.2XK$

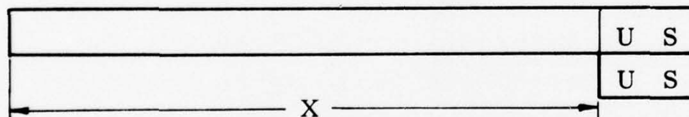
DECIMAL COMPARE TIMING

Decimal comparison is done in a manner to provide the fastest comparison result. The fields are scanned in the following order of significance: signs and units digits, extension fields, body fields. The extension field (of length X) is the most significant portion of the longer comparand and includes all digits of more significance than the most significant digit of the shorter comparand. The body fields (of length B) are those portions of the comparands remaining (i.e. not units or signs, or extension). Comparison may take one of four basic formats based on the relative lengths of the fields:

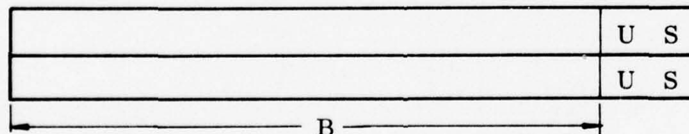
1. Source length = destination length = 1
   (both fields are units digits and signs).



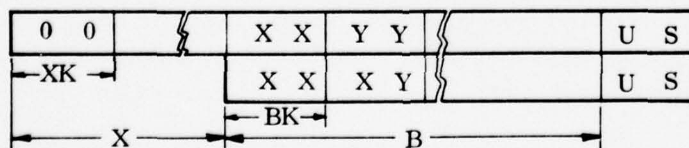
2. Source length ≠ destination length, but one field = 1
   (one comparand is units and sign).



3. Source length = destination length > 1
   (both fields are equal length and more than units and sign).



4. Source length ≠ destination length; both fields > 1
   (not equal length but both fields are more than units and sign).



where:

X is the difference in field lengths in characters (digit pairs).

B is the length of the shorter field minus one.

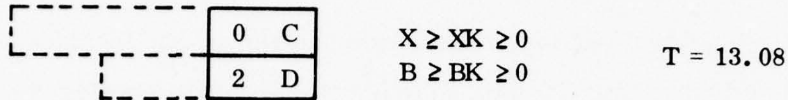
XK is the number of leading zero characters in the extension field.

BK is the number of leading characters in the body fields which are equal in corresponding positions (or are both zero in the case of Zero Scan), before non-equal (or non-zero) pair is encountered.

DECIMAL COMPARE TIME CALCULATIONS

Time (T) in microseconds can be calculated for three cases as follows:

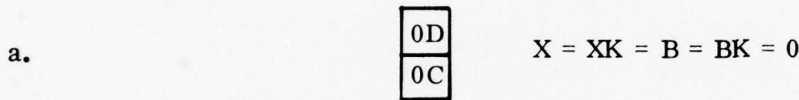
Case 1 - The signs are not equal and the units digits are not both zero.



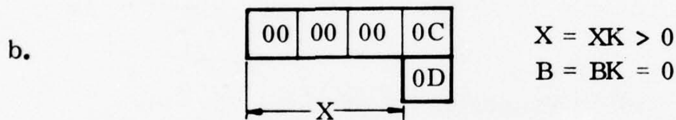
Case 2 - The signs are not alike and both units digits are zero. A zero scan is evoked.

Case 2 Formats:

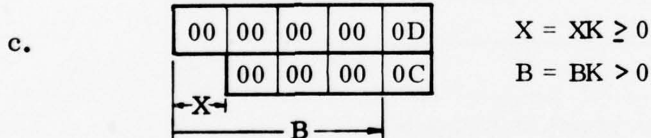
a.



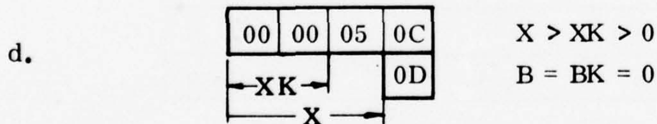
b.



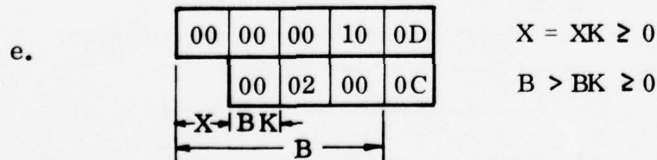
c.



d.



e.



DECIMAL COMPARE TIMING PROCEDURE FOR CASE 2

Step	For Case Formats	Conditions (Times in Microseconds)	Times (Microseconds)
1	a, b, c, d, e	Start with basic time of 14.40 and go to Step 2.	14.40
2	a, b, c, d, e	Add 0.13 if destination comparand sign is positive, and go to Step 3.	0.13
3	b, c, d, e	Add 0.03 if SL > DL, and go to Step 4.	0.03
4	b, c, d, e	Add 1.17 for each leading zero character in the extension field and go to Step 5.	1.17 XK
5	d	Add 1.69 and STOP (Note 1) if there is a non-zero character in the extension field; otherwise go to Step 6.	+1.69
6	a, b	Add 0.42 and STOP (Note 2) if B = 0 (i.e. no body) and if either the extension field is all zero ([X] = 0) or there is no extension field (X = 0); otherwise go to Step 7.	+0.42
7	c, e	Add 2.53 for each pair of leading zero characters in the body field. Go to Step 8.	2.53BK
8	e	Add 3.18 and STOP (Note 1) for the first non-zero character encountered in either body field; otherwise go to Step 9.	+3.18
9	c	Add 0.81 and STOP (Note 2) if both body fields are all zero (i.e. +0 and -0).	+0.81

Note 1 - The result is Greater-Than or Less-Than depending upon the signs.

Note 2 - The result is Equal (and specifically a positive zero equals a negative zero).

Instruction Set

Case 3 - The signs are alike. A compare scan is evoked.

Case 3 Formats:

f.

```
[7D]
[3D]
```

$X = XK = B = BK = 0$

g.

```
[00][00][00][2C]
|<----X---->[9C]
```

$X = XK > B = BK = 0$

h.

```
[00][02][37][5C]
    [02][37][4C]
|<X>|<---B--->|
```

$X = XK \geq 0$

$B = BK > 0$

j.

```
 [00][90][73][8D]
->|XK|<-     [8D]
 |<----X---->|
```

$X > XK \geq B = BK = 0$

k.

```
[00][02][39][7D]
    [02][30][2D]
|<X>|BK|<-
    |<--B-->|
```

$X = XK \geq 0$

$B > BK \geq 0$

DECIMAL COMPARE TIMING PROCEDURE FOR CASE 3

Step	For Case Formats	Conditions (Times in Microseconds)	Times (Microseconds)
1	f, g, h, j, k	Start with basic time of 14.01 and go to Step 2.	14.01
2	f, g, h, j, k	Add 0.13 if the destination comparand sign is positive and go to Step 3.	0.13
3	g, h, j, k	Add 0.03 if SL >DL and go to Step 4.	0.03
4	g, h, j, k	Add 1.17 for each leading zero character in the extension field and go to Step 5.	1.17XK
5	j	Add 1.98 and STOP (Note 3) if there is a non-zero character in the extension field; otherwise go to Step 6.	+1.98
6	f, g	Add 0.42 and STOP (Note 4) if B=0 (i.e. no body) and if either the extension field is all zero ([X]=0) or there is no extension field (X=0); otherwise, go to Step 7.	+0.42
7	h, k	Add 2.40 for each corresponding equal pair of leading characters in the body field and go to Step 8.	2.40BK
8	k	Add 4.34 and STOP (Note 5) for the first corresponding un-equal character pair in the body field; otherwise go to Step 9.	+4.34
9	h	Add 0.81 and STOP (Note 4) if all corresponding characters in the body fields are equal.	+0.81

Note 3 - Not equal: compare based on sign and which field is longer.
Note 4 - Compare based on units and signs only
Note 5 - Not equal: compare based on sign and un-equal body characters.

Instruction Set

AD-A060 256 BOLT BERANEK AND NEWMAN INC CAMBRIDGE MASS F/G 9/2
PLURIBUS DOCUMENT 4: BASIC SOFTWARE.(U)
SEP 78 M F KRALEY DCA200-77-C-0616
UNCLASSIFIED BBN-3001 NL

2 OF 2
AD
A060256

END
DATE
FILMED
12-78
DDC

Table A-7. SUE 1112A/B Instruction Times

Instruction		Execution Time in Microseconds	
Bit Manipulation		Selected by	
		XR or K	Mask w/wo XR
RBIT	Make the designated bit a zero	3.24	3.30
SBIT	Make the designated bit a one	3.24	3.30
CBIT	Change (complement) the designated bit	3.24	3.30
IBIT	Isolate (extract) the designated bit	3.24	3.30
TSBT	Test the designated bit and shift left	3.27	3.33
TBIT	Only test the designated bit	3.24	3.30
Move		Operand Value	
		Positive	Negative
NEGT	Move the twos complement value	2.49	2.49
CPLM	Move the ones complement value	2.49	2.49
MOVP	Move the positive magnitude	2.75	2.88
MOVN	Move the negative magnitude	2.75	2.88
Normalize and Count			
SxxN	Single Normalize, AC = 0	2.82	
SLAN	Single Left Arithmetic Normalize	3.24 + 0.32 per shift	
SLLN	Single Left Logical Normalize	2.95 + 0.29 per shift	
SRAN	Single Right Arithmetic Normalize	2.95 + 0.29 per shift	
SRLN	Single Right Logical Normalize	2.95 + 0.29 per shift	
DxxN	Double Normalize, both AC's = 0	3.89	
DLAN	Double Left Arithmetic Normalize	5.21 + 0.61 per shift	
DLLN	Double Left Logical Normalize	4.89 + 0.48 per shift	
DRAN	Double Right Arithmetic Normalize	5.18 + 0.97 per shift	
DRLN	Double Right Logical Normalize	4.31 + 0.81 per shift	

Table A-7. SUE 1112A/B Instruction Times (continued)

Instruction		Execution Time in Microseconds
Double-Length Shift		
DLAO	Double Left Arithmetic Open	5.08 + 0.61 per shift
	With Zero Count	3.95
DLLL	Double Left Logical Linked	4.50 + 0.32 per shift
	With Zero Count	3.89
DLLO	Double Left Logical Open	3.95 + 0.48 per shift
DLLC	Double Left Logical Closed	4.11 + 0.64 per shift
DRAO	Double Right Arithmetic Open	4.11 + 0.97 per shift
DRLL	Double Right Logical Linked	3.89 + 0.81 per shift
DRLO	Double Right Logical Open	3.50 + 0.81 per shift
DRLC	Double Right Logical Closed	3.76 + 1.07 per shift

Table A-8. SUE 1112A/B Single- and Double-Precision Fixed-Point Instruction Times

	Store	Load	Add	Sub	Multiply (See notes)		Divide (See notes)
	DSTA	DLOD	DADD	DSUB	MLTA	MULT	DIVD
Register Address (Indexed)	5.06	4.44	5.96	5.96	≈16.87	≈15.35	≈15.29
Extended (w/wo Indexing)	5.13	4.54	6.06	6.06	≈16.97	≈15.45	≈15.39
Register Address, Indirect*	7.04	6.45	7.97	7.97	≈18.88	≈17.36	≈17.30
Extended, Indirect*	6.72	6.13	7.65	7.65	≈18.56	≈17.04	≈16.98

Notes:

*Add 1.01 for each additional level of indirect

DIVIDE

1. Assumes divisor is positive and quotient is positive and even
2. Time is 4.10 if divide check occurs
3. Add 0.29 if divisor is negative
4. Subtract 0.06 if quotient is negative
5. Subtract 0.13 if quotient is odd
6. Total range (except divide check) for extended direct divide is 15.68 to 15.20

MULTIPLY

7. Assumes typically seven 'one' bits in the absolute value of the multiplier; if more (or less) add (or subtract) N x 0.03 to the time, where N is the additional number of significant multiplier bits.
8. Add 0.19 if the product is negative
9. Total range for extended direct multiply (w/o accumulate) is 15.24 to 15.88

APPENDIX B

INSTRUCTION SUMMARY AND INDEX

SUE 1110 (BASIC) INSTRUCTIONS SUMMARY

Status Register

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
L4	L3	L2	L1	A	N	Z	O	LP	F3	F2	F1	C	V	G	E

Control Instructions — Class O

15 14 13 12	11 10 9 8		7 6 5 4	3 2 1 0	MNEMONIC
0	0		x x x x	x x x x	HALT
0	2		0 F3 F2 F1	C V G E	RSTS
0	2		1 F3 F2 F1	C V G E	SETS
0	8		0	L4 L3 L2 L1	ENBL
0	8		4	L4 L3 L2 L1	ENBW
0	8		8	L4 L3 L2 L1	DSBL
0	8		C	L4 L3 L2 L1	DSBW
0	B	1	D		STSM
0	B	3	D		REGM
0	B	4	D		RETN
0	B	5	D		MSTS
0	P	7	D		MREG

B — 0 if Absolute, 1 if Relative.

D — Address field (words), two's complement form for Relative.

General Register Instructions — Classes 1-7

		15 14 13 12	11	10 9 8	7	6 5 4	3	2 1 0
Accumulator To Memory	Auto Decrement	1	B	OP	I	AR	E	XR
	Auto Increment	2	B	OP	I	AR	E	XR
		3	B	OP	I	AR	E	XR
	Jump	4	0	0	I	0	E	XR
	Jump to Subroutine	4	0	0	I	AR	E	XR
	Data to Accumulator	4	1	OP	0	AR	E	XR
		4	1	OP	1	AR	K	
Memory To Accumulator	Auto Decrement	5	B	OP	I	AR	E	XR
	Auto Increment	6	B	OP	I	AR	E	XR
		7	B	OP	I	AR	E	XR

Sequence , E, XR, I, B, +.

OP — Operation Code:

0	MOV	Move
1	SUB	Subtraction
2	ADD	Addition
3	AND	Logical Product
4	IOR	Logical Inclusive Or
5	EOR	Logical Exclusive Or
6	CMP	Compare
7	TST	Test

- B Word mode if 0, byte mode if 1.
- I Indirect when 1.
- AR Accumulator Register designator (0-7).
- E Extended or two word instruction if 1.
- XR Index Register designator (1-7), no indexing if 0.
- K 4 bit literal constant data.

Branch Instructions — Classes 8-9

	15 14 13 12	11 10 9 8	7 6 5 4 3 2 1 0
No Operation	8	0	x x x x x x x x
Branch Unconditional	9	0	D
Branch False	8	T	D
Branch True	9	T	D

D — Displacement word address in two's complement form.

T	INDICATOR	MNEMONIC
1	Equal	BEQx
2	Greater Than	BGTx
3	Overflow	BOVx
4	Carry	BCYx
5	Flag 1	BF1x
6	Flag 2	BF2x
7	Flag 3	BF3x
8	Loop Complete	BLPx
9	Odd	BODx
A	Zero	BZEx
B	Negative	BNGx
C	Less Than	BLTx
D, E, F.	Unimplemented	

X = T or F

Shift Instructions — Class A

15 14 13 12	11	10	9 8	7	6 5 4	3	2 1 0
A	B	L/R	OP	0	AR	X	XR
A	B	L/R	OP	1	AR	K	

- B 0 = single length, 1 = double length (not implemented on 1110).
- AR Accumulator Register designator (to be shifted).
- K Shift Count
- XR Shift Count Source Register.
- L/R Left (L) when 0, Right (R) when 1.

OP Shift Operation Code:
- 00 = Arithmetic
- 01 = Logical Linked
- 10 = Logical Open
- 11 = Logical Closed

Instruction

APPENDIX B

INSTRUCTION SUMMARY AND INDEX

SUE 1110 (BASIC) INSTRUCTIONS SUMMARY

Instruction Set

Status Register

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
L4	L3	L2	L1	A	N	Z	O	LP	F3	F2	F1	C	V	G	E

Control Instructions — Class 0

15 14 13 12	11	10 9 8	7 6 5 4	3 2 1 0	MNEMONIC
0	0		x x x x	x x x x	HALT
0	2		0 F3 F2 F1	C V G E	RSTS
0	2		1 F3 F2 F1	C V G E	SETS
0	8		0	L4 L3 L2 L1	ENBL
0	8		4	L4 L3 L2 L1	ENBW
0	8		8	L4 L3 L2 L1	DSBL
0	8		C	L4 L3 L2 L1	DSBW
0	B	1	D		STSM
0	B	3	D		REGM
0	B	4	D		RETN
0	B	5	D		MSTS
0	P	7	D		MREG

B — 0 if Absolute, 1 if Relative.
D — Address field (words), two's complement form for Relative.

General Register Instructions — Classes 1-7

		15 14 13 12	11	10 9 8	7	6 5 4	3	2 1 0
Accumulator To Memory	Auto Decrement	1	B	OP	I	AR	E	XR
	Auto Increment	2	B	OP	I	AR	E	XR
		3	B	OP	I	AR	E	XR
	Jump	4	0	0	I	0	E	XR
	Jump to Subroutine	4	0	0	I	AR	E	XR
	Data to Accumulator	4	1	OP	0	AR	E	XR
		4	1	OP	[illegible]	AR	K	
Memory To Accumulator	Auto Decrement	5	B	OP	I	AR	E	XR
	Auto Increment	6	B	OP	I	AR	E	XR
		7	B	OP	I	AR	E	XR

Sequence , E, XR, I, B, +

OP — Operation Code			
	0	MOV	Move
	1	SUB	Subtraction
	2	ADD	Addition
	3	AND	Logical Product
	4	IOR	Logical Inclusive Or
	5	EOR	Logical Exclusive Or
	6	CMP	Compare
	7	TST	Test

- B — Word mode if 0, byte mode if 1.
- I — *Indirect when 1.*
- AR — Accumulator Register designator (0-7).
- E — Extended or two word instruction if 1.
- XR — Index Register designator (1-7), no indexing if 0.
- K — 4 bit literal constant data.

Branch Instructions — Classes 8-9

	15 14 13 12	11 10 9 8	7 6 5 4 3 2 1 0
No Operation	8	0	x x x x x x x x
Branch Unconditional	9	0	D
Branch False	8	T	D
Branch True	9	T	D

D — Displacement word address in two's complement form.

T	INDICATOR	MNEMONIC
1	Equal	BEQx
2	Greater Than	BGTx
3	Overflow	BOVx
4	Carry	BCYx
5	Flag 1	BF1x
6	Flag 2	BF2x
7	Flag 3	BF3x
8	Loop Complete	BLPx
9	Odd	BODx
A	Zero	BZEx
B	Negative	BNGx
C	Less Than	BLTx
D, E, F.	*Unimplemented*	

X = T or F

Shift Instructions — Class A

15 14 13 12	11	10	9 8	7	6 5 4	3	2 1 0
A	B	L/R	OP	0	AR	X	XR
A	B	L/R	OP	1	AR	K	

- B — 0 = single length, 1 = double length (not implemented on 1110).
- AR — Accumulator Register designator (to be shifted).
- K — Shift Count
- XR — Shift Count Source Register.
- L/R — Left (L) when 0, Right (R) when 1.

OP Shift Operation Code:
- 00 = Arithmetic
- 01 = Logical Linked
- 10 = Logical Open
- 11 = Logical Closed

SUE 1110 (BASIC) INSTRUCTION INDEX

		Page
A. GENERAL REGISTER INSTRUCTIONS		
ADDB	Add byte	12
SUBB	Subtract byte	12
CMPB	Compare byte	13
ANDB	Logical Product byte (and)	12
IORB	Logical sum byte (inclusive or)	12
EORB	Logical difference byte (exclusive or)	13
TSTB	Test byte	13
MOVB	Move byte	12
ADDW	Add word	12
SUBW	Subtract word	12
CMPW	Compare word	13
ANDW	Logical product word (and)	12
IORW	Logical sum word (inclusive or)	12
EORW	Logical difference word (exclusive or)	13
TSTW	Test word	13
MOVW	Move word	12
B. JUMP INSTRUCTIONS		
JSBR	Jump to subroutine	16
JUMP	Jump to location	16
C. BRANCH CONDITIONAL INSTRUCTIONS		
NOPR	No operation	22
BRUN	Branch unconditional	23
BEQT	Branch if equal true	21
BGTT	Branch if greater than true	21
BLTT	Branch if less than true	21
BZET	Branch if zero true	21
BNGT	Branch if negative true	21
BLPT	Branch if loop true	21
BODT	Branch if odd true	21
BOVT	Branch if overflow true	21
BCYT	Branch if carry true	21
BF1T	Branch if flag 1 true	21
BF2T	Branch if flag 2 true	21
BF3T	Branch if flag 3 true	21

Page

BEQF Branch if equal false 21
BGTF Branch if greater than false 21
BLTF Branch if less than false 21
BZEF Branch if zero false 21
BNGF Branch if negative false 21
BLPF Branch if loop false 21
BODF Branch if odd false 21
BOVF Branch if overflow false 21
BCYF Branch if carry false 21
BF1F Branch if flag 1 false 21
BF2F Branch if flag 2 false 21
BF3F Branch if flag 3 false 21

D. SHIFT INSTRUCTIONS

SLAO Single left arithmetic open 26
SLLO Single left logical open 26
SLLC Single left logical closed 26
SLLL Single left logical linked 26
SRAO Single right arithmetic open.............. 27
SRLO Single right logical open 27
SRLC Single right logical closed 27
SRLL Single right logical linked 27

E. CONTROL INSTRUCTIONS

RETN Return from interrupt 32
STSM Status to memory 32
REGM Registers to memory 32
MSTS Memory to status 32
MREG Memory to registers 33
HALT Halt the computer 29
WAIT Wait for interrupt 31
DSBL Disable interrupts 31
DSBW Disable interrupts and wait 31
ENBL Enable interrupts 30
ENBW Enable interrupts and wait.............. 30
SETS Set programmable status indicators 29
RSTS Reset programmable status indicators....... 29

Instruction Set

Page

SUE 1110A INSTRUCTIONS

Includes all instructions listed under SUE 1110 (Basic) and the following:

SKEY Store Key 35

SUE 1110B INSTRUCTIONS

Includes all instructions listed under SUE 1110 (Basic), 1111A, and the following:

FCLW Fetch and Clear Word 37
FCLB Fetch and Clear Byte 37

SUE 1111A INSTRUCTIONS

Includes all instructions listed under SUE 1110 (Basic), 1110A, and the following:

ZADD Zero and Add 42
ADDD Add 42
SUBD Subtract 42
CMPD Compare Decimal 43
SFTR Shift Right 43
MOVR Move Right 44
SFTL Shift Left 44
MOVL Move Left 44
COMP Compare Field 45

SUE 1111B INSTRUCTIONS

Includes all instructions listed under SUE 1110 (Basic), 1110A, 1111A and the following:

FCLW Fetch and Clear Word 46
FCLB Fetch and Clear Byte 46

Page

SUE 1112A INSTRUCTIONS

Includes all instructions listed under SUE 1110 (Basic), 1110A, and the following:

A. BIT MANIPULATION INSTRUCTIONS

RBIT	Make the Designated Bit a Zero	52
SBIT	Make the Designated Bit a One	52
CBIT	Change (Complement) the Designated Bit	52
IBIT	Isolate (Extract) the Designated Bit	52
TSBT	Test the Designated Bit and Shift Left	52
TBIT	Only Test the Designated Bit	52

B. MOVE INSTRUCTIONS

NEGT	Move the Twos Complement Value	53
CPLM	Move the Ones Complement Value	53
MOVP	Move the Positive Magnitude	53
MOVN	Move the Negative Magnitude	53

C. NORMALIZE AND COUNT INSTRUCTIONS

SLAN	Single Left Arithmetic Normalize	54
SLLN	Single Left Logical Normalize	54
SRAN	Single Right Arithmetic Normalize	55
SRLN	Single Right Logical Normalize	55
DLAN	Double Left Arithmetic Normalize	55
DLLN	Double Left Logical Normalize	56
DRAN	Double Right Arithmetic Normalize	56
DRLN	Double Right Logical Normalize	56

D. DOUBLE LENGTH SHIFT

DLAO	Double Left Arithmetic Open	58
DLLL	Double Left Logical Linked	58
DLLO	Double Left Logical Open	58
DLLC	Double Left Logical Closed	58
DRAO	Double Right Arithmetic Open	59
DRLL	Double Right Logical Linked	59
DRLO	Double Right Logical open	59
DRLC	Double Right Logical Closed	59

Page

E. SINGLE PRECISION FIXED POINT

MLTA	Multiply and Add	61
MULT	Multiply (no add)	61
DIVD	Divide	61

F. DOUBLE PRECISION FIXED POINT

DLOD	Double Load Accumulator	63
DSTA	Double Store Accumulator	63
DADD	Double Add	63
DSUB	Double Subtract	63

G. CONTROL INSTRUCTIONS

SKEY	Store Value (K) in Key Bits	64
JKEY	Store Value (K) in Key Bits and Address M into Program Counter	64
LCPU	Load Processor Number into (XR) Bits 5 and 6	64
LKEY	Load Key Bits into (XR)	64

SUE 1112B INSTRUCTIONS

Includes all instructions listed under SUE 1110 (Basic), 1110A, 1112A, and the following:

FCLW	Fetch and Clear Word	65
FCLB	Fetch and Clear Byte	65

APPENDIX C

INPUT/OUTPUT ADDRESSES

Table C-1. Input-Output Device Addresses*

Address (Hex)	Input/Output Device Controller
F800	Teletypewriter No. 1
F810	Teletypewriter No. 2
F820	High Speed Paper Tape Reader No. 1
F830	High Speed Paper Tape Punch No. 1
F840	High Speed Paper Tape Reader No. 2
F850	High Speed Paper Tape Punch No. 2
F860	Card Reader No. 1
F870	Card Reader No. 2
F880	Card Punch No. 1
F890	Card Punch No. 2
F8A0	Line Printer No. 1
F8B0	Line Printer No. 2
F8C0	Magnetic Tape No. 1 (handles 4 Drives)
F8D0	Magnetic Tape No. 2 (handles 4 Drives)
F8E0	Bulk File No. 1 (Fixed Head)
F8F0	Bulk File No. 2 (Fixed Head)
F900	Disk File Unit No. 1 (Fixed and Removable)
F910	Disk File Unit No. 2 (Fixed and Removable)
F920	Cassette No. 1
F930	Cassette No. 2
FA00	CRT Display, Alphanumeric No. 1
FA10	CRT Display, Alphanumeric No. 2
⋮	⋮
[illegible]	CRT Display, Alphanumeric No. 16
FF90	Input Keyboard No. 1, Business System
FFA0	CRT Display No. 1, Business System

*Note: Device address assignment is variable by jumper wires connected on each controller. The addresses shown are recommended and are subject to change.

APPENDIX D

SELF-INTERRUPT AND SYSTEM INTERRUPT EXECUTIVE SPACE

Interrupt	Level				
SELF INTERRUPT	6	ADDRESS THAT CAUSED ABORT ADDR00 0028 ADDR10 0048 ADDR01 0038 ADDR11 0058	STATUS ADDR00 002A ADDR10 004A ADDR01 003A ADDR11 005A	ABORTED INSTRUCTION ADDRESS ADDR00 002C ADDR10 004C ADDR01 003C ADDR11 005C	SERVICE ROUTINE VECTOR ADDR00 002E ADDR10 004E ADDR01 003E ADDR11 005E
	5	UNIMPLEMENTED INSTRUCTION ADDR00 0020 ADDR10 0040 ADDR01 0030 ADDR11 0050	STATUS ADDR00 0022 ADDR10 0042 ADDR01 0032 ADDR11 0052	UNIMPLEMENTED INSTRUCTION ADDRESS ADDR00 0024 ADDR10 0044 ADDR01 0034 ADDR11 0054	SERVICE ROUTINE VECTOR ADDR00 0026 ADDR10 0046 ADDR01 0036 ADDR11 0056
SYSTEM INTERRUPT	4	MODULE ADDRESS 0018	STATUS 001A	PROGRAM COUNTER 001C	SERVICE ROUTINE VECTOR 001E
	3	MODULE ADDRESS 0010	STATUS 0012	PROGRAM COUNTER 0014	SERVICE ROUTINE VECTOR 0016
	2	MODULE ADDRESS 0008	STATUS 000A	PROGRAM COUNTER 000C	SERVICE ROUTINE VECTOR 000E
	1	MODULE ADDRESS 0000	STATUS 0002	PROGRAM COUNTER 0004	SERVICE ROUTINE VECTOR 0006

1110-R03-72

Instruction Set

APPENDIX E

USASCII CHARACTER SET AND HEXADECIMAL CODES

HEX	CHARACTER	HEX	CHARACTER
A0	space	C1	A
A1	!	C2	B
A2	"	C3	C
A3	#	C4	D
A4	$	C5	E
A5	%	C6	F
A6	&	C7	G
A7	' (apostrophe)	C8	H
A8	(	C9	I
A9	)	CA	J
AA	*	CB	K
AB	+	CC	L
AC	, (comma)	CD	M
AD	-	CE	N
AE	. (period)	CF	O
AF	/	D0	P
B0	0	D1	Q
B1	1	D2	R
B2	2	D3	S
B3	3	D4	T
B4	4	D5	U
B5	5	D6	V
B6	6	D7	W
B7	7	D8	X
B8	8	D9	Y
B9	9	DA	Z
BA	:	DB	[left bracket
BB	;	DC	\ back slash
BC	< less than	DD	] right bracket
BD	=	DE	↑ up arrow
BE	> greater than	DF	← left arrow
BF	?		
C0	@	87	bell
		8A	line feed
		8D	carriage return

PLURIBUS DOCUMENT 4: BASIC SOFTWARE

PART 2: INTRODUCTION TO ASSEMBLY LANGUAGE

Intro to Assembly Language

Update History:

Originally written by S. Jeske, November 1975.

Revised by R. Hinden, August 1978.

Intro to Assembly Language

TABLE OF CONTENTS

page

1.0 INTRODUCTION 1
2.0 INSTRUCTION SETS 1
2.1 Memory Reference Instructions 1
2.1.1 Word Mode 1
2.1.2 Byte Mode 2
2.1.3 Indexed Mode 3
2.1.4 Indexed-Extended Mode 3
2.1.5 Indirect Mode 4
2.1.6 Indexed-Indirect Mode 4
2.1.7 Extended-Indexed-Indirect Mode 5
2.1.8 Auto-Increment Mode 5
2.1.9 Auto-Decrement Mode 6
2.1.10 Complex Example 6
2.1.11 Multilevel Indirect Mode 7
2.2 Program Transfers 7
2.2.1 The Jump Instruction 7
2.2.1.1 Jump Direct 7
2.2.1.2 Jump Indirect 7
2.2.1.3 Jump Indexed 8
2.2.1.4 Exotic Jumps 8
2.2.2 The Jump-to-Subroutine Instruction 8
2.2.3 Branches 9
2.3 Register-to-Register Instructions 11
2.4 Immediate Operands 11
2.5 Shift Instructions 12
2.6 Control Instructions 12
2.6.1 Halt 12
2.6.2 Immediate Operand Control Instructions . . 13
2.6.3 Address Operand Control Instructions . . . 13

TABLE OF CONTENTS (cont'd)

page

3.0 ASSEMBLER COMMANDS 14

3.1 Numbers 14

3.1.1 Radix 14

3.1.2 Auxiliary Information 14

3.2 Numeric Operations 15

3.3 Program Locations 15

3.4 Current Location 16

3.5 General Location 16

3.5.1 Implicit 16

3.5.2 Explicit 17

3.6 General Assignment 17

3.7 Data 17

3.7.1 General Data 17

3.7.2 Byte Data 18

3.7.3 Character Data 18

3.8 Program Termination 18

3.9 Macros 18

3.10 Macro Packages 19

3.11 Format 19

INTRODUCTION TO ASSEMBLY LANGUAGE

1.0 Introduction

The assembler used to translate source code for the Pluribus multiprocessor into executable machine code is called PLURIBUS and runs on any TENEX system. This paper is not intended to be the definitive reference on this assembler, but rather to impart a general familiarity to the extent that a program listing could be examined and mostly understood. While familiarity with the PLURIBUS assembler is not assumed, it will be helpful if the reader has knowledge of some assembler, or at least the concepts of one. Also, some knowledge of the SUE instruction set is assumed (see the G3 Reference Manual). The interested reader is referred to the assembler Reference Manual in Pluribus Document 5 for more detailed information than that presented here, especially for the more uncommon commands which are not discussed here at all.

2.0 Instruction Sets

2.1 Memory Reference Instructions

2.1.1 Word Mode

By far the most common types of instructions are those that reference memory locations. There are six classes (1,2,3,5,6,7) distinguished by the left four bits of the instruction. Each of the six has one of eight possible operation codes -- Move, Subtract, Add, Logical And, Logical Inclusive Or, Logical Exclusive Or, Compare, and Test.

For instance, an instruction to add the 16-bit contents of location 100 to Register 1 would appear:

```
ADD R1,100
```

The machine language product would be: 0111001000011000 in the first word (of 2) in memory, and the address (the 100) in the second. For convenience, the binary machine language is hardly ever referred to, and the more convenient numerical representation is used. The PLURIBUS assembler normally outputs listings in octal, where the 7218 would instead appear as 071030, but there is a command option to select hex listings. As to what actual address the number 100 in the above example refers, that is more variable than you would expect, and will be covered

Intro to Assembly Language

later. It is not important usually, since most programs use symbolic addresses.

If the programmer, by methods which will be covered later, caused the symbol XYZ to have the value 100, he would probably write the above example as:

```
ADD R1,XYZ
```

If he wanted instead to add the 16-bit contents of Register 1 to the 16-bit contents of XYZ, with the new answer replacing the old value in XYZ, he would have:

```
ADDM R1,XYZ
```

The above instruction would assemble as 3218 (hex) in the first word, and 100 in the second. Note that the direction of data movement is given by the presence or absence of an "M".

The above convention is true for six of the other seven op codes. For SUB, AND, IOR, EOR, CMP, and TST, the "M" is made part of the op code. For the Move operation it is still technically possible to have MOV and MOVM, but most often the mnemonic LDA is used for the memory-to-register direction, and STA for the register-to-memory direction.

2.1.2 Byte Mode

If, instead of adding 16-bit words, we wish to subtract an 8-bit data byte found in memory location XYZ from Register 1, we have:

```
SUBB R1,XYZ
```

This instruction would be assembled as 7918 (hex) in the first word and the address in the second. An interesting thing to note is that now XYZ could be 101 or any other odd location, whereas for word operations the address must always be even.

If we wish to subtract the low order 8-bit data byte in Register 1 from the 8-bit contents at location XYZ, we have:

```
SUBBM R1,XYZ
```

The instruction would be assembled as 3918 (hex) in the first word, and the address in the second.

All of the following addressing modes have their byte mode counterparts in the above manner, therefore, we will not mention each one explicitly.

2.1.3 Indexed Mode

It is not necessary to have an explicit address (the XYZ in the above examples). You could have the address in an index register; this has the advantages of less required program storage and faster execution time. The index register is any one of the seven general registers that has been loaded previously with the number of the desired address location. If we want to AND the contents of memory location 100 into Register 3, and Register 2 has the number 100 in it, we can write:

```
AND R3,(R2)
```

This instruction would be assembled as 7332 (hex)

If we want to AND the contents of Register 3 into memory location 100, and Register 2 has the number 100 in it, we can write:

```
ANDM R3,(R2)
```

This instruction would be assembled as 3332 (hex). Note that in either case the whole instruction would take up only one word of memory, as the second word for the explicit address is not needed.

2.1.4 Indexed-Extended Mode

It is possible to combine the basic mode (with the explicit address) and the indexed mode (with the address in an index register). In this case, the contents of the index register are added to the explicit address to obtain the effective address. If XYZ has a value of 100, Register 4 has a 42 in it, and we wish to EOR the contents of location 142 into Register 5, we can write:

```
EOR R5,XYZ(R4)
```

This would be assembled as 755C (hex) with a 100 in the next word.

If we wish the results of the EOR to end up in memory, we have:

```
EORM R5,XYZ(R4)
```

This would be assembled as 355C (hex) with a 100 in the second word, and would EOR into location 142.

2.1.5 Indirect Mode

It is possible to address the memory locations indirectly. If we wish to IOR the contents of 150 into Register 6, location 100 has a 150 in it, and the symbol XYZ has a value of 100, we can write:

```
IOR R6,@XYZ
```

This would be assembled as 74E8 (hex) with a 100 in the next word.

If we wish to IOR to memory, it looks like this:

```
IORM R6,@XYZ
```

and is assembled as 34E8 (hex) with the 100 in the next word. It would IOR into location 150.

2.1.6 Indexed-Indirect Mode

It is possible to combine the indexed mode with the indirect mode. If we want to load the contents of 200 into Register 7, location 300 has a 200 in it, and Register 3 has a 300 in it, we can write:

```
LDA R7,@(R3)
```

This would be assembled as 70F3 (hex). Note the order: the indexing happens first, then the indirecting.

If we want to go the other way, it is:

```
STA R7,@(R3)
```

This would be assembled as 30F3 (hex).

2.1.7 Extended-Indexed-Indirect Mode

It is possible to combine the explicit address mode with the indexed-indirect mode. If we wish to CMP the contents of Register 4 with the contents of location 340, location 240 has a 340 in it, Register 2 has a 140 in it, and the symbol XYZ has a value of 100, we write:

```
CMP R4,@XYZ(R2)
```

This would be assembled as 76CA (hex), with a 100 in the second word. Note that, as always, the indexing happens first, then the indirecting.

If we wish to compare in the other order we write:

```
CMPM R4,@XYZ(R2)
```

This would be assembled as 36CA (hex), with 100 in the second word. Note that the only difference, in the case of the CMP, is which flags get set if they are unequal. Similarly, there is no difference between a TST register-to-memory, and a TST memory-to-register.

Intro to Assembly Language

2.1.8 Auto-Increment Mode

In all the above cases where an index register is used, it is possible to automatically increment the contents of the index register after they are used in the address calculation. It will be incremented by one if in byte mode, and by two if in word mode. If the example in Section 2.1.3 were done in auto-increment mode it would look like this:

```
AND R3,(R2)+
```

and would be assembled as 6332 (hex). Register 2, which started the instruction with 100 in it, would end up with 102 in it. The contents of 100 would still be AND-ed into Register 3, since the incrementation happens <u>after</u> the effective address calculation is done.

The other direction of movement would look like:

```
ANDM R3,(R2)+
```

and would be assembled as 2332 (hex). Register 3 would be AND-ed into 100, and Register 2 would still end up with a 102 in it.

2.1.9 Auto-Decrement Mode

Similar to the auto-increment mode, in all cases where an index register is used, it is possible to have the contents of the index register automatically decremented before they are used in the effective address calculation. They will be decremented by one if in byte mode, and by two if in word mode. If the example in Section 2.1.4 were done in auto-decrement mode it would look like this:

```
EOR R5,XYZ(-R4)
```

This would be assembled as 555C (hex), with a 100 in the next word. But note that a different memory word would be EOR-ed into Register 5 than was the case in Section 2.1.4. If XYZ has a value of 100, and Register 4 has a 42 in it before the instruction is executed, after the instruction is executed Register 4 would have a 40 in it, and we would have EOR-ed into Register 5 the contents of the memory word 140, because the auto-decrement happens <u>before</u> the address calculation.

If we want to EOR to memory, we write:

```
EORM R5,XYZ(-R4)
```

This is assembled as a 155C (hex), with a 100 in the next word. We would EOR the contents of Register 5 into location 140, and Register 4 would have 40 in it after the instruction.

2.1.10 Complex Example

If we wish to have the maximum number of addressing modes present at once we can write:

```
ADDBM R1,@XYZ(R2)+
```

This is assembled as 2A9A (hex), with a 100 in the second word, assuming XYZ has a value of 100. If Register 2 had a 40 in it before the instruction was executed, it would have a 41 in it after the instruction was executed. If address 140 had a 201 in it, the right byte of Register 1 would be added to the data byte at location 201.

2.1.11 Multilevel Indirect Mode

One addressing mode you will hardly ever see is that of the multilevel indirect address. This mode is available only in word mode, not byte mode, and is available in every mode that uses indirect addressing. It causes the processor to continue to indirect down an address chain if the low-order bit of the word fetched is on. If we wish to store the contents of Register 4 into location 350, location 100 contains 150+1 or 151, location 150 contains 250+1 or 251, location 250 contains a 350, and XYZ has a value of 100, we can write:

```
STA R4,@XYZ
```

This would be assembled as 30C8 (hex), with a 100 in the following word.

2.2 Program Transfers

2.2.1 The Jump Instruction

2.2.1.1 Jump Direct

In normal program flow the instruction immediately after the one currently being executed will be executed next. It is possible to alter this normal flow, however, such that the next instruction is somewhere else; and one way is by use of the Jump instruction. If we wish to have the next instruction executed be the one at location 100, and XYZ has a value of 100, we write:

```
JMP XYZ
```

This is assembled as 4008 (hex) in the first word, with a 100 in the second word.

2.2.1.2 Jump Indirect

If we wish to Jump to location 140, location 100 has a 140 in it, and XYZ has a value of 100, we write:

```
JMP @XYZ
```

This is assembled as 4088 (hex) in the first word, with 100 in the second word. It is also possible to have a multilevel indirect Jump.

Intro to Assembly Language

2.2.1.3 Jump Indexed

If we wish to Jump to location 240, and Register 2 has a 240 in it, we write:

```
JMP (R2)
```

This is assembled as 4002 (hex).

2.2.1.4 Exotic Jumps

As in the case of the memory reference instructions, we can combine the indexed mode with either direct or indirect addressing. For instance, if we want to Jump to location 250, Register 3 has a 50 in it, location 150 a 250, and XYZ has a value of 100, we write:

```
JMP @XYZ(R3)
```

This is assembled as 408B (hex) in the first word, with a 100 in the second word.

2.2.2 The Jump-to-Subroutine Instruction

Frequently it is desirable to be able to return to the instruction after a Jump and continue the normal program flow. The Jump-to-Subroutine instruction allows this by loading a specified Register with the address of the next instruction, and then jumping to the desired address. It has all the same addressing modes as the Jump instruction, the only difference being that now a Register is specified. The example in Section 2.2.1.1 appears as follows, if we want to load Register 7 with the "return address" first:*

```
JSB R7,XYZ
```

It is assembled as 4048 (hex) in the first word, with the 100 in the second word. The address of the word after the second word of the instruction would be in Register 7 after the instruction is executed.

* By convention, most subroutines written for the Pluribus make use of R7 as a linkage register.

The example in Section 2.2.1.2 would appear:

```
JSB R7,@XYZ
```

It would be assembled as 40C8 (hex) in the first word, with the 100 in the next word.

The example in Section 2.2.1.3 would appear:

```
JSB R7,(R2)
```

and would be assembled as 4042 (hex).

The complex example in Section 2.2.1.4 would appear:

```
JSB R7,@XYZ(R3)
```

This would be assembled as 40CB (hex) in the first word, with 100 in the second word.

In all the above cases the instructions jump to exactly the same addresses as their Jump counterparts. The only difference is that first Register 7 is loaded with the address of the next instruction. Note that the address will be the address immediately after the JSB if there is no extended address word (e.g., no XYZ). Multilevel indirect Jump-to-Subroutine instructions also work.

Intro to Assembly Language

2.2.3 Branches

The Jump and the Jump-to-Subroutine instructions are both unconditional program transfers in that they always will transfer program control to their target address. It is possible to conditionally transfer control to the target address based on the state of several internal flags. These flags are bits in the Status Register, and are set or cleared by various instructions, under various circumstances; all of which are described explicitly in the aforementioned G3 Reference Manual. The instructions which conditionally transfer, depending on the current state of these flags, are called Branches, and there are twenty-six different types, each branching under different conditions. A constraint on Branches is that the target address must be near the Branch. It must be no more than -128 (-80 hex) or +127 (+7F hex) words away. This is equivalent to -256 (-100 hex) or +254 (+FE hex) bytes away, but you can only transfer to a word, or even byte, address. Two of the twenty-six are

degenerate cases, one of which always branches, the other of which never branches.

```
BR XYZ
```

will always branch to XYZ. The following one will never branch

```
NOP
```

The assembled value of all Branches has three parts. The left-most four bits will be either a 1000 if it is going to branch if the tested condition is true, or a 1001 if it is going to branch if the tested condition is false. The next four bits select which condition will be tested. The right byte contains the target address information, in the form of an address displacement from the address of the Branch. That displacement, treated as a signed number, is a word displacement (since instruction addresses are always word addresses), and is multiplied by two to turn it into a byte displacement, whereupon it is added to the (in effect, byte) address of the Branch to determine the target address.

If our always-Branch instruction is located at 100, and XYZ has a value of 140, it is assembled as 9020 (hex) and will branch to 140. If the Branch is located at 140, and XYZ has a value of 100, it will be assembled as 90A0 (hex) and will Branch to 100.

If we want to Branch to XYZ, assuming it is within range of the Branch, if the last CMP compared two equal things, we can write:

```
BE XYZ
```

If we want to Branch and they are not equal, we can write:

```
BNE XYZ
```

For more detail and a complete list of all branch instructions see the assembler reference manual in Pluribus Document 5.

2.3 Register-to-Register Instructions

The eight general register operations that may be done in a memory reference instruction may also be done without referencing memory. In this case, rather than one operand being in a memory location, it is in a Register. If we wish to subtract the contents of Register 1 from Register 2, we write:

```
SUB R2,R1
```

It is assembled as 4921 (hex).

2.4 Immediate Operands

Rather than taking the second operand of a general register instruction from memory (see Section 2.1) or from another Register (see Section 2.3), it is possible to take it from part of the instruction itself. This is known as an immediate instruction, or a literal instruction.

If the number is less than sixteen, the instruction is normally written to take up only one word. If we wish to CMP the contents of Register 5 with 7, for instance, it looks like this:

```
CMP R5,=7
```

It is assembled as 4ED7 (hex).

If the number is sixteen or greater, the instruction has to take up two words. If we wish to ADD the number 1234 to the contents of Register 3, we write:

```
ADD R3,#1234
```

It is assembled as 4A38 (hex) in the first word, and 1234 in the second. Note that, as in the case of our first example (in Section 2.1.1), the 1234 might not mean the same thing in each case. Also, the "#" could have been an "="; in that case the assembler would check the operand to see if it would fit in a one-word instruction, and if so, do so; the "#" says put it in the second word no matter how big or small it is.

A variant of the two-word type is one which also uses an index register. In this case the number in the second word is added, no matter what the instruction operation is, to the contents of the index register, and the result is the second

operand of the instruction. If we wish to subtract the number 140 from the contents of Register 2, and Register 6 contains a 40, we write:

```
SUB R2,#100(R6)
```

This is assembled as 492E (hex) in the first word, and 100 in the second.

2.5 Shift Instructions

There are various instructions which cause the word in a register to be moved around. It is possible to shift the word left or right, paying attention to the carry bit or not, losing bits or not, wrapping around or not, etc. See the Pluribus manual for full information. For example, if we wish to shift the contents of Register 2 left six bits, with the carry bit shifted in on the right, we write:

```
RLA R2,6
```

This will be assembled as A1A6 (hex).

If, instead, we want to shift it, ignoring the carry bit and lost data bits, the number of bits that are indicated by the low order four bits of the contents of Register 3, we write:

```
SLL R2,R3
```

This is assembled as A223 (hex).

These two examples are typical of the two main types of shifts. For the others the only difference is the mnemonics.

2.6 Control Instructions

2.6.1 Halt

This instruction halts the processor. It is indicated by a left byte of 0; the right byte may have anything in it, and is essentially ignored. It looks like this:

```
HLT
```

2.6.2 Immediate Operand Control Instructions

This sub-class takes the operand that tells it on what bits to perform its operation from the instruction itself. For example, the right seven bits in instructions which set or reset the Status Register bits (the ones the Branch instructions test) or the right four bits of instructions that enable or disable interrupts. If we wish to enable the computer for interrupts on levels one and three, we write:

ENB 5

This is assembled as 0805 (hex).

2.6.3 Address Operand Control Instructions

This sub-class, which does housekeeping tasks that are useful in servicing interrupts, references at least one memory location in the process. This (first) memory location is determined by the right byte of the instruction, which is interpreted in either of two ways. If the fifth bit from the left in the whole instruction is on, the byte is interpreted as a relative word displacement, exactly the same as in the Branch instructions. If that fifth bit is off, the byte is interpreted as an absolute word address in low memory.

If, for instance, we wish to move the Status Register to the memory word at 1000, and the instruction is located at 1002, we write:

STM 1000

It will be assembled at 09FF (hex).

If we are still located at 1000, but wish to load the Status Register from the word at location 2, we write:

MTS 2

It will be assembled as 0501 (hex).

3.0 Assembler Commands

Section 2 concerns itself with the actual instructions executed by the processing unit in a running Pluribus computer. This section, on the other hand, deals with the assembler and assembly process itself, especially the assembly language conventions and assembler pseudo-operations (or directives).

3.1 Numbers

As mentioned earlier, numbers are interpreted by the assembler as having different values, depending on two things. One is the prevailing radix, and the other is auxiliary information with the number proper.

3.1.1 Radix

Pluribus is initialized to have a radix of 8(octal). You can change to radix n (1<n<37) by a

```
.RADIX n
```

command. In the assembler a number with no auxiliary information is interpreted according to the prevailing radix. Note that with a radix greater than 10, all numbers must begin with 0-9 in order to be noticed as numbers (assuming no auxiliary information). After a

```
.RADIX 20
```

instruction, the radix would be 16 (or 20 octal) and 0ABCD would be a number, whereas ABCD would be a symbol.

3.1.2 Auxiliary Information

In the Pluribus assembler, no matter what the prevailing radix, it is possible to override this assumed radix with auxiliary information that explicitly gives the radix for that number.

The two characters ^O preceding a number explicitly declare it as octal, the two characters ^D declare it to be decimal, and the two characters ^H declare it to be hexadecimal.

For example, the decimal number 254 could be written as:

```
^D254
376
^O376
^HFE
```

assuming, that the assembler's radix is in its initial state.

3.2 Numeric Operations

In the Pluribus assembler, there are various operations that can be done to numbers and their equivalents. They are summarized here; if more information is needed, the Pluribus manual in Document 5 should be consulted.

These are the more common operators and operations:

OPERATOR	OPERATION
+	arithmetic addition
-	arithmetic subtraction
*	arithmetic multiplication
/	arithmetic division
?	logical exclusive OR
!	logical inclusive OR
&	logical AND
<	operator precedence begin
>	operator precedence end

Note that PLURIBUS treats all operators equally and does them from left to right (unless grouped).

3.3 Program Locations

Most programs want to be assembled to reside in specific memory locations, rather than wherever the assembler would put them. Sometimes data tables must be in particular locations so that all programs using them know where they will be. For these and similar reasons, it is possible to direct the assembler to assemble the next statement into a particular place.

This is done by preceding the number or numeric expression with the two characters period-equals.

For example, if we wanted the next word to be assembled at hexadecimal 200, we might say:

```
.=^H200
```

There are also two pseudo-ops available which effect a location change:

```
.ODD
```

will cause the next byte (not word) to be placed in the next odd byte address location if the current byte address is not odd. Similarly, the pseudo-op

```
.EVEN
```

will cause the next word or byte to be assembled into the next even byte address if the current one is not.

3.4 Current Location

It is often handy to be able to conveniently refer to the location of the current word, or perhaps the following word, without knowing the exact location. This "location counter" or "self-reference indicator" is a period. If we, for instance, want to kill some time by subtracting 1 from the number in Register 3 until it is 0 we might write:

```
SUB A3,=1
BNZ .-2
```

If we wish the next word to always have an address that is evenly divisible by 16, we might write:

```
.=.-1&^HFFF0+^H10
```

3.5 General Location

3.5.1 Implicit

For many reasons it is desirable to be able to refer to a location symbolically, without needing to know what the location's address really is. This is done by putting nothing before the symbol on a line, and following it with a colon. This symbol for a location is called a label or a tag.

3.5.2 Explicit

Besides such implicit tag definitions, it is also possible to explicitly define a tag. This is done by putting the symbol to the left of an equals-point. For example:

```
HERE = .
```

3.6 General Assignment

A general assignment statement associates a symbol with a value. The general format for a general assignment statement is:

```
Symbol = Expression
```

This will cause the value of the "Expression" to be assigned to the "Symbol". Examples are:

```
A = 1000
B = 'A&MASKLOW
. = .+7
```

3.7 Data

3.7.1 General Data

If an address should be initialized to a certain constant, the data expression can be written just as any other statement. An explicit pseudo-op, .WORD, is provided which does the same thing. Also, .WORD can have more than one operand:

```
.WORD 1,2,3,4
```

with each separated by commas. This will put 1,2,3,4 in succeeding words. Or the .WORD pseudo-op might be implied:

```
1,2,3,4
```

Intro to Assembly Language

3.7.2 Byte Data

Pluribus can generate byte data with its .BYTE. Operation is exactly like the word pseudo-ops.

3.7.3 Character Data

Arbitrary strings of characters are converted to their ASCII equivalent by the ASCII pseudo-op, which may have any delimiter not in the string itself. The pseudo-ops .ASCIZ will always pad with a zero byte. Strings may be multi-line. Note also that there are two special operators for one- and two-character strings; the apostrophe (for one character) and the quote mark (for two characters). Thus, the following are identical:

```
"XY

.ASCII /XY/
```

as far as the data that they generate.

3.8 Program Termination

The end of the source program is signalled by a .END pseudo-op. An optional argument may be given which is where the program will be started when it is loaded.

3.9 Macros

A discussion of what macros are, and how they are used, is beyond the scope of this document, but let it be said that if you see what looks like an unfamiliar instruction or pseudo-op, it may be a macro "call". To check this, look previously in the program for a macro "definition" which will appear in these forms:

```
.MACRO MNAME
....
.ENDM
```

with the MNAME being the "name" of the newly-defined macro.

3.10 Macro Packages

In the process of programming the Pluribus IMP, the PTIP and related systems, a number of software mechanisms have been developed which are adaptable to more general applications. Many of these tools take the form of macro definitions for the PLURIBUS assembler that provide a convenient syntactic form for some common operation. Larger subroutines and run-time support packages have also been designed which can serve as a foundation for user code. A brief description of each of the packages is given below as an overview.

RATMAC — The RATMAC package defines several higher-level language forms (such as IF/THEN/ELSE and REPEAT/UNTIL) to encourage structured programming in the assembler language environment.

PAGE — The PAGE macros allow the definition of logically distinct program pages and control the assembly on each page. The PAGE macros allow logically connected code and data to be assembled into different memory regions.

STRUCT — The STRUCT macro package allows the definition of data structure formats in a highly readable, easily modified form.

TRANSFER — The transfer macros are defined in a number of systems and provide a conditional control transfer facility which does not have the branch range limitation.

QUTPAT — The QUTPAT macro, together with the patterned quit handler, allows the user to specify the appropriate action in the case of failed bus access operations (QUITs).

3.11 Format

PLURIBUS expects that its source programs will consist of rigidly-defined "statements", one per line. A statement can have up to four "fields", each in order, and each with its specific definitions:

```
label:   operator  operands           ;comments
```

The label is mentioned in Section 3.5; the operator is separated from its operands by at least one space or tab, while the comment field starts with a semicolon. Thus, only relative position is important, with the colon and semicolon removing any ambiguity.

PLURIBUS DOCUMENT 4: BASIC SOFTWARE

PART 3: DDT

DDT

Update History:

Originally written by M.F. Kraley, November 1975.

Revised by R. Hinden, August 1978.

TABLE OF CONTENTS

	page
DDT	1
Addresses, opening and closing	1
Type out modes	2
Other type out commands	3
Type in	4
Address spaces	5
Control	7
Miscellaneous commands	8
Special locations	8
Control structure of DDT	9
Debugging environment of DDT	9
Exceptional conditions	10
User Teletype I/O	10
Entry points	10
Assembly	11

DDT

DDT

DDT is a program which basically provides a mechanism for inspecting and changing registers of the machine. In a broader sense, however, it can be viewed as a simple operating system which controls the starting and stopping of processors and handles extraordinary conditions (QUIT and ILLOP). This manual is not intended as a tutorial; some knowledge of how other DDTs work (see, for example, the PDP-10 DDT manual) may be helpful.

Pluribus DDT comes in many flavors and shapes for different configurations and applications, which will be discussed later. Regardless of the internal structure, all versions appear basically the same to the user. DDT requires a controlling device, like a Teletype or VISTAR, which we will call the TTY. In its simplest form, a single processor runs DDT, either "stand-alone" or in conjunction with a user program. These and other control structures are described below.

We begin by describing the various commands the user may type. A number is represented by "nn", and <altmode> (or <escape>) is represented by "$". A dollar-sign character is indicated by "<dollar>". A caret or uparrow "^" followed by a letter indicates a control character. The character caret (or uparrow) is indicated "<uparr>". The underscore or backarrow character is indicated by "<backarr>". The carriage return character is denoted "<cr>", and linefeed "<lf>". The word "register" generally means a location in address space; a "processor register" is just that. Numbers are followed by "!" to indicate that they are hexadecimal (base 16).

Addresses, opening and closing

Whenever a register is "opened", its contents are typed out in the current mode (except as noted for certain commands). When a register is "closed", the last value typed in while open, if any, will be written to that register. If nothing or <delete> is typed in, nothing is written.

nn/	opens register nn.
<cr>	closes current register, if any open.
<lf>	closes current register, if any open, and opens next "instruction", that is, if type out mode is symbolic (see below) and the

DDT

	current register is a double-word instruction, skip one register.
$<lf>	same as <lf> but always opens the next register, that is, a register is never skipped.
<uparr>	closes current register, if any open, and opens the previous one.
$<uparr>	like <uparr> but goes up two registers, not one.
.	by itself, is the value of the address of the current register, if any open; if none, then the last current register.
/	types out the contents of the register addressed by the current register but does not open it or change ".". The address used is the second word operand address if in symbolic mode or the second word register contents otherwise.
$/	closes the current register and opens the register addressed by the current register, as in "/".

Type out modes

There are two orthogonal type out modes. One controls the radix of type out:

^H	numbers are typed out in hex (base 16) - the default.
^O	numbers are typed out in octal (base 8).

The other controls how register contents are interpreted:

^S	type out symbolically, that is, try to interpret as an instruction, including next word if a two-word instruction code.
^K	(Konstant) type out as a number.

^A	type out as two ASCII characters.

Other type out commands

= retypes out the current register in the alternate mode as follows:

current	alternate
symbolic	constant
constant	symbolic
ASCII	constant

$=	retypes out the current register in the alternate mode, as in "=", and changes the current mode to the alternate mode.
nn=	When preceded by a number or an expression, types out the value of that expression. The result of such expression arithmetic is <u>not</u> considered a value to be written to an open register when closed.
>	retypes out the current register as two ASCII characters, but does not change the mode.
nn"	opens location nn, but does not type out contents; remains in this mode until / or \ is typed.
$"	analogous to $/
"	analogous to /
nn\	opens location nn, but the address type out is suppressed on succeeding lines until / or " is typed.
$\	analogous to $/
\	analogous to /
nn[	opens location nn, but types out contents in the alternate mode (see =, above); does not change current mode.

DDT

$[	analogous to $/
[	analogous to /

<u>Type in</u>

symbols

DDT contains symbols with predefined values to facilitate type in of symbolic data. All of the op codes and other instruction components of the Pluribus assembler are appropriately defined. By using <space> and/or <tab>, instructions may be entered in virtually the same format as the assembler expects. Type in routines correctly interpret displacements in branch instructions. Malformed instructions result in the type out " ", and all current type in is cancelled. There is presently <u>no</u> facility for user defined symbols.

<u>NOTE</u>: The characters <comma>, "=", " ", "+", "-", "(", and ")" have special meaning within an instruction type in, as do the symbols R0, R1, ... R7. Refer to BBN Report No. 3001, Pluribus Document 4, Basic Software, Part 2, for a description of the Pluribus assembler format.

nn

Typed in numbers are generally interpreted according to the current type out radix, except that numbers containing letters A-F are always hexadecimal. Note that some numbers look just like symbols; e.g., ADD, ADDB, BC, BE, BF1, BF2, and BF3. These are treated as <u>symbols</u> unless they are explicitly denoted as <u>numbers</u> by a leading zero or by an "!" after the number. It is a good habit to precede all hexadecimal numbers beginning with the letters A-F by a leading 0.

nn.	a decimal number
nn'	an octal number

nn!	a hex number
<delete>	echoes as "#" and cancels current input, that is, it is as if whatever is being typed in was never typed.
+	addition
<space>	addition
-	subtraction
=	when preceded by an expression, types out the value of that expression. The result of such expression arithmetic is not considered a value to be written to a register when closed.
<backarr>	has the value of the last quantity typed out as a result of examining a register. This would be the value of second word of a two-word instruction when in symbolic mode. If the value of the first word is desired, use "=" followed by <backarr>.
<comma>	is used to input two words at a time as in an instruction. Typing <comma> after the first value saves that value until the terminator is typed after the second value, then both values are written to memory. The value of "." is not changed. A <delete> typed after the <comma> aborts the entire input. If nothing is typed before the <comma>, only the second word will be changed.

Address Spaces

nn:	sets the number of the current processor address space. The processor number is specified according to the Pluribus convention that assigns coupler addresses to indicate the physical processor position in the machine. If set to other than the processor currently running DDT (the "local" processor), all references will be transformed into accesses in that processor's

DDT

address space (on its bus and with the appropriate key bits). If on another bus, BBC will be used; if the other processor on the same bus (the "buddy" processor), the local processor will run in the buddy's registers to examine or change the location. If the buddy is running, he will be stopped, context saved and restored, and restarted. A negative argument to ":" implies the local processor; zero means the buddy. Specifying a non-existent processor (really the control register of the bus coupler from that bus to the ^V I/O bus - see below) causes a "#" type out. Note that when a processor on another bus is selected, attempting to reference a register on the bus being used for BBC (the ^V bus) will result in a QUIT. Use "nn^V" to choose another bus for BBC.

nn^V — sets the base address of the I/O bus to be used for BBC to nn. The default is E000!. A null argument resets to the default.

nn^F — sets the map value of the memory page to be referenced when examining addresses in the first mappable segment (4000! - 6000!) to nn, regardless of current map values. This feature is enabled only when the "local" processor is selected. The procedure used is to transform an access in the 4000! - 6000! window to one in the 8000! - A000! window. The third map (FC04!) is set to the current ^F value and the access performed. The map is then restored per the contents of location F4! in which the user program should keep a copy of the desired map value. An argument of -1 will disable this feature: no address transformation nor map changing will occur. Note that when using a common memory DDT and ^F is disabled, references in the 4000! - 6000! range will access the DDT code page itself, since the code is executed through the first window.

Control

nn^G	starts the selected processor at nn.
^G	starts the selected processor at the address last specified by a ^G command. DDT remembers two independent "last ^G addresses": one for the local processor and one for all others. If no argument is given and no address has been specified previously, a "#" will print and nothing else will happen.
^X	stops the selected processor if running and types out the contents of the program counter. If not running, types out "HALTED". For the local processor, "stops" means stop executing the user program and return control to stand-alone DDT. For processors on other busses, repeatedly tries to stop processor if QUITs occur (usually due to the BBC reference being aborted in favor of a forward access) and reports "FAILED" after 100 unsuccessful tries.
^P	if the local processor is selected, it causes it to "proceed" from where it was last ^Xed (per R0). For other processors, start running without changing the current value of their program counter.
^Z	steps the selected processor one instruction and types that instruction; the local processor cannot be stepped.
$^Z	like ^Z but does not type the instruction.
nn^Z	like ^Z but first sets the program counter to nn.

Miscellaneous commands

Rnn references register nn of the selected processor. For the local processor, the values of registers 0-8 are saved whenever control is returned to DDT and Rnn refers to the memory locations where they are saved. For other processors, refers directly to the registers themselves; the processor must be halted before registers can be examined. The processor number will be echoed.

nn,nn^B copies contents of the local processor's private memory to the corresponding locations of the private memory of the currently selected processor. The two arguments give the inclusive bounds on the addresses to be copied. Omitting arguments will reuse the last value previously specified for that field; if none exists, a "#" will be echoed and no copy takes place.

Special locations

DDT maintains several fixed locations in the private memory of the local processor to communicate with user programs:

F0! F4! Map 0 and Map 2 respectively, are the values to which the maps will be restored when DDT finishes using them. Map 0 is used by common memory DDTs to execute code. Map 2 is used to examine common memory.

F8! the map value of the page in which a common memory DDT resides (should be set up by the user before a common memory DDT is loaded).

FA! DDT option version (see section on Assembly, below).

FC! location of TTY interface.

FE! the number of the currently selected processor.

Control structure of DDT

DDT either runs "stand-alone" or in conjunction with a user program. When stand-alone, DDT is the only program running and thus mostly idles, waiting for a command from the TTY. When DDT starts up a program (e.g., via ^G) it transfers control to that program and expects to have control periodically returned to it so that it can sample the TTY for input, perform any commands requested, and continue any output in progress. This return of control can be accomplished in several different ways:

1. By an explicit call of DDT from the user program. A call to the polling entry (see below) should be executed periodically.

   ```
   JSB R7,ORIGIN+15
   ```

   Control will be returned to the address contained in R7 at the time of the call.

2. By using the line frequency interrupt to call DDT. This requires such interrupts to be enabled for the processor which is to run DDT.

3. By using the TTY interrupt to wake up DDT only when there is actual data flowing to and from the TTY. This requires the TTY to be on the same bus as the processor.

Specification of these options is done when the particular version of DDT is assembled. DDT preserves the contents of registers 1-7, the programmable flags, and enabled interrupt levels across an invocation.

Debugging Environment of DDT

DDT attempts to maintain a "logical" debugging environment similar to the environment the programmer is coding in when he/she is writing the program. In particular, DDT assigns special meanings to such hardware features as the memory map registers (and their effects) and the processor registers. The debugger can thus simulate step-by-step the action of a particular routine by changing the contents of these hardware registers in DDT. DDT, of course, does not change the actual registers, since it is using the registers for its own purposes. Instead, the registers and (in the case of the maps) their side-effects are simulated by DDT.

Exceptional Conditions

As part of its role as "mini-operating system", DDT makes provision for handling QUITs and ILLOPs. DDT assembly parameters specify whether the user or DDT will initially handle these events. DDT assumes (and uses) the standard "expected QUIT" format: a "password" (80FE!) two words after the beginning of the instruction which may cause a QUIT and the address where control should be transferred after a QUIT four words after the instruction. Any "unexpected QUITs" (i.e., those without a password), which DDT's QUIT handler intercepts, will print "QUIT @<program counter>" on the TTY and enter stand-alone mode. Similarly, ILLOPs will print "ILLOP @<program counter>" and also enter stand-alone. DDT does no interpretation of ILLOPs. If the user program fields these events, it can have DDT give these type outs via the appropriate entry points (see below). When DDT gets a QUIT as a result of inspecting or changing registers, it will type out "QUIT". QUITs while executing other DDT functions may result in "FAILED", "RUNNING", or "WHO?" messages, as appropriate.

User Teletype I/O

In normal operation DDT owns the teletype. It is possible for a user program to use DDT to do teletype I/O. The functions are as follows:

- Poll just TTY handler

- In Character Subroutine

- Out Character Subroutine

The user program can call these routines by using DDT Entry points. These are described in the next section.

Entry points

Given here are some of the addresses to which control may be passed to cause various DDT functions. ORIGIN is a parameter set at assembly time to show where the DDT code begins.

ORIGIN	normal restart - DDT enters stand-alone mode.
ORIGIN + 4	ILLOP entry - prints "ILLOP @FOO" and enters stand-alone.

ORIGIN + 8	QUIT entry - prints "QUIT @FOO" and enters stand-alone.
ORIGIN + C!	cold start - clears output buffers and enters stand-alone -- DDT starts here when the paper tape is loaded.
ORIGIN + 10!	polling entry - checks for and performs any work for DDT to do, then returns via R7.
ORIGIN + 14!	poll teletype handler only (if user program wants character I/O).
ORIGIN + 18!	subroutine to get next input character.
ORIGIN + 1C!	subroutine to send next output character.

When DDT is initialized, either by a "cold start" or a "normal restart", all bus coupler control registers on the ^V bus are set to the default state of forward enable and backward disable.

Assembly

DDT has several parameters set at assembly time to specify such things as the addresses of the various pieces of code and variables, the machine environment, the type and location of TTY, method of control transfer, special features (e.g., 1112 processor set), and so forth. There are several collections of these parameters which contain appropriate settings for some configurations of interest. The number of the desired collection is specified at assembly time. More information is given in the source files. The values used for these parameters are printed at the beginning of the DDT listing. The selected option version is stored in location FA! by DDT.

DDT