

**STARAN S
APPLE**
**Programming
Manual**



a new way of thinking

GOODYEAR AEROSPACE CORPORATION

AKRON, OHIO 44315

Price \$25.00

GOODYEAR AEROSPACE CORPORATION

AKRON, OHIO 44315

STARAN S APPLE PROGRAMMING MANUAL

GER-15637

JUNE 1972

NOTICE

This document contains material generated by Goodyear Aerospace Corporation and is transmitted for the purpose of aiding the transaction of business between Goodyear Aerospace Corporation and the recipient. It is understood that the material contained herein will not be used, copied, or disclosed to others, without specific written consent of Goodyear Aerospace Corporation.

APPLE UPDATING

The Associative Processor Programming Language (APPLE) continues to be improved and expanded. Interested parties should contact Goodyear Aerospace Corporation, Computer Division Marketing, Akron, Ohio 44315, Telephone: (216) 794-3631 for information regarding the latest update of APPLE.

LIST OF EFFECTIVE PAGES

Insert latest changed pages and dispose of superseded pages.

NOTE: On a changed page, the portion of the text affected by the latest change is indicated by a vertical line in the outer margin of the page. Changes to illustrations are indicated by miniature pointing hands. A zero in the change number column indicates an original page.

The total number of pages in this manual is 247, consisting of the following:

<u>Page No.</u>	<u>Change Number</u>
Title	0
A	0
i - vi	0
1-1 - 1-3	0
2-i, 2-1 - 2-163.	0
3-1 - 3-29	0
Ai, A1 - A5 . . .	0
Bi, B1	0
Ci, C1 - C7 . . .	0
Di, D1 - D5 . . .	0
Ei, E1 - E4 . . .	0
Fi, F1	0
Gi, G1 - G7 . . .	0
X-1 - X-6	0

NOTE - This document supersedes GER-15532 and GER-15635.

TABLE OF CONTENTS

CHAPTER	TITLE	PAGE
1	INTRODUCTION	1-1
	General	1-1
	APPLE	1-2
	One-To-One	1-2
	One-To-Many	1-2
	In Line	1-2
	Subroutine Call Sequence	1-2
	Assembler Directives	1-2
	Comment Statements	1-2
	APPLE Features	1-2
QUICK INDEX OF APPLE INSTRUCTION GROUPS		2-i
2	APPLE LANGUAGE STRUCTURE	2-1
	Source Statements	2-1
	Label Field	2-1
	Command Field	2-1
	Argument Field	2-1
	Comment Field	2-3
	Required Entries	2-3
	Summary	2-3
	Language Elements	2-4
	Character Set	2-4
	Symbols	2-4
	Symbol Table	2-4
	Constants	2-5
	Octal Constants	2-5
	Decimal Constants	2-5
	Hexadecimal Constants	2-5
	Expressions	2-6
	Examples	2-6
	Location Counters	2-6
	Load Location Counter	2-6
	Execution Location Counter	2-6
	Location Counter Symbol (\$)	2-6
	Addressing	2-7
	Control Memory Address	2-7
	Associative Memory or Common Register Field Expression	2-7
	1	2-8
	2	2-8
	Example 1	2-8
	Example 2	2-8
	Assembler Directives	2-9

TABLE OF CONTENTS

CHAPTER	TITLE	PAGE
2 (cont)	Branch Instructions.....	2-17
	Register Instructions	2-35
	Associative Instructions	2-55
	Loads	2-55
	Stores	2-79
	Searches	2-108
	Moves	2-125
	Arithmetics	2-138
	Control and Test.....	2-154
	Pager Instructions	2-159
3	SUPERVISOR CALLS.....	3-1
	Introduction	3-1
	Slot Numbers	3-1
	Device Assignment Table (DAT).....	3-2
	Instruction Description	3-3
	BUFFER Pseudo-op Format	3-3
	Supervisor Call (SVC) Format	3-3
	Buffer.....	3-3
	Supervisor Call (SVC).....	3-3
	SPS Services or Calls.....	3-4
	Attach.....	3-5
	Format	3-5
	Device Codes	3-5
	STARAN Special Device Codes For ATTACH Function	3-6
	STARAN Control Memory.....	3-6
	Format	3-6
	Buffer Format For Device -1	3-7
	Example.....	3-8
	STARAN Associative Memory.....	3-8
	Format.....	3-8
	Buffer Format For Device -2	3-9
	Example.....	3-11

TABLE OF CONTENTS

CHAPTER	TITLE	PAGE
3 (cont)	STARAN Registers.....	3-12
	Format.....	3-12
	Buffer Format For Device -3	3-13
	Example.....	3-14
	Read	3-15
	Format	3-15
	Example	3-15
	Write	3-16
	Format	3-16
	Example	3-16
	Read/Write BUFFER Pseudo-op	3-17
	Format	3-17
	Example	3-19
	Restart Program	3-21
	Reset Peripheral Devices	3-22
	Free Device For New Task	3-23
	Exit to Supervisor	3-24
	Timer Start	3-25
	Int - Signal Sequential Processor Interrupt	3-26
	Isetup - Setup Interrupt.....	3-27
	Pager Control.....	3-28
	PI/O Control.....	3-29

APPENDIX	TITLE	PAGE
A	SUMMARY OF APPLE MNEMONICS AND INSTRUCTION FORMATS	A-i
B	ERROR CODES	B-i
C	TERMS AND SYMBOLS.....	C-i
D	HEXADECIMAL/DECIMAL TABLE	D-i
E	OCTAL/DECIMAL	E-i
F	POWERS OF TWO TABLE	F-i
G	PROGRAM EXAMPLES.....	G-i
	INDEX	X-1

LIST OF FIGURES

FIGURE	TITLE	PAGE
Frontispiece	STARAN S Computer System.....	vi
2-1	APPLE Assembler Coding Form	2-2
3-1	Device Assignment Table (DAT)	3-2

LIST OF TABLES

TABLE	TITLE	PAGE
2-1	Registers	2-36
2-2	Register Combinations	2-36

FOREWORD

GENERAL

The APPLE Programming Manual is one of five standard manuals for STARAN S. As a composite group, the manuals provide the information necessary for programming, operating, and maintaining the standard STARAN S. The titles and publication numbers of the STARAN S manuals are as follows:

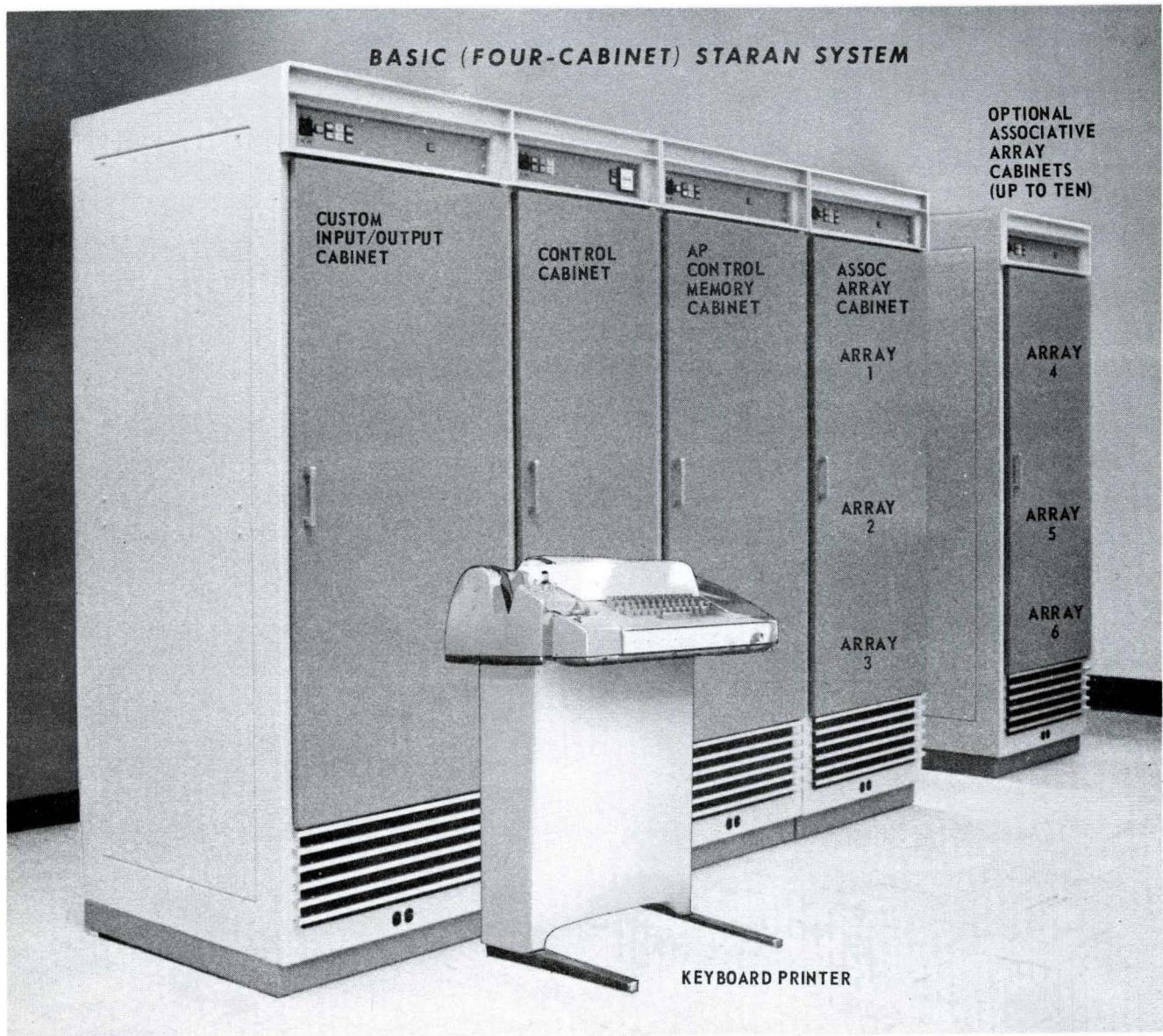
<u>Title</u>	<u>Publication</u>
STARAN S Reference Manual	GER-15636
STARAN S APPLE Programming Manual	GER-15637
STARAN S Operator's Guide	GER-15638
STARAN S Systems Programmer's Reference Manual	GER-15639
STARAN S Maintenance Manual	GER-15640

APPLE MANUAL

The APPLE Programming Manual is intended as a reference manual to guide the programmer in the use of the assembly language. The manual is written for the experienced programmer who has familiarized himself with the STARAN S Reference Manual, GER-15636.

CUSTOM INPUT/ OUTPUT

Since the I/O cabinets are not standard units, but are customized for each particular installation, this manual includes no description of I/O mnemonics included in the APPLE language of a given installation.



STARAN S COMPUTER SYSTEM

CHAPTER 1

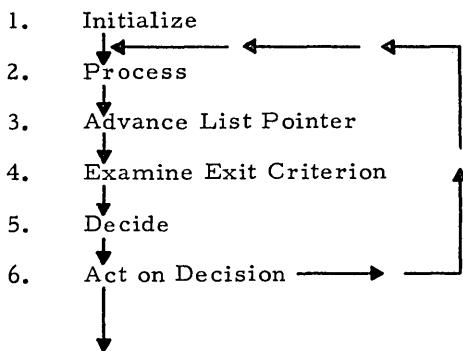
INTRODUCTION

GENERAL

The Goodyear Aerospace Corporation (GAC) Associative Processor, STARAN S*, is a new digital computer system differing significantly from conventional digital computers.

The Associative Processor (AP) is a general-purpose computer capable of performing search, arithmetic, logic, and store operations simultaneously on many independent sets of data. This capability, which is a feature unique to STARAN S, results in certain major differences between programming techniques for STARAN S and those for conventional machines.

As an example, consider the familiar "loop" programming concept. A loop is defined as a set of commands repeatedly and consecutively executed on different sets of data. Conventional programming of a loop involves the following steps:



To process a new set of data conventionally requires execution of the complete loop, including steps 3, 4, 5, and 6, as coding and execution time overhead.

In an AP, execution of the equivalent of a loop on associative items requires initialization and a single pass through the process step. There is no need to advance a list pointer to reference the next set of data to be processed, to determine when to exit from the loop, or to repeatedly execute the process step. The loop is one of many examples of program simplification and improved execution time possible with an AP.

APPLE

Development of a new digital machine organization involves the design of a programming language suitable for the computer.

APPLE is the acronym for the Asociative Processor Programming LanguagE. APPLE is a machine-oriented symbolic language designed to expedite programming for the STARAN S system.

APPLE mnemonics produce four basic types of assembler generated output:

- 1) One-to-One Translation
- 2) One-to-Many Translation
- 3) Assembler Directives
- 4) Comment Statements

ONE-TO-ONE

Most assembler level languages for conventional computers generate one machine language instruction per mnemonic. Many of the basic APPLE mnemonics fall into this category.

ONE-TO-MANY

Several APPLE mnemonics are in the one-to-many category. Many basic AP programming functions require more than one machine language instruction per mnemonic. Some of these mnemonics produce in-line machine instructions; others generate a subroutine call to a sequence of machine instructions.

In Line

The one-to-many mnemonics producing in-line machine instructions are equivalent to macro instructions of higher level assembly languages.

Subroutine Call Sequence

A library of subroutines is provided by APPLE and resides in Page 0 memory. The one-to-many mnemonics produce in-line subroutine call sequences similar to the linkages provided in FORTRAN to the SIN or TAN functions of a FORTRAN library.

ASSEMBLER DIRECTIVES

Assembler directive statements provide functions that assist the programmer in controlling the assignment of storage addresses, defining data and storage fields, and controlling the APPLE system itself. With a few exceptions, assembler directive statements do not generate machine language code.

COMMENT STATEMENTS

Comment statements may appear anywhere in the program and will be printed on the listing device. However, comment statements have no effect on the object code produced.

APPLE
FEATURES

APPLE is essentially a symbolic assembly language. All AP memories and registers may be referenced symbolically.

Constants can be expressed as decimal, octal, or hexadecimal numbers in source statements. Addresses can be expressed absolutely or symbolically.

A listing of the source program statements, the resulting machine language code, and a symbol table may be produced by APPLE for each program. When a source program is assembled, an extensive syntactical check is provided by APPLE. Detected errors are printed on the program listing in error codes (Appendix B) at the left-hand margin of the particular statement in error. A maximum of two error codes can be printed for each statement.

QUICK INDEX

APPLE INSTRUCTION GROUPS

ASSEMBLER DIRECTIVES _____

BRANCH INSTRUCTIONS _____

REGISTER INSTRUCTIONS _____

ASSOCIATIVE INSTRUCTIONS _____

Loads _____

Stores _____

Searches _____

Moves _____

Arithmetics _____

CONTROL and TEST _____

PAGER INSTRUCTIONS _____

SUPERVISOR CALLS _____

CHAPTER 2

APPLE LANGUAGE STRUCTURE

SOURCE STATEMENTS

The source statement is the basic component of an APPLE program. Source statements consist of the following four entries: Label, Command Argument, and Comment. APPLE accepts source statements in free format. Blanks act as field delimiters. The suggested coding form for source statements is shown in figure 2-1. The columns on the coding form correspond to those of a standard 80-column Hollerith coded card. One line of coding on the form corresponds to one source card.

Columns 1 through 72, inclusive, constitute the active line. Columns 73 through 80 are ignored by APPLE except for listing purposes. The source statement may be continued past 72 columns by inserting a semicolon (;), which, when scanned, terminates the present active line. APPLE then searches the next active line to complete the source statement.

LABEL FIELD

The Label Field is usually an optional symbol created by the programmer to identify the statement line. The symbol may consist of nine characters or less, with the first character in column one. If the first column is blank, the Label entry is assumed omitted. The symbol in the Label Field can contain alphabetics (A-Z) or numerics (0-9); however, at least one of the characters must be an alphabetic. The Label Field entry may have the same configuration as predefined mnemonics without conflict, since APPLE distinguishes through context which usage is intended. Only one entry is permitted in the Label Field.

COMMAND FIELD

The Command Field is a requirement. It may consist of several symbols separated by commas (,). The first symbol is the predefined mnemonic (Appendix A) for a particular command. Command modifiers may follow the command, depending upon the individual command. No embedded blanks are allowed in the Command Field.

ARGUMENT FIELD

Entries in the Argument Field properly specify the instruction. In general, the purpose of this field is to identify the source and destination locations to the command. Other entries, such as Control Digits, are also included in this field. The entries are separated by commas and no embedded blanks are allowed. APPLE assumes no Argument Field entries if 16 contiguous blanks follow the Command

L-1751(5-72)

*APPLE ASSEMBLER CODING FORM

PROGRAM _____

PAGE ____ OF ____

PROGRAMMER _____

DATE _____

*THIS IS A SUGGESTED FORM TO FACILITATE READING SOURCE LISTINGS. THE "APPLE"
ASSEMBLER WILL ACCEPT FREE FORM WITH A SPACE TERMINATING EACH FIELD.

LABEL	COMMAND	ARGUMENT	COMMENTS	ID/SEQ
1				
2				
3				
4				
5				
6				
7				
8				
9				
10				
11				
12				
13				
14				
15				
16				
17				
18				
19				
20				

Figure 2-1. APPLE Assembler Coding Form

ARGUMENT FIELD (cont)	Field. Symbols appearing in the Argument Field must be defined to the program, either by being predefined by APPLE or by appearing in the Label Field of a source statement.
COMMENT FIELD	Comments are descriptive items of information that may be included on the program listing. Comment entries consist of any information the programmer wishes to record. All valid characters, including blanks, can be used. The Comment Field begins one blank after the Argument Field, or if no Argument Field exists, comments begin after 16 contiguous blanks follow the Command Field. An asterisk (*) in column one indicates the entire source statement is a comment.
REQUIRED ENTRIES	Required entries for the various mnemonics are underlined in the Format description of each instruction discussion (i. e., <u>B</u> <u>a(r)±k</u> , cd).
SUMMARY	<ol style="list-style-type: none"> 1) APPLE interprets the fields from left to right: Label, Command, Argument, Comment. 2) A blank column terminates any field except the Comment Field, which is terminated at column 80. 3) One or more blanks at the beginning of a line indicates there is no Label Field entry. 4) The Label Field entry, when present, must begin in column 1. 5) The Command Field begins with the first nonblank column following the Label Field or in the first nonblank column following column 1, if the Label Field is omitted. 6) The Argument Field begins with the first nonblank column following the Command Field. An Argument Field is designated as being blank in either of two ways: <ol style="list-style-type: none"> a. Sixteen or more blank columns follow the Command Field. b. The end of the active line (column 72) is encountered and continuation is not indicated. 7) The Comment Field begins in the first nonblank column following the Argument Field, or when the Argument Field is omitted, at least 16 blank columns following the Command Field.

CHARACTER
SET

APPLE language statements are written using the following alphabetics, numerics, operators, and delimiters:

Alphabetics A through Z

Numerics 0 through 9

Operators \$ + - * =

Delimiters , () BLANK ' ;

Each character is represented by an 8-bit byte. Only 47 characters of the set of 256 code combinations defined as the Extended Binary Coded Decimal Interchange Code (EBCDIC) are included in APPLE's character set. Most of the terms used in APPLE source statements are expressed in the character set shown above; however, language features, such as comments, permit the use of any of the 256 EBCDIC codes.

SYMBOLS

Symbols are formed from combinations of characters. Symbols provide programmers with a convenient means of identifying program elements so that they can be referred to by other elements. Symbols must conform to the following rules:

- 1) Symbols consist of 1 to 9 alphanumeric characters.
- 2) At least one character in a symbol must be alphabetic.
- 3) No special characters or embedded blanks can appear in a symbol.
- 4) A symbol may be defined only once. If duplicate symbols occur they will be flagged as errors.

Symbols provide the most commonly used means of addressing source statements, constants, and storage locations. Symbols are normally defined in the Label Field of a source statement. After a symbol has been defined, it can be referred to by Argument Field entries. The value of a symbol can be equated to an absolute value (see EQU, DF in the Assembler Directives discussion)

Symbol
Table

APPLE compiles a table containing all the symbols that appear in the Label Field and the addresses at which they appear. References to symbols cause APPLE to interrogate the symbol table for the address associated with the symbol.

CONSTANTS

A constant is a self-defining language element whose value is explicit. Self-defining terms are useful in constants requiring a value rather than the symbolic address of the location where that value is stored. Three constant notations are used in APPLE instructions: octal, decimal, and hexadecimal.

Octal Constants

An octal constant consists of a signed octal number enclosed by single quotation marks and preceded by the letter O.

The constant is right-justified in its field. For example,

Constant	Binary Value				Hexadecimal Value			
O' 1234'	001	010	011	100	0010	1001	1100	(29C)

The octal digits and their binary equivalents are as follows:

0 - 000	4 - 100
1 - 001	5 - 101
2 - 010	6 - 110
3 - 011	7 - 111

Decimal Constants

A decimal constant consists of an integer (no decimal point) that may be signed. For example, 100 or -5423.

Hexadecimal Constants

A hexadecimal constant consists of a signed hexadecimal number enclosed by single quotation marks and preceded by the letter X.

For example,

X'9C01F' X'C0FFEE' X'FFFF'

The assembler generates four binary bits of storage for each hexadecimal digit. The hexadecimal digits and their binary equivalents are as follows:

0 - 0000	8 - 1000
1 - 0001	9 - 1001
2 - 0010	A - 1010
3 - 0011	B - 1011
4 - 0100	C - 1100
5 - 0101	D - 1101
6 - 0110	E - 1110
7 - 0111	F - 1111

EXPRESSIONS Argument Field entries consist of either single-term expressions or double-term expressions. Single-term expressions are symbols, constants, or Location Counter references (\$). Double-term expressions are two single terms connected with an arithmetic operator. The valid arithmetic operators are a plus sign (+) for addition and a minus sign (-) for subtraction. The first single-term expression of a double-term expression may be a symbol or constant, and the second single-term expression must be a constant.

Examples	<u>Valid</u>	<u>Invalid</u>
	TAG+5	TAG-LABEL
	LABEL-23	5+TAG
	5+32	TAG+5+23

LOCATION COUNTERS APPLE maintains two internal Location Counters: a Load Location Counter and an Execution Location Counter. The Load Location Counter keeps track of the addresses associated with the instructions when the program is loaded. The Execution Counter keeps track of the addresses associated with the instructions when they are executed.

Load Location Counter The Load Location Counter keeps track of the addresses associated with the instructions when they are loaded.

As each instruction or data area is assembled, the Load Location Counter is incremented by the length of the assembled item. Therefore, the Load Location Counter is the address of the next available storage location in Control Memory after the instruction is assembled. This address is the location where the instruction will reside after being loaded.

Execution Location Counter As each instruction or data area is assembled, the Execution Location Counter is incremented by the length of the assembled item. The Execution Location Counter differs from the Load Location Counter when Pager commands are encountered. (See Pager Instructions.) Each STRTSG that appears in an assembly reinitializes the Execution Location Counter. This address is the location where the instruction will reside when executed.

Location Counter Symbol (\$) The special symbol, \$ (dollar sign), is predefined by APPLE as Location Counters. The \$ may be used to alter the Location Counters at assembly time (see ORG in Assembler Directives Discussion). The \$ may also be used in an absolute expression to refer to an address. In this context it is the Execution Location Counter that forms the address.

ADDRESSING

Control Memory Address

The Control Memory Address is a symbolic or absolute address in bulk core, page memory, or High Speed Data Buffer. A Control Memory Address expression is comprised of four terms in the form $a(r)\pm k, cd$. Note that required terms are underlined.

a - This entry is the only one required. This term may be a symbol or a constant.

k - This entry must be a constant. At assembly time $\pm k$ is added to the value of 'a' to form the address.

r - This entry must be one of the following registers:
R0 through R7, DP.

At execution time the contents of this specified register is added to the value $a\pm k$. It is this result that defines the Control Memory Address. The contents of the register can be considered to be the base address, and the double-term expression $a\pm k$ can be considered to be the displacement.

cd - This entry is the Control Digit. A Control Digit indicates that after the specified instruction is completed a step is desired. This step may increment or decrement the data pointer (DP) register by one and/or decrement the block length (BL) register by one. The Control Digit may be specified by an $a\pm k$ type of expression, where 'a' and 'k' are defined as above.

<u>cd Values</u>	<u>Action</u>
1	Decrement BL
2	Increment DP
3	Decrement BL and Increment DP
4	Decrement DP
5	Decrement BL and DP

The Control Digit is a valid entry only when the base register option has been selected, and the register forming the base register is the DP register.

Associative Memory or Common Register Field Expression

A field expression defines the most significant bit position and the number of contiguous bit positions (field length) occupied by a field. There are two ways of constructing a field expression:

Associative
Memory or
Common Register
Field Expression
(cont)

- 1

b±i

where b must be a symbol, and i is an optional constant modifier. b should have been previously defined in a DF instruction. b represents the most significant bit position and the number of contiguous bits occupied by a field in either the Common register or Associative Memory. The optional constant modifier, i, modifies only the most significant bit position.

- 2

(b, i)±j

where b may be a constant or a symbol and represents the most significant bit position of a field. If b was defined as a field via a previous DF instruction, the most significant bit position is the value used. i must be a constant and represents the number of contiguous bits occupied by the field. j is an optional constant modifying only the most significant bit position of the field.

- Example 1

AJAX	DF	10, 3
	.	
	.	
	.	
SC		AJAX-3,(100, 3)

The field AJAX begins in bit column 10 and spans 3 bit columns (bit columns 10, 11, 12).

The expression AJAX-3 has modified the most significant bit position to a value of 7 and spans 3 bit columns (bit columns 7, 8, 9).

The expression (100, 3) defines a field beginning with bit column 100 and spans 3 bit columns (bit columns 100, 101, 102).

- Example 2

FIELD1	DF	0, 5
	.	
	.	
	.	
SC		(FIELD1, 17)+O'17',(X'80', O'21')

The field FIELD1 begins in bit column 0 and spans 5 bit columns (bit columns 0, 1, 2, 3, 4).

The expression (FIELD1, 17)+O'17+ has modified the most significant bit position to a value of 15, and has also modified the number of bit columns to 17 (bit columns 15, 16, ..., 31).

The expression (X'80', O'21') defines a field beginning with bit column 128 and spans 17 bit columns (bit columns 128, 129, ..., 144).

ASSEMBLER DIRECTIVES

Assembler directive statements provide auxiliary functions to APPLE and assist the programmer in checking, documenting, and organizing a program.

The assembler directives are:

<u>Mnemonic</u>	<u>Instruction</u>
START	Start APPLE
END	End APPLE
ORG	Initialize Location Counter
EQU	Equate
DF	Define a Field
DS	Define Storage
TOF	Top of Form
EVEN	Make Location Counter Even
DC	Define Constant
GEN	Generate Machine Instructions
NOP	No Operation
A or E	Character String Generator

START

Start APPLE

This instruction performs initializing functions for APPLE, and generates pertinent header information for all object programs. This instruction is required and should be the first source statement in all APPLE programs.

Format	Label	Command	Argument	Comment
	symbol	<u>START</u>		

- Label Any valid symbol or blank.

- Command START

- Argument No entry required.

END

End APPLE

This instruction will process and assemble all previous source program statements. The END instruction is required and must be the last source statement of every assembly.

Format	Label	Command	Argument	Comment
	symbol	END	a±k	

- Label Any valid symbol or blank.

- Command END

- Argument An optional entry.

- a±k

'a' may be either a symbol or a constant whose value may be optionally modified by plus or minus the constant k. This term represents an address designating where program execution will start immediately after the object program is loaded.

- blank

If no address is specified in the argument field, this program will not automatically begin execution upon completion of loading. In this case the END - statement signals to the assembler the end of the current program.

ORG

Initialize Location Counter

This instruction commands the assembler to assemble succeeding instructions beginning at the address specified in the Argument Field. The Load Location Counter and Execution Location Counter are loaded with the value of $a \pm k$.

Format

Label	Command	Argument	Comment
symbol	<u>ORG</u>	<u>$a \pm k$</u>	

- Label Any valid symbol or blank.
- Command ORG
- Argument One entry is required.
- • $a \pm k$ 'a' may be either a symbol or a constant whose value may be optionally modified by plus or minus the constant k. Moreover 'a' may be one of the following special predefined symbols provided for ease of programming:

<u>a</u>	<u>Definition</u>
PAGE0	Page 0 Memory Starting Address
PAGE1	Page 1 Memory Starting Address
PAGE2	Page 2 Memory Starting Address
HSDB	High-Speed Data Buffer Memory Starting Address
DMA	Direct Memory Access Memory Starting Address
BULKC	Bulk Core Storage Memory Starting Address

Example

ORG BULKC+16

- Note In this example the first instruction following the ORG statement will be assigned the Bulk core address X'8020' (BULKC assigns the address X'8010' in the APPLE assembler).

EQU

Equate

This instruction permits the programmer to assign a value to a symbol. Whenever the symbol appears in a succeeding instruction, the equated value will be used to form the machine language code.

Format

Label	Command	Argument	Comment
<u>symbol</u>	<u>EQU</u>	<u>a±k</u>	

• Label

Any valid symbol. This entry is required.

• Command

EQU

• • a±k

'a' may be either a symbol or a constant whose value may be optionally modified by plus or minus the constant k. 'a' may also be one of the special predefined APPLE symbols such as register abbreviations (Table 2-1) PAGE0, PAGE1, PAGE2, HSDB, DMA, BULKC, X, Y, and M. However, if a special symbol is used it cannot be modified by k.

DF

Define a Field

This instruction permits the programmer to assign a field definition value to a symbol for later use. Whenever the symbol appears in instructions, the defined field value will be used to form the machine language code.

Format

Label	Command	Argument	Comment
<u>symbol</u>	<u>DF</u>	<u>a₁±k₁, a₂±k₂</u>	

• Label

Any valid symbol. This entry is required.

• Command

DF

• Argument

Two entries are required.

• • a₁±k₁, a₂±k₂

'a' may be either a symbol or a constant whose value may be optionally modified by plus or minus the constant k. The value of the term a₁±k₁ represents the most significant bit position of the field being defined. The value of the term a₂±k₂ represents the number of contiguous bit positions (field length) occupied by the field being defined.

Note

The sum of a₁±k₁ or a₂±k₂ must not exceed the total number of bits in an associative memory word (0 to 255). If the field being defined is a field in the Common register, the sum of a₁±k₁ or a₂±k₂ should not exceed the number of bits in the Common register (0 to 31).

DS

Define Storage

This assembler directive will allocate the next specified number of 32 bit words as a contiguous block of control memory.

Format

Label	Command	Argument	Comment
symbol	<u>DS,a±k</u>		

- Label

Any valid symbol or blank.

- Command

DS

- a±k

'a' may be either a symbol or a constant whose value may be optionally modified by plus or minus the constant k. The value of the term a±k specifies the number of contiguous words to be reserved. If this entry is omitted, a default value of one is assumed.

- Argument

Blank

TOF

Top of Form

This assembler directive will issue a form feed to the assembly listing device. TOF may be placed anywhere in the program and has no effect on the object code produced.

Format

Label	Command	Argument	Comment
	<u>TOF</u>		

- Label

Must be blank.

- Command

TOF

- Argument

None required.

- Comment

The comment will be printed at the top of the page after the form feed.

EVEN

Make Location Counter Even

If the Execution Location Counter is odd when this instruction is encountered, an NOP will be produced in the object code; otherwise, no object code will be produced. Therefore, after this instruction has been processed the Execution Location Counter will be even. (Ref. SPSW instruction.)

Format	Label	Command	Argument	Comment
	symbol	<u>EVEN</u>		

- Label Any valid symbol or blank.
- Command EVEN
- Argument None required.

DC

Define Constant

This instruction will generate a specified value for a specified number of 32 bit control memory words.

Format	Label	Command	Argument	Comment
	symbol	<u>DC,a₁±k₁</u>	<u>a₂±k₂</u>	

- Label Any valid symbol or blank.
- Command DC
- • a₁±k₁ a₁ may be either a symbol or a constant whose value may be optionally modified by plus or minus the constant k₁. The value of the term a₁±k₁ specifies the number of contiguous 32-bit words. If this entry is omitted, a value of one is assumed.
- Argument a₂ may be either a symbol or a constant whose value may be optionally modified by plus or minus the constant k₂. The value of the term a₂±k₂ is the value to be inserted in each of the 32-bit words.

GEN

Generate Machine Instructions

This instruction permits the programmer to generate machine codes for instructions not covered by APPLE. (See STARAN S Reference Manual for detailed machine language coding.) This instruction is also useful when generating words of data rather than instructions.

Format	Label	Command	Argument	Comment
	symbol	<u>GEN, k₁, ..., k_n</u>	<u>a₁^{±j₁}</u> , ..., a _n ^{±j_n}	
• Label	Any valid symbol or blank.			
• Command	GEN			
• • k _i	One or more constants that define the length of the consecutive data fields a _i ^{±j_i, respectively. The sum of all the k's must be less than or equal to 32.}			
• Argument				
• • a _i ^{±j_i}	a _i may be either a symbol or a constant whose value may be optionally modified by plus or minus the constant j _i . These term(s) represent the value(s) to be inserted into each of the corresponding data field(s). There must be a one-to-one correspondence between the k _i and a _i ^{±j_i terms.}			
Note	If the sum of the lengths of the data fields is less than 32, the information will be right-justified in the word.			

NOP

No Operation

This instruction performs no operation when it is executed.

Format	Label	Command	Argument	Comment
	symbol	<u>NOP</u>		
• Label	Any valid symbol or blank.			
• Command	NOP			
• Argument	No entry required.			

A or E

Character String Generator

These two assembler directives enable the programmer to generate messages for output.

Format

Label	Command	Argument	Comment
symbol	<u>A</u> $c_1 c_2 \dots c_{i-1} c_i$ <u>x</u>		

or

Label	Command	Argument	Comment
symbol	<u>E</u> $c_1 c_2 \dots c_{i-1} c_i$ <u>x</u>		

• Label

Any valid symbol or blank.

• Command

A character string entry is required.

• • A

'A' represents an assembler directive commanding the assembler to generate the seven bit ASCII code equivalent to the succeeding character string.

• • E

E represents an assembler directive commanding the assembler to generate the eight bit EBCDIC code equivalent to the succeeding character string.

• • x

x must be any non-alphanumeric character and serves as the "begin" and "end" marker of the character string $c_1 c_2 \dots c_{i-1} c_i$. x cannot be a ';'.

• • $c_1 c_2 \dots c_{i-1} c_i$

The c_i may be any allowable ASCII or EBCDIC character (except the ';') depending on whether A or E is used respectively. One or more full thirty-two bit words are generated with the ASCII or EBCDIC code of the c_i packed on byte boundaries at four characters per word. If there are not enough characters to generate a full word, the remaining bytes will be padded with blank (or space) characters.

• Argument

No entry is required.

Note

There is no provision for continuation of a character string onto several source cards.

A ';' character can be used in the text of the character string if there are enough blank delimiters preceding it so that it would fall in the comment field when parsed according to the free format rules of an ordinary source statement.

**BRANCH
INSTRUCTIONS**

Branch instructions alter the execution sequence of a program if certain conditions exist.

The branch instructions are:

<u>Mnemonic</u>	<u>Instruction</u>
B	Unconditional Branch
BZ	Branch if Zero
BNZ	Branch if Not Zero
BBS	Branch if Bit Set
BBZ	Branch if Bit Zero
BRS	Branch if Response
BNR	Branch if No Response
BOV	Branch if Overflow
BNOV	Branch if No Overflow
BAL	Branch and Link
RPT	Repeat
LOOP	Loop

B

Unconditional Branch

This instruction will transfer control from the current program address to the address specified in the Argument Field.

Format

Label	Command	Argument	Comment
symbol	<u>B</u>	<u>a(r)±k, cd</u>	

- Label Any valid symbol or blank

- Command B

- Argument The Control Memory Address is a symbolic or absolute address in Bulk Core, Page Memory, or High-Speed Data Buffer. The Control Memory Address may be represented by four terms in the form $a(r)±k, cd$.

- • a This entry is required only if the optional term (r) is omitted. This term may be either a symbol or a constant.

- • k This optional term must be a constant and modifies 'a'.

- • r This entry may be one of the following nine registers: R0 through R7, DP. The contents of this specified register is added to the value $a±k$ at execution time. It is this result that defines the Control Memory Address. The contents of the register can be considered the base address, and the $a±k$ expression can be considered the displacement.

- • cd This entry is the Control Digit. A Control Digit indicates that after the specified instruction is completed a step is desired. This step may increment or decrement the data pointer (DP) register by one and/or decrement the block length (BL) register by one. The Control Digit may be specified by an $a±k$ type of expression where 'a' and k are as defined above.

<u>cd Values</u>	<u>Action</u>
------------------	---------------

1	Decrement BL
2	Increment DP
3	Decrement BL and increment DP
4	Decrement DP
5	Decrement BL and DP

The Control Digit is a valid entry only when the base register option has been selected, and the register forming the base register is the DP register.

BZ

Branch if Zero

This instruction will transfer control from the current program address to the address specified in the argument field, if the command field register, r_1 , is zero.

Format	Label	Command	Argument	Comment
	symbol	BZ, r_1	$a(r_2)\pm k, cd$	

- Label Any valid symbol or blank.

- Command BZ

<u>Register</u>	<u>Definition</u>
FP1	Field Pointer 1 (8 bits)
FP2	Field Pointer 2 (8 bits)
FP3	Field Pointer 3 (8 bits)
FL1	Field Length Counter 1 (8 bits)
FL2	Field Length Counter 2 (8 bits)
FPE	Field Pointer E (8 bits)
BL	Block Length Counter (16 bits)
DP	Data Pointer Register (16 bits)

- Argument The Control Memory Address is a symbolic or absolute address in Bulk Core, Page Memory, or High-Speed Data Buffer. The Control Memory Address may be represented by four terms in the form $a(r_2)\pm k, cd$.

- • a This entry is required only if the optional term (r_2) is omitted. This term may be either a symbol or a constant.

- • k This optional term must be a constant and modifies 'a'.

- • r_2 This entry must be one of the following nine registers: R0 through R7, DP. The contents of the specified register is added to the value $a\pm k$ at execution time. The result defines the Control Memory Address. The contents of the register can be considered the base address, and the $\pm k$ expression can be considered the displacement.

BZ

• • cd

This entry is the Control Digit. A Control Digit indicates that after the specified instruction is completed a step is desired. This step may increment or decrement the data pointer (DP) register by one and/or decrement the block length (BL) register by one. The Control Digit may be specified by an $a \pm k$ type of expression, where 'a' and k are as defined above.

<u>cd Values</u>	<u>Action</u>
1	Decrement BL
2	Increment DP
3	Decrement BL and increment DP
4	Decrement DP
5	Decrement BL and DP

The Control Digit is a valid entry only when the base register option has been selected, and the register forming the base register is the DP register.

BNZ

Branch if Not Zero

This instruction will transfer control from the current program address to the address specified in the argument field, if the command field register, r_1 , is not zero.

Format	Label	Command	Argument	Comment
	symbol	<u>BNZ, r₁</u>	<u>a(r₂)±k, cd</u>	

- Label Any valid symbol or blank.

- Command BNZ

- • r_1

Register	Definition
FP1	Field Pointer 1 (8 bits)
FP2	Field Pointer 2 (8 bits)
FP3	Field Pointer 3 (8 bits)
FL1	Field Length Counter 1 (8 bits)
FL2	Field Length Counter 2 (8 bits)
FPE	Field Pointer E (8 bits)
BL	Block Length Counter (16 bits)
DP	Data Pointer Register (16 bits)

- Argument The Control Memory Address is a symbolic or absolute address in Bulk Core, Page Memory, or High-Speed Data Buffer. The Control Memory Address may be represented by four terms in the form $a(r_2)±k, cd$.

- • a This entry is required only if the optional term (r_2) is omitted. This term may be either a symbol or a constant.

- • k This optional term must be a constant and modifies 'a'.

- • r_2 This entry may be one of the following nine registers: R0 through R7, DP. The contents of the specified register is added to the value $a±k$ at execution time. It is this result that defines the Control Memory Address. The contents of the register can be considered the base address, and the $a±k$ expression can be considered the displacement.

- • cd This entry is the Control Digit. A Control Digit indicates that after the specified instruction is completed a step is desired. This step may increment or decrement the data pointer (DP) register by one and/or decrement the block length (BL) register by one. The Control Digit may be specified by an $a±k$ type of expression, where 'a' and k are defined as above.

• • cd (cont)	<u>cd Values</u>	<u>Action</u>
1		Decrement BL
2		Increment DP
3		Decrement BL and increment DP
4		Decrement DP
5		Decrement BL and DP

The Control Digit is a valid entry only when the base register option has been selected, and the register forming the base register is the DP register.

The execution of the branch in this instruction is contingent on the status of a selected bit in the Common register. Prior to execution of this instruction, an instruction must be executed to load the FP1 register with the address of the bit in the Common register to be tested for the contingency. If the selected Common register bit is one, this instruction will transfer control from the current program address to the address specified in the Argument Field.

Format	Label	Command	Argument	Comment
	symbol	<u>BBS</u>	a(r) \pm k, cd	

- Label Any valid symbol or blank.
 - Command BBS
 - Argument The Control Memory Address is a symbolic or absolute address in Bulk Core, Page Memory, or High-Speed Data Buffer. The Control Memory Address may be represented by four terms in the form $a(r)\pm k, cd$.
 - • a This entry is only required if the optional term (r) is omitted. This term may be either a symbol or a constant.
 - • k This optional term must be a constant and modifies 'a'.
 - • r This entry may be one of the following nine registers: R0 through R7, DP. The contents of the specified register is added to the value $a\pm k$ execution time. The result defines the Control Memory Address. The contents of the register can be considered the base address, and the $a\pm k$ expression can be considered the displacement.
 - • cd This entry is the Control Digit. A Control Digit indicates that after the specified instruction is completed a step is desired. This step may increment or decrement the data pointer (DP) register by one and/or decrement the block length (BL) register by one. The Control Digit may be specified by an $a\pm k$ type of expression, where 'a' and k are defined as above.

<u>cd Values</u>	<u>Action</u>
1	Decrement BL
2	Increment DP
3	Decrement BL and increment DP
4	Decrement DP
5	Decrement BL and DP

The Control Digit is a valid entry only when the base register option has been selected, and the register forming the base register is the DP register.

BBS

Example

	LI	FP1, 24	CHECK BIT 24
	BBS	TAG	BRANCH IF BIT IS ONE
		.	
		.	
		.	
	TAG	DECR FP1	CHECK BIT 23
		BBS CONT	BRANCH IF BIT IS ONE
		WAIT	
		.	
		.	
		.	
	CONT	LRR C, FP2	

• Common
Register
Contents



The first branch (BBS TAG) will take place, since FP1 is loaded with 24 and bit 24 in the Common register is one.

The second branch (BBS CONT) will not take place, since after the DECR, FP1 contains 23 and bit 23 in the Common register is zero. Thus the next instruction executed will be the WAIT.

BBZ

Branch if Bit Zero

The execution of the branch in this instruction is contingent on the status of a selected bit in the Common register. Prior to execution of this instruction, an instruction must be executed to load the FPL register with the address of the bit in the Common register to be tested for the contingency. If the selected Common register bit is zero, this instruction will transfer control from the current program address to the address specified in the argument field.

Format	Label	Command	Argument	Comment
	symbol	<u>BBZ</u>	<u>a(r)±k, cd</u>	

- Label Any valid symbol or blank.
- Command BBZ
- Argument The Control Memory Address is a symbolic or absolute address in Bulk Core, Page Memory, or High Speed Data Buffer. The Control Memory Address may be represented by four terms in the form $a(r)\pm k, cd$.
- • a This entry is required only if the optional term (r) is omitted. This term may be either a symbol or a constant.
- • k This optional term must be a constant and modifies 'a'.
- • r This entry may be one of the following nine registers: R0 through R7, DP. The contents of the specified register is added to the value $a\pm k$ at execution time. The result defines the Control Memory Address. The contents of the register can be considered the base address, and the $a\pm k$ expression can be considered the displacement.
- • cd This entry is the Control Digit. A Control Digit indicates that after the specified instruction is completed a step is desired. This step may increment or decrement the data pointer (DP) register by one and/or decrement the block length (BL) register by one. The Control Digit may be specified by an $a\pm k$ type of expression, where 'a' and k are defined as above.

<u>cd Values</u>	<u>Action</u>
1	Decrement BL
2	Increment DP
3	Decrement BL and increment DP
4	Decrement DP
5	Decrement BL and DP

The Control Digit is a valid entry only when the base register option has been selected, and the register forming the base register is the DP register.

BBZ

Example

	LI	FP1, 10	CHECK BIT 10	
	BBZ	TAG	BRANCH IF ZERO	
	.			
	.			
	.			
	TAG	INCR	FP1	CHECK BIT 11
		BBZ	CONT	BRANCH IF BIT IS ZERO
		WAIT		
	.			
	.			
	.			
	CONT	DECR	FP2	

- Common Register Contents



The first branch (BBZ TAG) will take place, since FP1 is loaded with the number 10 and bit 10 in the Common register is zero. The second branch (BBZ CONT) will not take place, since after the INCR, FP1 contains 11 and bit 11 in the Common register is one. Thus the next instruction executed will be the WAIT.

Branch if Response

This instruction will check the Y response store register. If any Y response store register bit position is set to one in any enabled array, the branch will be executed.

Format	Label	Command	Argument	Comment
	symbol	<u>BRS</u>	<u>a(r)±k, cd</u>	

- Label Any valid symbol or blank
 - Command BRS
 - Argument The Control Memory Address is a symbolic or absolute address in Bulk Core, Page Memory, or High-Speed Data Buffer. The Control Memory Address may be represented by four terms in the form $a(r)\pm k$, cd .
 - • a This entry is required only if the optional term (r) is omitted. This term may be either a symbol or a constant.
 - • k This optional term must be a constant and modifies ' a '.
 - • r This entry may be one of the following nine registers: R0 through R7, DP. The contents of the specified register is added to the value $\pm k$ at execution time. The result defines the Control Memory Address. The contents of the register can be considered the base address, and the $\pm k$ expression can be considered the displacement.
 - • cd This entry is the Control Digit. A Control Digit indicates that after the specified instruction is completed a step is desired. This step may increment or decrement the data pointer (DP) register by one and/or decrement the block length (BL) register by one. The Control Digit may be specified by an $\pm k$ type of expression, where ' a ' and k are as defined above.

<u>cd Values</u>	<u>Action</u>
1	Decrement BL
2	Increment DP
3	Decrement BL and increment DP
4	Decrement DP
5	Decrement BL and DP

The Control Digit is a valid entry only when the base register option has been selected, and the register forming the base register is the DP register.

BNR

Branch if No Response

This instruction will check the Y response store register. If all Y response store register bit positions are equal to zero in all enabled arrays, the branch will be executed.

Format	Label	Command	Argument	Comment
	symbol	<u>BNR</u>	<u>a(r)±k, cd</u>	

- Label Any valid symbol or blank.
- Command BNR
- Argument The Control Memory Address is a symbolic or absolute address in Bulk Core, Page Memory, or High Speed Data Buffer. The Control Memory Address may be represented by four terms in the form $a(r)±k, cd$.
- • a This entry is required only if the optional term (r) is omitted. This term may be either a symbol or a constant.
- • k This optional term must be a constant and modifies 'a'.
- • r This entry may be one of the following nine registers: R0 through R7, DP. The contents of this specified register is added to the value $a±k$ at execution time. The result defines the Control Memory Address. The contents of the register can be considered the base address, and the $a±k$ expression can be considered the displacement.
- • cd This entry is the Control Digit. A Control Digit indicates that after the specified instruction is completed a step is desired. This step may increment or decrement the data pointer (DP) register by one and/or decrement the block length (BL) register by one. The Control Digit may be specified by an $a±k$ type of expression, where 'a' and k are defined as above.

<u>cd Values</u>	<u>Action</u>
1	Decrement BL
2	Increment DP
3	Decrement BL and increment DP
4	Decrement DP
5	Decrement BL and DP

The Control Digit is a valid entry only when the base register option has been selected, and the register forming the base register is the DP register.

BOV

Branch if Overflow

This instruction allows the programmer to test for an overflow or underflow condition after an arithmetic operation. This instruction will perform X exclusive OR Y ANDed with M and store the result in Y. Then if any Y response store bit in any enabled array equals one an overflow condition exists for that word. If such is the case, this instruction will branch to the address specified in the Argument Field. The X response store will equal one for the corresponding word of associative memory if an underflow occurred; otherwise an overflow occurred.

Format	Label	Command	Argument	Comment
	symbol	<u>BOV</u>	<u>a(r)±k, cd</u>	

- Label Any valid symbol or blank.
 - Command BOV
 - Argument The Control Memory Address is a symbolic or absolute address in Bulk Core, Page Memory, or High Speed Data Buffer. The Control Memory Address may be represented by four terms in the form $a(r)\pm k, cd$.
 - • a This entry is required only if the optional term (r) is omitted. This term may be either a symbol or a constant.
 - • k This optional term must be a constant and modifies 'a'.
 - • r This entry may be one of the following nine registers: R0 through R7, DP. The contents of this specified register is added to the value $\pm k$ at execution time. The result defines the Control Memory Address. The contents of the register can be considered the base address, and the $\pm k$ expression can be considered the displacement.
 - • cd This entry is the Control Digit. A Control Digit indicates that after the specified instruction is completed a step is desired. This step may increment or decrement the data pointer (DP) register by one and/or decrement the block length (BL) register by one. The Control Digit may be specified by an $\pm k$ type of expression, where 'a' and k are defined as above.

<u>cd Values</u>	<u>Action</u>
1	Decrement BL
2	Increment DP
3	Decrement BL and increment DP
4	Decrement DP
5	Decrement BL and DP

The Control Digit is a valid entry only when the base register option has been selected, and the register forming the base register is the DP register.

BNOV

Branch if No Overflow

This instruction allows the programmer to test for the absence of an overflow or underflow condition following an arithmetic instruction.

This instruction will perform X exclusive OR Y ANDed with M and store the result in Y. If all Y response store bits of all enabled arrays equal zero, i.e., no overflow condition exists, this instruction will branch to the address specified in the Argument Field. If the branch does not take place, the X response store will equal one for the corresponding word of associative memory if an underflow occurred; otherwise, an overflow occurred.

Format	Label	Command	Argument	Comment
	symbol	<u>BNOV</u>	<u>a(r)±k, cd</u>	

- Label

Any valid symbol or blank.

- Command

BNOV

- Argument

The Control Memory Address is a symbolic or absolute address in Bulk Core, Page Memory, or High-Speed Data Buffer. The Control Memory Address may be represented by four terms in the form $a(r)±k, cd$.

- • a

This entry is required only if the optional term (r) is omitted. This term may be either a symbol or a constant.

- • k

This optional term must be a constant and modifies 'a'.

- • r

This entry may be one of the following nine registers: R0 through R7, DP. The contents of this specified register is added to the value $a±k$ at execution time. The result defines the Control Memory Address. The contents of the register can be considered the base address, and the $a±k$ expression can be considered the displacement.

- • cd

This entry is the Control Digit. A Control Digit indicates that after the specified instruction is completed a step is desired. This step may increment or decrement the data pointer (DP) register by one and/or decrement the block length (BL) register by one. The Control Digit may be specified by an $a±k$ type of expression, where 'a' and k are defined as above.

cd Values	Action
1	Decrement BL
2	Increment DP
3	Decrement BL and increment DP
4	Decrement DP
5	Decrement BL and DP

The Control Digit is a valid entry only when the base register option has been selected, and the register forming the base register is the DP register.

BAL

Branch and Link

This instruction will transfer control to a subroutine after storing the Execution Location Counter of the next instruction in the branch and link register r_1 .

Format	Label	Command	Argument	Comment
	symbol	<u>BAL, r</u> ₁	<u>a(r</u> ₂ <u>)</u> ^{±k} , cd	

- Label Any valid symbol or blank.
 - Command BAL
 - • r_1 One of the branch and link registers R0 through R7.
 - Argument The Control Memory Address is a symbolic or absolute address in Bulk Core, Page Memory, or High-Speed Data Buffer. The Control Memory Address may be represented by four terms in the form $a(r)\pm k, cd$.
 - • a This entry is required only if the optional term (r_2) is omitted. This term may be either a symbol or a constant.
 - • k This optional term must be a constant and modifies 'a'.
 - • r This entry may be one of the following nine registers: R0 through R7, DP. The contents of this specified register is added to the value $a\pm k$ at execution time. The result defines the Control Memory Address. The contents of the register can be considered the base address, and the $a\pm k$ expression can be considered the displacement.
 - • cd This entry is the Control Digit. A Control Digit indicates that after the specified instruction is completed a step is desired. This step may increment or decrement the data pointer (DP) register by one and/or decrement the block length (BL) register by one. The Control Digit may be specified by an $a\pm k$ type of expression, where 'a' and k are defined as above.

<u>cd Values</u>	<u>Action</u>
1	Decrement BL
2	Increment DP
3	Decrement BL and increment DP
4	Decrement DP
5	Decrement BL and DP

The Control Digit is a valid entry only when the base register option has been selected, and the register forming the base register is the DP register.

BAL

Note

When a programmer branches and links to a subroutine, he generally will return by issuing an unconditional branch on the register r_1 that specified the branch and link instruction:

Example

BAL, R2 SUB

•
•
•

SUB

•
•
•

B (R2)

RPT

Repeat

This instruction will execute the following instruction the number of times specified in the repeat constant term a±k. If a±k is omitted, it is assumed that FL1 previously has been loaded with the number of times minus one, the next instruction is to be repeated.

Format

Label	Command	Argument	Comment
symbol	<u>RPT</u> , a±k		

- Label Any valid symbol or blank.
 - Command RPT
 - • a±k 'a' may be either a constant or a symbol, and k is an optional constant modifier. The value of this optional term specifies the number of times the following instruction will be repeated, i.e., $1 \leq a \pm k \leq 256$.
 - • blank Assumes FL1 has been loaded with the number of times, minus one, that the next instruction is to be repeated. FL1 should be loaded with a constant from 0 to 255.
 - Argument No entry required.
- Note FL1 will be decremented to zero when this instruction is executed.

LOOP

Loop

This instruction will sequentially cycle the program from the program counter location following the loop instruction up to and including the address specified in the Argument Field. The loop may be cycled any number of iterations from 1 to 256. After the loop has cycled the specified number of times, the program continues with the next address following the Argument Field address.

Format	Label	Command	Argument	Comment
	symbol	<u>LOOP</u> , $a_1^{\pm k_1}$	<u>$a_2(r)^{\pm k_2}$</u>	

- Label Any valid symbol or blank.

- ### • Command LOOP

- • $a_1 \pm k_1$ a_1 may be either a constant or a symbol, and k_1 is an optional constant modifier. The value of this optional term specifies the number of times the program will be cycled.

- • blank APPLE assumes the number of loop iterations, minus one, has already been loaded into F11 by the programmer.

- Argument The Control Memory Address is a symbolic or absolute address in Bulk Core, Page Memory, or High Speed Data Buffer. The Control Memory Address may be represented by three terms in the form $a_2(r)\pm k_2$.

- $a_2^{\pm k_2}$ a_2 may be either a constant or a symbol, and k_2 is an optional constant modifier. The value of the required term specifies the Control Memory Address of the last instruction of the sequence of instructions cycled by the LOOP.

- • r This entry may be one of the following nine registers: R0 through R7, DP. The contents of this specified register is added to the value of $a_2 \pm k_2$ at execution time. The result defines the Control Memory Address. The contents of the register can be considered the base address, and the $a_2 \pm k_2$ expression can be considered the displacement.

Note 1

Instructions that alter the program counter, i.e., branches, skips, external functions, etc., will produce unpredictable results if used within a loop.
Also, Load and Store register instructions are illegal within a loop.

Note 2

Execution times can be improved for instructions within a loop.

Note 3

FL1 register will be decremented to zero upon completion of the loop.

Note 4

The register modification term, r , is only legal when the number of iterations term, $a_1 \pm k_1$, is omitted.

**REGISTER
INSTRUCTIONS**

The register instructions allow the programmer to either alter or save the contents of STARAN S registers.

The register instructions are:

<u>Mnemonic</u>	<u>Instruction</u>
LRR	Load Register from Register
LI	Load Register with Immediate Data
LR	Load Register from Control Memory
SR	Store Register in Control Memory
INCR	Increment the Register
DECR	Decrement the Register
LPSW	Load Program Status Word
SPSW	Swap Program Status Word

Table 2-1. Registers

<u>Mnemonic</u>	<u>Register Name</u>	<u>Length in Bits</u>
AS	Array Selector	32
ASH	Most-Significant Bits of Array Selector	16
ASL	Least-Significant Bits of Array Selector	16
BL	Block Length Counter	16
DP	Data Pointer	16
C	Common Register	32
CH	Most-Significant Bits of Common Register	16
CL	Least-Significant Bits of Common Register	16
F	Field Register group (FL1, FP3, FP1, FP2)	32
FL1	Field Length Counter 1, Bits 0 to 7 of F	8
FP3	Field Pointer 3, Bits 8 to 15 of F	8
FP1	Field Pointer 1, Bits 16 to 23 of F	8
FP2	Field Pointer 2, Bits 24 to 31 of F	8
FL2	Field Length Counter 2	8
FPE	Field Pointer Extra	8
PC	Program Counter, Most-Significant Bits of PSW	16
IMASK	Interrupt Mask, Least-Significant Bits of PSW	4
R0	Branch and Link Register 0	32
R1	Branch and Link Register 1	32
R2	Branch and Link Register 2	32
R3	Branch and Link Register 3	32
R4	Branch and Link Register 4	32
R5	Branch and Link Register 5	32
R6	Branch and Link Register 6	32
R7	Branch and Link Register 7	32

Table 2-2. Register Combinations

<u>Valid Register Combinations</u>	<u>Length in Bits</u>
(ASH, ASL) or AS	32
(BL, DP)	32
(CH, CL) or C	32
(FL1, FP3, FP1, FP2) or F	32
(FL1, FP3)	16
(FP3, FP1)	16
(FP1, FP2)	16
(FP2, FL1)	16
(FL2, FPE)	16
(PC, IMASK)	32

LRR**Load Register From Register**

This instruction will load register or valid register combination r_2 with the contents of register or valid register combination r_1 . The contents the source register is not affected, and the original contents of the destination register is destroyed.

Format	Label	Command	Argument	Comment
	symbol	<u>LRR</u> , k_s	r_2, r_1	

- Label Any valid symbol or blank.
- Command LRR
- • k_s k_s may be either a constant or a symbol.

Legal values:

- | | |
|--------|--|
| 1 | - Shift the contents of the source register (r_1) left end around 8 bits before loading. |
| 2 | - Shift the contents of the source register (r_1) left end around 16 bits before loading. |
| 3 | - Shift the contents of the source register (r_1) left end around 24 bits before loading. |
| symbol | - Must be equal to a value of 1, 2, or 3. |
| blank | - APPLE will provide a shift constant to the source register r_1 to align the least-significant bits of the registers. (See examples.) |

- Argument Both register entries are required.

- • r_2 The destination register(s)

Valid Entries:

Any register or register combination noted in table 2-2.

- • r_1 The source register(s)

Valid Entries:

Any register or register combination noted in table 2-2.

Note 1 An R (Register) error indicates both r_1 and r_2 are branch and link registers, which is invalid.

R LRR R0, R1

Note 2 A T (Truncation) error warns that r_1 is a larger register than r_2 ; therefore all bits cannot be loaded.

LRR

• Example

T LRR FP1,(PC,IMASK)

32-bit (PC,IMASK) cannot be loaded into 8-bit FP1.

Note 3

A W (Warning) error warns that r_1 is a smaller register than r_2 . Not only is r_1 loaded into r_2 , but also the other register or registers in r_1 's group are loaded into r_2 . (See Reference Manual, Bus Positions.)

• Example

W LRR AS,FP2

32-bit register AS is loaded with the four 8-bit registers, FL1, FP3, FP1, FP2.

LI**Load Register with Immediate Data**

This instruction will load register or valid register combination r with the value of a±k in the Argument Field.

Format	Label	Command	Argument	Comment
	symbol	<u>LI</u> , k _s	r, a±k	

- Label Any valid symbol or blank.

- Command LI

- • k_s k_s may be either a constant or a symbol.

Legal values:

- | | |
|--------|---|
| 1 | - Shift the value of a±k left end-around 8 bits before loading. |
| 2 | - Shift the value of a±k left end-around 16 bits before loading. |
| 3 | - Shift the value of a±k left end-around 24 bits before loading. |
| symbol | - Must be equal to a value of 1, 2, or 3. |
| blank | - APPLE will provide a shift constant to the data to align the least-significant bit of the data with the least-significant bit of the register(s) specified. |

- Argument Both entries are required.

- • r The destination register(s).

Valid entries:

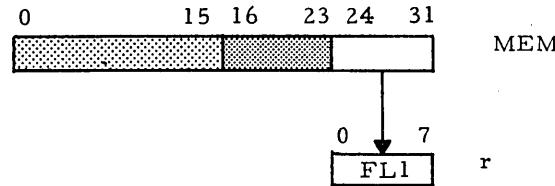
Any register or register combination noted in table 2-2.

- • a±k The immediate value to be loaded may be a single-term or a double-term expression whose value is less than 65,536₁₀. 'a' may be either a constant or a symbol; k is an optional constant modifier.

LI

Example 1*

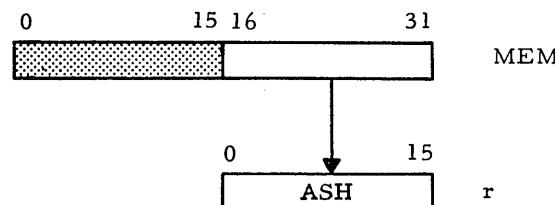
LI FL1, MEM



* Typical for 8-bit registers

Example 2**

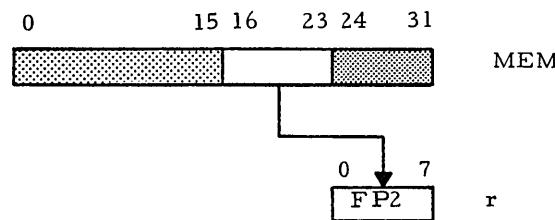
LI ASH, MEM



** Typical for 16-bit registers

Example 3

LI, 3 FP2, MEM



LR

Load Register From Control Memory

This instruction will load the register or valid register combination r_2 with the contents of the Control Memory Address specified by $a(r_1)\pm k$. The Control Memory Address is a symbolic or absolute address in Bulk Core or High-Speed Data Buffer. The Control Memory Address may be represented by four terms in the form $a(r)\pm k, cd$. The contents of the control memory address is not affected. However, the contents of the base register may be changed by the control digit (cd). The original contents of the destination register is destroyed.

Format

Label	Command	Argument	Comment
symbol	<u>LR</u> , k_s	$r_2 \underline{a(r_1)\pm k, cd}$	

- Label Any valid symbol or blank.

- Command LR

- • k_s k_s may be either a constant or a symbol.

Legal values:

- | | |
|--------|--|
| 1 | - Shift the contents of the address $a(r_1)\pm k$, left end-around 8 bits before loading the register. |
| 2 | - Shift the contents of the address $a(r_1)\pm k$, left end-around 16 bits before loading the register. |
| 3 | - Shift the contents of the address $a(r_1)\pm k$, left end-around 24 bits before loading the register. |
| symbol | - Must be equal to a value of 1, 2, or 3. |
| blank | - APPLE assumes that no shifting is desired. |

- Argument Two entries are required.

- • r_2 The destination register(s).

Valid entries:

Any register or register combination noted in table 2-2.

- • $a\pm k$ 'a' may be either a constant or a symbol; k is an optional constant modifier. The value of this term specifies a Control Memory Address.

- • r_1 This entry may be one of the following nine registers: R0 through R7, DP.

The contents of the specified register is added to the value $a\pm k$ at execution time. The result defines the Control Memory Address. The contents of the register can be considered the base address, and the $a\pm k$ expression can be considered the displacement.

LR

• • cd

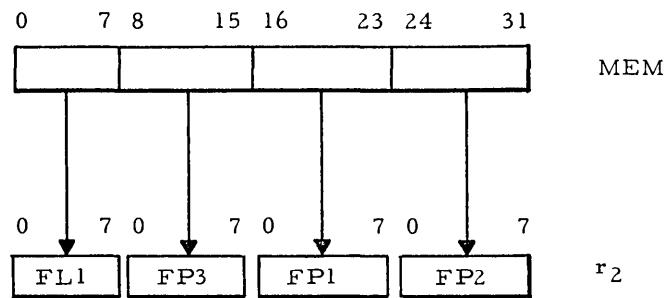
This entry is the Control Digit. A Control Digit indicates that after the specified instruction is completed a step is desired. This step may increment or decrement the pointer (DP) register by one and/or decrement the block length (BL) register by one. The Control Digit may be specified by an $a\pm k$ type of term, where, 'a' and k are as defined as above.

<u>cd Values</u>	<u>Action</u>
1	Decrement BL
2	Increment DP
3	Decrement BL and increment DP
4	Decrement DP
5	Decrement BL and DP

The Control Digit is a valid entry only when the base register option has been selected, and the register forming the base register is the DP register.

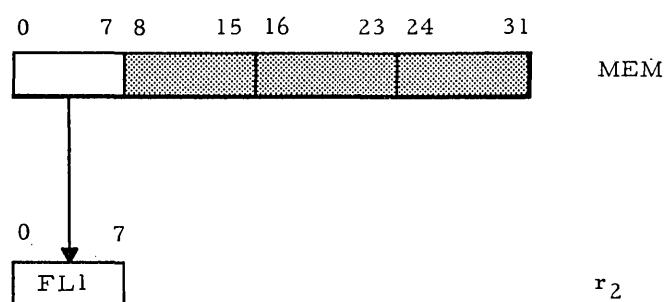
Example 1

LR F, MEM



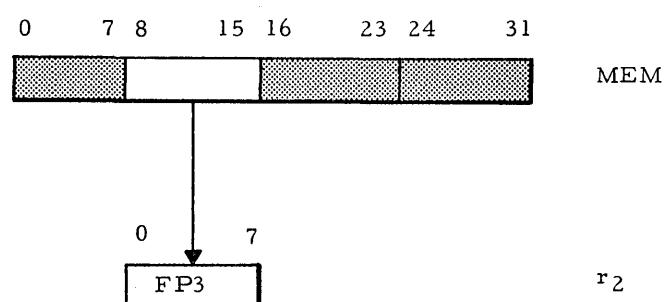
Example 2

LR FLL1, MEM



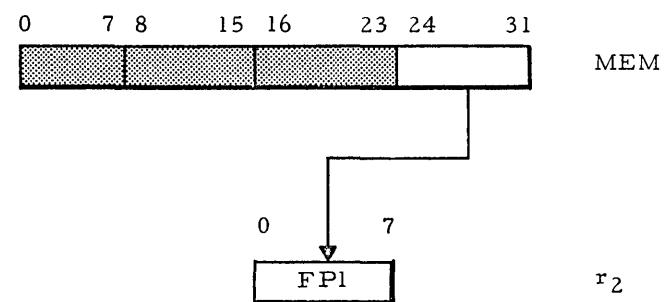
Example 3

LR FP3, MEM



Example 4

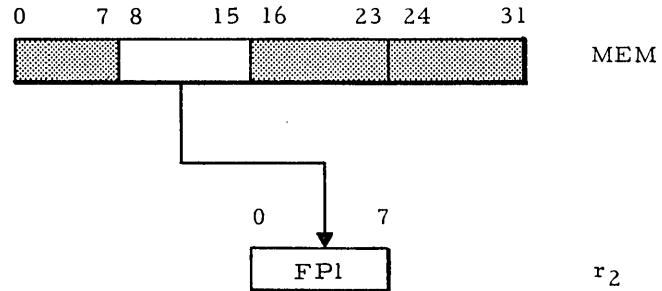
LR, 1 FP1, MEM



LR

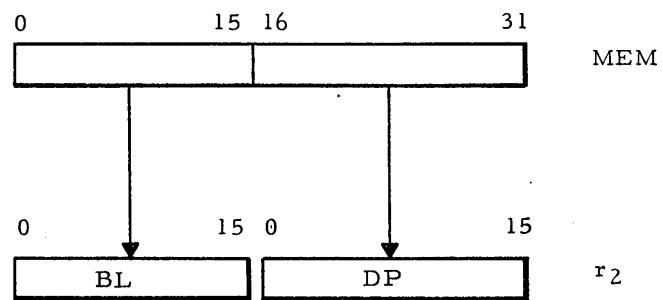
Example 5

LR, 3 FP1, MEM



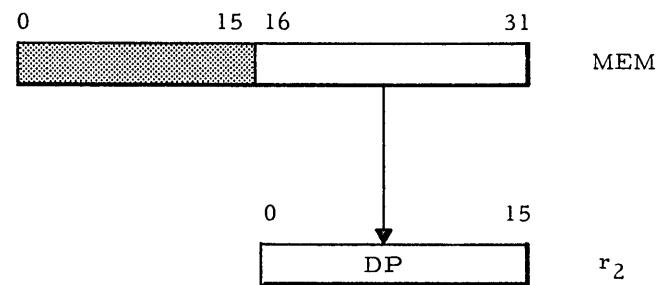
Example 6

LR (BL, DP), MEM



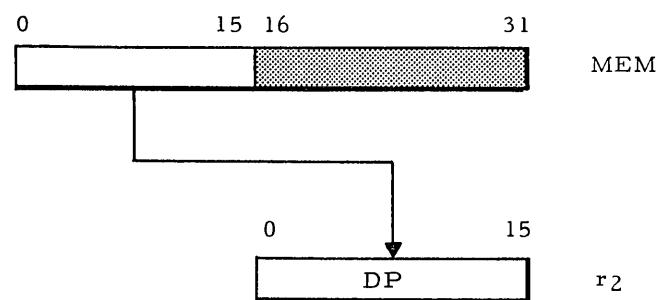
Example 7

LR DP, MEM



Example 8

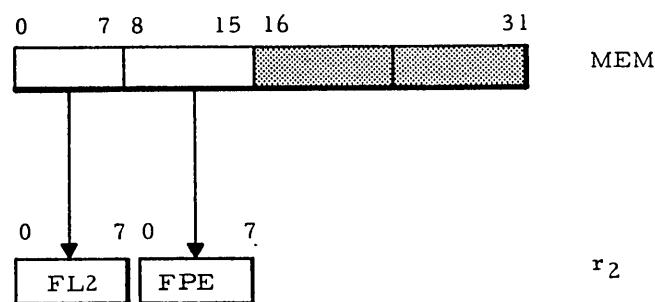
LR, 2 DP, MEM



LR

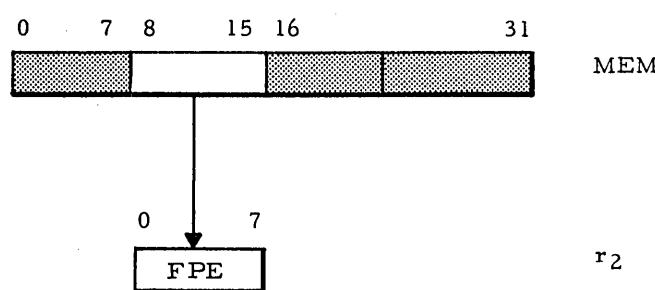
Example 9

LR (FL2, FPE), MEM



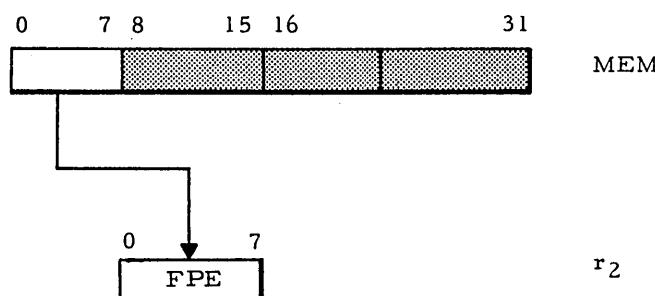
Example 10

LR FPE, MEM



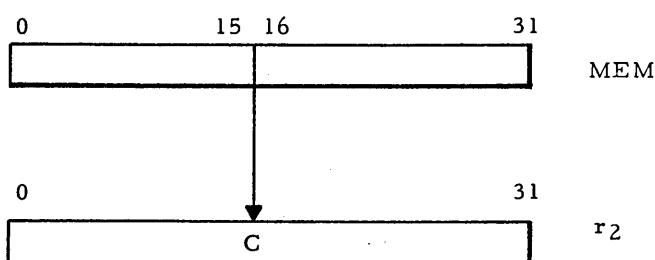
Example 11

LR, 3 FPE, MEM



Example 12

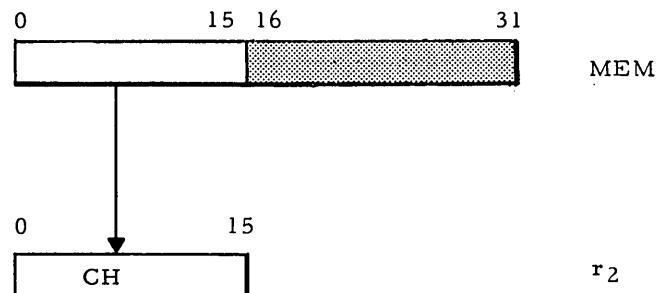
LR C, MEM



LR

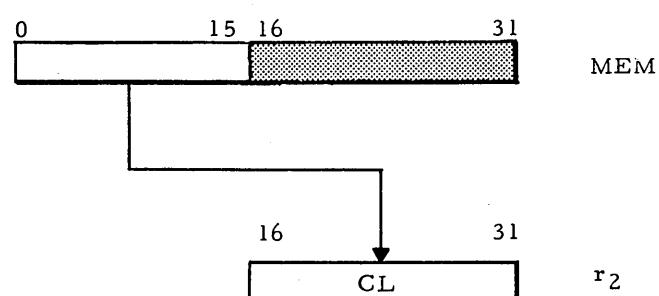
Example 13

LR CH, MEM



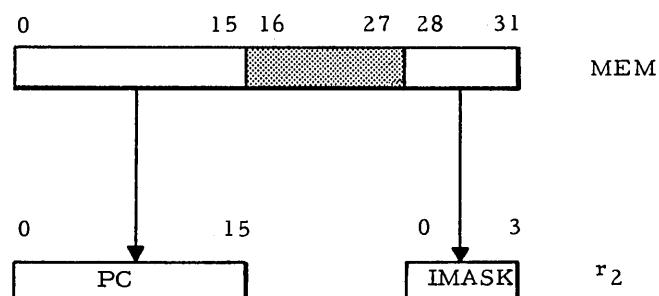
Example 14

LR, 2 CL, MEM



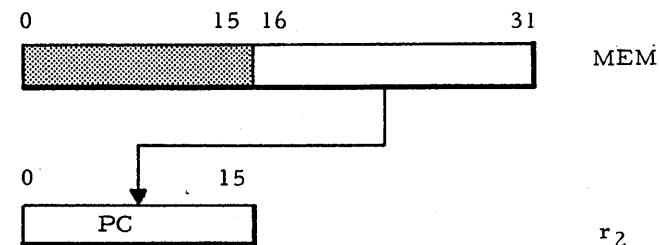
Example 15

LR (PC, IMASK), MEM



Example 16

LR, 2 PC, MEM



Store Register in Control Memory

This instruction will store the contents of the register or valid register combination r_2 in the Control Memory Address specified in $a(r_1)\pm k, cd$. The Control Memory Address is a symbolic or absolute address in Bulk Core or High-Speed Data Buffer. The Control Memory Address may be represented by four terms in the form $a(r)\pm k, cd$. The contents of the source register is not affected and the contents of the Control Memory Address destination is destroyed. The contents of the base register (r_1) may be changed by the control digit, cd .

Format	Label	Command	Argument	Comment
	symbol	<u>SR</u> , k_s	<u>r_2</u> , <u>$a(r_1)\pm k$</u> , cd	

- Label Any valid symbol or blank.

- Command SR

- • k_s k_s may be either a constant or a symbol.

Legal values:

- | | |
|--------|--|
| 1 | - Shift the contents of register (r_2) left end-around 8 bits before storing in Control Memory. |
| 2 | - Shift the contents of register (r_2) left end-around 16 bits before storing in Control Memory. |
| 3 | - Shift the contents of register (r_2) left end-around 24 bits before storing in Control Memory. |
| symbol | - Must be equal to a value of 1, 2, or 3. |
| blank | - APPLE assumes that no shifting is desired. |

- Argument Two entries are required.

- • r_2 The source register(s).

Valid entries:

Any register or register combination noted in table 2-2.

- • $a\pm k$ 'a' may be either a constant or a symbol; k is an optional constant modifier. The value of the term specifies the Control Memory Address.

- • r_1 This entry may be one of the following nine registers: R0 through R7, DP.

The contents of the specified register is added to the value $a\pm k$ at execution time. The result defines the Control Memory Address. The contents of the register can be considered the base address, and the $a\pm k$ expression can be considered the displacement.

- • cd This entry is the Control Digit. A Control Digit indicates that after the specified instruction is completed a step is desired. This step may increment or decrement the pointer (DP) register by one and/or decrement the block length (BL) register by one. The Control Digit may be specified by an $a\pm k$ type of term, where 'a' and k are defined as above.

SR

• • cd

Valid entries:

cd Values	Action (After the Storage Operations)
1	Decrement BL register
2	Increment DP register
3	Increment DP register and decrement BL register
4	Decrement DP
5	Decrement both DP and BL registers

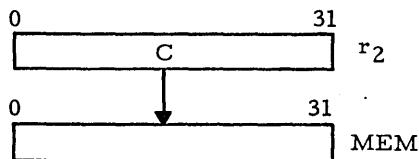
The Control Digit is a valid entry only when the base register option has been selected, and the register forming the base register is the DP register.

Note

When register r_2 is not a 32-bit register or group, a W (Warning) error is produced to remind the programmer that the whole 32-bit group will be stored.

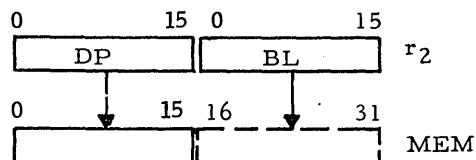
Example 1

SR C, MEM



Example 2

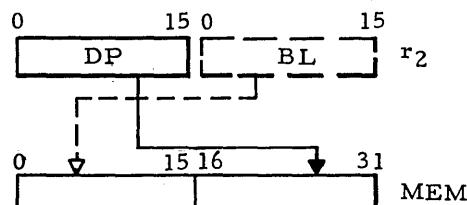
W SR DP, MEM



When register r_2 is not a 32-bit register or group, a W (Warning) error is produced to remind the programmer that the whole 32-bit group will be stored. Location MEM will contain both DP and BL.

Example 3

W SR, 2 DP, MEM



INCR

Increment the Register

The contents of the registers specified in the argument field by the term r_1, \dots, r_n will be incremented by one.

Format

Label	Command	Argument	Comment
symbol	<u>INCR</u>	r_1, \dots, r_n	

- Label May be any valid symbol or blank.

- Command INCR

- Argument The term r_1, \dots, r_n identifies the specific registers to be incremented by one.

Valid entries:

FP1	- Field Pointer 1
FP2	- Field Pointer 2
FP3	- Field Pointer 3
FPE	- Field Pointer Extra
FL1	- Field Length Counter 1, Decrementable only
FL2	- Field Length Counter 2, Decrementable only
BL	- Block Length Counter, Decrementable only
DP	- Data Pointer

Note 1 If the DP is specified in the Argument Field, it must be the only register listed.

Note 2 Only one of the three "decrementable only" registers, FL1, FL2, and BL, may be used in a single instruction. If one is chosen it will be decremented, even though the other registers chosen will be incremented.

Note 3 FP2 and FP3 may not be used in the same instruction.

Note 4 A register may not appear more than once in any given INCR instruction.

Note 5 A W (Warning) error is produced whenever a "decrementable only" register is specified. Such registers will be decremented rather than incremented.

DECR**Decrement the Register**

The contents of the registers specified in the argument field by the term r_1, \dots, r_n will be decremented by one.

Format

Label	Command	Argument	Comment
symbol	<u>DECR</u>	r_1, \dots, r_n	

- Label Any valid symbol or blank.
- Command DECR
- Argument The term r_1, \dots, r_n identifies the specific registers to be decremented by one.

Valid entries:

FP1	- Field Pointer 1
FP2	- Field Pointer 2
FP3	- Field Pointer 3
FPE	- Field Pointer Extra
FL1	- Field Length Counter 1, Decrementable only
FL2	- Field Length Counter 2, Decrementable only
BL	- Block Length Counter, Decrementable only
DP	- Data Pointer

- Note 1 If the DP is specified in the Argument Field, it must be the only register listed.
- Note 2 Only one of the three "decrementable only" registers, FL1, FL2, and BL, may be chosen in a single instruction.
- Note 3 FP2 and FP3 may not be used in the same instruction.
- Note 4 A register may not appear more than once in any given DECR instruction.

LPSW

Load Program Status Word

This instruction will load the contents of a designated Control Memory Address into the Program Counter (PC) and Interrupt Mask (IMASK) registers. From the Control Memory word, bits 0 through 15 are loaded into the Program Counter and bits 28 through 31 into the Interrupt Mask. The contents of the source memory word is not affected. The original contents of the registers are destroyed.

Format	Label	Command	Argument	Comment
	symbol	<u>LPSW</u> , k_s	$a(r)\pm k, cd$	

- Label Any valid symbol or blank.

- Command LPSW

- • k_s k_s may be either a constant or a symbol.

Legal values:

- | | |
|--------|--|
| 1 | - Shift the contents of the address $a(r_1)\pm k$, left end-around 8 bits before loading the register. |
| 2 | - Shift the contents of the address $a(r_1)\pm k$, left end-around 16 bits before loading the register. |
| 3 | - Shift the contents of the address $a(r_1)\pm k$, left end-around 24 bits before loading the register. |
| symbol | - Must be equal to a value of 1, 2, or 3. |
| blank | - APPLE assumes that no shifting is desired. |

- Argument The Control Memory Address is a symbolic or absolute address in Bulk Core or High-Speed Data Buffer. The Control Memory Address may be represented by four terms in the form $a(r)\pm k, cd$.

- • $a\pm k$ ' a ' may be either a constant or a symbol; k is an optional constant modifier. The value of this term specifies a Control Memory address.

- • r This entry may be one of the following nine registers: R0 through R7, DP.

The contents of the specified register is added to the value $a\pm k$ at execution time. The result defines the Control Memory Address. The contents of the register can be considered the base address; and the $a\pm k$ expression can be considered the displacement.

• • cd

This entry is the Control Digit. A Control Digit indicates that after the specified instruction is completed a step is desired. This step may increment or decrement the data pointer (DP) register by one and/or decrement the block length (BL) register by one. The Control Digit may be specified by $a\pm k$ type of term, where 'a' and k are defined as above.

<u>cd Values</u>	<u>Action</u>
1	Decrement BL
2	Increment DP
3	Decrement BL and increment DP
4	Decrement DP
5	Decrement BL and DP

The Control Digit is a valid entry only when the base register option has been selected, and the register forming the base register is the DP register.

SPSW

Swap Program Status Word

This instruction will store the current Program Counter (PC) and Interrupt Mask (IMASK) registers in the designated Control Memory Address. If the Control Memory Address is an even address (see EVEN), the contents of the next location will be loaded into the PC and IMASK registers. Loading the PC register causes a branch to the address loaded. If the Control Memory address is an odd address, only the store takes place.

Format	Label	Command	Argument	Comment
	symbol	<u>SPSW</u>	<u>a(r)±k, cd</u>	

- Label Any valid symbol or blank.
- Command SPSW
- Argument The Control Memory Address is a symbolic or absolute address in Bulk Core or High-Speed Data Buffer. The Control Memory Address may be represented by four terms in the form $a(r)\pm k, cd$.
- • a±k 'a' may be either a constant or a symbol; k is an optional constant modifier. The value of this term specifies a Control Memory Address.
- • r This entry may be one of the following nine registers: R0 through R7, DP. The contents of the specified register is added to the value a±k at execution time. The result defines the Control Memory Address. The contents of the register can be considered the base address, and the a±k expression can be considered the displacement.
- • cd This entry is the Control Digit. A Control Digit indicates that after the specified instruction is completed a step is desired. This step may increment or decrement the pointer (DP) register by one and/or decrement the block length (BL) register by one. The Control Digit may be specified by an a±k type of term, where 'a' and k are defined as above.

cd Values	Action
1	Decrement BL
2	Increment DP
3	Decrement BL and increment DP
4	Decrement DP
5	Decrement BL and DP

The Control Digit is a valid entry only when the base register option has been selected, and the register forming the base register is the DP register.

SPSW

Example

```
PUT      EVEN
GET      DS
          DC      X'90000009'
.
.
.
SPSW    PUT
```

The current PC and IMASK registers will be stored at location PUT (note that the address of PUT will always be even); then X'9000' will be loaded into PC register and X'9' loaded into IMASK register. Since the program counter (PC) is being loaded, a branch to location X'9000' occurs.

ASSOCIATIVE INSTRUCTIONS

The associative instructions allow the programmer to load, store, search, move, and perform arithmetic operations on the associative array memory and the response store registers X, Y, and M.

LOADS

This group of associative instructions allows the programmer to load the response store registers or the Common register from an associative memory bit column or an associative memory field, respectively. All instructions dealing with response store registers and/or associative memory fields or bit columns, only affect those associative array memory modules enabled via the Array Select register. The response store registers and associative array memory modules disabled via the Array Select register remain unchanged.

<u>Mnemonic</u>	<u>Instructions</u>
L	Load Response Store Register
LN	Load Complemented
LOR	Load Logical OR
LORN	Load Logical OR Complemented
LAND	Load Logical AND
LANDN	Load Logical AND Complemented
LXOR	Load Logical Exclusive OR
LXORN	Load Logical Exclusive OR Complemented
LC	Load Common Register from an Associative Memory Word
LCM	Load a Common Register Field from an Associative Memory Word
SET	Set Response Store Register
CLR	Clear Response Store Register
ROT	Rotate Response Store Register

Load Response Store Register

This instruction will load the response store register, rs_2 , with the designated source. The content of the source is not affected, and the original content of the destination, rs_2 , is destroyed.

Format	Label	Command	Argument	Comment
		<u>L</u>	rs_2 $\left[\begin{array}{c} rs_1 \\ a \pm k \\ r \end{array} \right]$	
• Label				Any valid symbol or blank.
• Command		L		
• Argument				Two entries are required. The first entry is the destination; the second entry is the source. As shown there are three distinct types of source expressions. The brackets are not a part of the possible argument field terms.
• • rs_2				The destination response store register.
				Valid entries: X - X response store register Y - Y response store register M - M response store register
• • rs_1				The source response store register.
				Valid entries: X - X response store register Y - Y response store register M - M response store register
• • $a \pm k$				'a' may be a constant or a symbol; k is an optional constant modifier. k is legal only when 'a' is present as a symbol. If 'a' was defined as a field via a DF instruction, the most-significant bit position is the value used. This term represents a source bit position in all words of enabled associative memory. The value of $a \pm k$ should be $0 \leq a \pm k \leq 255$.
• • r				A field pointer register, which may be post-incremented or post-decremented. If this form is used, the response store register, rs_2 , is loaded indirectly through this register. This register contains the address of the source bit column.
				Valid entries: FP1 Field Pointer 1 FP1+ Field Pointer 1 with a post-increment FP1- Field Pointer 1 with a post-decrement FP2 Field Pointer 2 FP2+ Field Pointer 2 with a post-increment FP2- Field Pointer 2 with a post-decrement FP3 Field Pointer 3 FP3+ Field Pointer 3 with a post-increment FP3- Field Pointer 3 with a post-decrement

L

Example 1

L M, Y

		<u>Before</u>		<u>After</u>	
		M	Y	M	Y
256 Bits	0	0	0	0	0
	0	1	1	1	1
	1	0	0	0	0
	1	1	1	1	1

Example 2

TAG DF 9, 4
· ·
· ·
· ·
L Y, TAG

		<u>Before</u>		<u>After</u>	
		Y	Array Memory Column 9	Y	Array Memory Column 9
256 Bits	0	0	0	0	0
	0	1	1	1	1
	1	0	0	0	0
	1	1	1	1	1

Example 3

TAG DF 9, 4
· ·
· ·
· ·
L X, TAG

		<u>Before</u>		<u>After</u>	
		X	Array Memory Column 9	X	Array Memory Column 9
256 Bits	0	0	0	0	0
	0	1	1	1	1
	1	0	0	0	0
	1	1	1	1	1

LN

Load Complemented

This instruction will load the response store register, rs_2 , with the one's complement value of the designated source. The content of the source is not affected and the original content of the destination, rs_2 , is destroyed.

Format

Label	Command	Argument	Comment
symbol	<u>LN</u>	rs_2 [$\frac{rs_1}{a \pm k}$] r	

- Label

Any valid symbol or blank.

- Command

LN

- Argument

Two entries are required. The first entry is the destination; the second entry is the source. As shown, there are three distinct types of source expressions. The brackets are not a part of the possible argument field terms.

- • rs_2

The destination response store register.

Valid entries:

X - X response store register
Y - Y response store register
M - M response store register

- • rs_1

The source response store register.

Valid entries:

X - X response store register
Y - Y response store register
M - M response store register

- • $a \pm k$

'a' may be a constant or a symbol; k is an optional constant modifier. k is legal only when 'a' is present as a symbol. If 'a' was defined as a field via a DF instruction, the most-significant bit position is the value used. This term represents a source bit position in all words of enabled associative memory. The value of $a \pm k$ should be $0 \leq a \pm k \leq 255$.

- • r

A field pointer register. If this form is used, the response store register, rs_2 , is loaded indirectly through this register. This register contains the address of the source bit column.

Valid entries:

FP1	Field Pointer 1
FP1+	Field Pointer 1 with a post-increment
FP1-	Field Pointer 1 with a post-decrement
FP2	Field Pointer 2
FP2+	Field Pointer 2 with a post-increment
FP2-	Field Pointer 2 with a post-decrement
FP3	Field Pointer 3
FP3+	Field Pointer 3 with a post-increment
FP3-	Field Pointer 3 with a post-decrement

LN

Note

If M is chosen for the rs₂ entry, the original content of the X response store register is destroyed when this multiple instruction is executed.

Example 1

LN M, Y

Before			After		
X	Y	M	X	Y	M
1	0	0	d	0	1
1	1	0	e	1	0
0	0	1	s	0	1
0	1	1	t	1	0
.	.	.	r	.	.
.	.	.	o	.	.
.	.	.	y	.	.
.	.	.	e	.	.
.	.	.	d	.	.

Example 2

TAG DF 9, 4

• •

• •

• •

LN Y, TAG

Before			After		
Y	Array Memory Col 9		Y	Array Memory Col 9	
0	0		1	0	
0	1		0	1	
1	0		1	0	
1	1		0	1	
.	.		.	.	
.	.		.	.	
.	.		.	.	
.	.		.	.	

Example 3

TAG DF 9, 4

• •

• •

• •

LI FP1, TAG

LN X, FP1

Before			After		
X	Array Memory Col 9		X	Array Memory Col 9	
0	0		1	0	
0	1		0	1	
1	0		1	0	
1	1		0	1	
.	.		.	.	
.	.		.	.	
.	.		.	.	
.	.		.	.	

LOR

Load Logical OR

This instruction will load the response store register, rs_2 , with a logical inclusive OR of itself and the value of the designated source. The content of the source is not affected and the original content of the destination, rs_2 , is destroyed.

Format	Label	Command	Argument	Comment
	symbol	<u>LOR</u>	\underline{rs}_2 $\underline{a \pm k}$ \underline{r}	

- Label Any valid symbol or blank.
 - Command LOR
 - Argument Two entries are required. The first entry is the destination; the second entry is the source. As shown, there are three distinct types of source expressions. The brackets are not a part of the possible argument field terms.
 - • rs₂ The destination response store register.
Valid entries:
X - X response store register
Y - Y response store register
M - M response store register
 - • rs₁ The source response store register.
Valid entries:
X - X response store register
Y - Y response store register
M - M response store register
 - • a±k 'a' may be a constant or a symbol; k is an optional constant modifier. k is legal only when 'a' is present as a symbol. If 'a' was defined as a field via a DF instruction, the most-significant bit position is the value used. This term represents a source bit position in all words of enabled associative memory. The value of a±k should be $0 \leq a \pm k \leq 255$.
 - • r A field pointer register. If this form is used, the response store register, rs₂, is loaded indirectly through this register. This register contains the address of the source bit column.

Valid entries:

FP1	Field Pointer 1	
FP1+	Field Pointer 1	with a post-increment
FP1-	Field Pointer 1	with a post-decrement
FP2	Field Pointer 2	
FP2+	Field Pointer 2	with a post-increment
FP2-	Field Pointer 2	with a post-decrement
FP3	Field Pointer 3	
FP3+	Field Pointer 3	with a post-increment
FP3-	Field Pointer 3	with a post-decrement

LOR

Note

If M is chosen for the rs_2 entry, the original content of the X response store register is destroyed when this multiple instruction is executed.

Example 1

LOR M, Y

<u>Before</u>			<u>After</u>		
X	Y	M	X	Y	M
1	0	0	d	0	0
1	1	0	e	1	1
0	0	1	s	0	1
0	1	1	t	1	1
.	.	.	r	.	.
.	.	.	o	.	.
.	.	.	y	.	.
256 Bits	.	.	d	.	.

Example 2

TAG DF 9, 4

⋮ ⋮
⋮ ⋮
⋮ ⋮

LOR Y, TAG

<u>Before</u>			<u>After</u>		
Y	Array Memory	Col 9	Y	Array Memory	Col 9
0	0	0	0	0	0
0	1	1	1	1	1
1	0	0	1	0	0
1	1	1	1	1	1
.
.
.
256 Bits

Example 3

TAG DF 9, 4

⋮ ⋮
⋮ ⋮
⋮ ⋮

LI FP2, TAG
LOR X, FP2

<u>Before</u>			<u>After</u>		
X	Array Memory	Col 9	X	Array Memory	Col 9
0	0	0	0	0	0
0	1	1	1	1	1
1	0	0	1	0	0
1	1	1	1	1	1
.
.
.
256 Bits

LORN

Load Logical OR Complemented

This instruction will load the response store register, rs_2 , with the logical inclusive OR of itself and the one's complement of the value of the designated source. The content of the source is not affected and the original content of the destination, rs_2 , is destroyed.

Format

Label	Command	Argument	Comment
symbol	<u>LORN</u>	rs_2 [rs_1 $a \pm k$ r]	

- Label Any valid symbol or blank.

- Command LORN

- Argument Two entries are required. The first entry is the destination; the second entry is the source. As shown, there are three distinct types of source expressions. The brackets are not a part of the possible argument field terms.

- • rs_2 The destination response store register.

Valid entries:

X - X response store register
Y - Y response store register
M - M response store register

- • rs_1 The source response store register.

Valid entries:

X - X response store register
Y - Y response store register
M - M response store register

- • $a \pm k$ ' a ' may be a constant or a symbol; k is an optional constant modifier. k is legal only when ' a ' is present as a symbol. If ' a ' was defined as a field via a DF instruction, the most significant bit position is the value used. This term represents a source bit position in all words of enabled associative memory. The value of $a \pm k$ should be $0 \leq a \pm k \leq 255$.

- • r A field pointer register. If this form is used, the response store register, rs_2 , is loaded indirectly through this register. This register contains the address of the source bit column.

Valid entries:

FP1 Field Pointer 1
FP1+ Field Pointer 1 with a post-increment
FP1- Field Pointer 1 with a post-decrement
FP2 Field Pointer 2
FP2+ Field Pointer 2 with a post-increment
FP2- Field Pointer 2 with a post-decrement
FP3 Field Pointer 3
FP3+ Field Pointer 3 with a post-increment
FP3- Field Pointer 3 with a post-decrement

LORN

Note

If M is chosen for the rs_2 entry, the original content of the X response store register is destroyed when this multiple instruction is executed.

Example 1

LORN M, Y

Before			After		
X	Y	M	X	Y	M
1	0	0	d	0	1
1	1	0	e	1	0
0	0	1	s	0	1
0	1	1	t	1	1
.	.	.	r	.	.
.	.	.	o	.	.
.	.	.	y	.	.
.	.	.	e	.	.
.	.	.	d	.	.

Example 2

TAG DF 9, 4

• •
• •
• .
LORN Y, TAG

Before			After		
Y	Array Memory Col 9		Y	Array Memory Col 9	
0	0		1	0	
0	1		0	1	
1	0		1	0	
1	1		1	1	
.	.		.	.	
.	.		.	.	
.	.		.	.	
.	.		.	.	

Example 3

LORN Y, 9

Before			After		
Y	Array Memory Col 9		Y	Array Memory Col 9	
0	0		1	0	
0	1		0	1	
1	0		1	0	
1	1		1	1	
.	.		.	.	
.	.		.	.	
.	.		.	.	
.	.		.	.	

LAND

Load Logical AND

This instruction will load the response store register, rs_2 , with a logical AND of itself and the value of the designated source. The content of the source is not affected and the original content of the destination, rs_2 , is destroyed.

Format	Label	Command	Argument	Comment
	symbol	<u>LAND</u>	<u>rs</u> ₂ ¹ <u>a</u> _{±k} <u>r</u>	

- Label Any valid symbol or blank.

- Command LAND

- Argument Two entries are required. The first entry is the destination; the second entry is the source. As shown, there are three distinct types of source expressions. The brackets are not a part of the possible argument field terms.

- • rs₂ The destination response store register.

Valid entries:

X - X response store register
Y - Y response store register
M - M response store register

- • rs₁ The source response store register.

Valid entries:

X - X response store register
Y - Y response store register
M - M response store register

- • $a \pm k$ 'a' may be a constant or a symbol; k is an optional constant modifier. k is legal only when 'a' is present as a symbol. If 'a' was defined as a field via a DF instruction, the most significant bit position is the value used. This term represents a source bit position in all words of enabled associative memory. The value of $a \pm k$ should be $0 \leq a \pm k \leq 255$.

- • r A field pointer register. If this form is used, the response store register, rs_2 , is loaded indirectly through this register. This register contains the address of the source bit column.

Valid entries:

FP1	Field Pointer 1
FP1+	Field Pointer 1 with a post-increment
FP1-	Field Pointer 1 with a post-decrement
FP2	Field Pointer 2
FP2+	Field Pointer 2 with a post-increment
FP2-	Field Pointer 2 with a post-decrement
FP3	Field Pointer 3
FP3+	Field Pointer 3 with a post-increment
FP3-	Field Pointer 3 with a post-decrement

LAND

Note

If M is chosen for the rs₂ entry, the original content of the X response store register is destroyed when this multiple instruction is executed.

Example 1

LAND M, Y

<u>Before</u>			<u>After</u>		
X	Y	M	X	Y	M
1	0	0	d	0	0
1	1	0	e	1	0
0	0	1	s	0	0
0	1	1	t	1	1
.	.	.	r	.	.
.	.	.	o	.	.
.	.	.	y	.	.
256 Bits			e	.	.
			d	.	.

Example 2

TAG DF 9, 4

· ·
· ·
· ·
LAND Y, TAG

<u>Before</u>			<u>After</u>		
Y	Array Memory Col 9		Y	Array Memory Col 9	
0	0		0	0	
0	1		0	1	
1	0		0	0	
1	1		1	1	
.	.		.	.	
.	.		.	.	
.	.		.	.	
256 Bits					

Example 3

TAG DF 9, 4

· ·
· ·
· ·
LI FP3, TAG
LAND X, FP3

<u>Before</u>			<u>After</u>		
X	Array Memory Col 9		X	Array Memory Col 9	
0	0		0	0	
0	1		0	1	
1	0		0	0	
1	1		1	1	
.	.		.	.	
.	.		.	.	
.	.		.	.	
256 Bits					

LANDN

Logical AND Complemented

This instruction will load the response store register, rs_2 , with the logical AND of itself and the one's complement of the designated source. The content of the source is not affected and the original content of the destination, rs_2 , is destroyed in the execution of the load.

Format	Label	Command	Argument	Comment
	symbol	<u>LANDN</u>	<u>rs</u> ₂ <u>rs</u> ₁ <u>a</u> _{±k} <u>r</u>	

- Label Any valid symbol or blank.
 - Command LANDN
 - Argument Two entries are required. The first entry is the destination; the second entry is the source. As shown, there are three distinct types of source expressions. The brackets are not a part of the possible argument field terms.
 - • rs₂ The destination response store register.
Valid entries:
X - X response store register
Y - Y response store register
M - M response store register
 - • rs₁ The source response store register.
Valid entries:
X - X response store register
Y - Y response store register
M - M response store register
 - • a±k 'a' may be a constant or a symbol; k is an optional constant modifier. k is legal only when 'a' is present as a symbol. If 'a' was defined as a field via a DF instruction, the most-significant bit position is the value used. This term represents a source bit position in all words of enabled associative memory. The value of a±k should be 0≤a±k≤255.
 - • r A field pointer register. If this form is used, the response store register, rs₂, is loaded indirectly through this register. This register contains the address of the source bit column.
Valid entries:
FP1
FP1+ Field Pointer 1 with a post-increment
FP1- Field Pointer 1 with a post-decrement
FP2 Field Pointer 2
FP2+ Field Pointer 2 with a post-increment
FP2- Field Pointer 2 with a post-decrement
FP3 Field Pointer 3
FP3+ Field Pointer 3 with a post-increment
FP3- Field Pointer 3 with a post-decrement

LANDN

Note

If M is chosen for the rs₂ entry, the original content of the X response store register is destroyed when this multiple instruction is executed.

Example 1

LANDN M, Y

<u>Before</u>			<u>After</u>		
X	Y	M	X	Y	M
1	0	0	d	0	0
1	1	0	e	1	0
0	0	1	s	0	1
0	1	1	t	1	0
.	.	.	r	.	.
.	.	.	o	.	.
.	.	.	y	.	.
256 Bits			e	.	.
			d	.	.

Example 2

LANDN Y, 9

<u>Before</u>			<u>After</u>		
	Array Memory	Col 9		Array Memory	Col 9
Y			Y		
0	0	0	0	0	0
0	0	1	0	0	1
1	0	0	1	0	0
1	1	1	0	1	1
.
.
.
256 Bits					

Example 3

TAG DF 9, 4

.

.

.

LI FP1, TAG

LANDN X, FP1

<u>Before</u>			<u>After</u>		
	Array Memory	Col 9		Array Memory	Col 9
X			X		
0	0	0	0	0	0
0	0	1	0	0	1
1	0	0	1	0	0
1	1	1	0	1	1
.
.
.
256 Bits					

LXOR

Load Logical Exclusive OR

This instruction will load the response store register, rs_2 , with the logical exclusive OR of itself and the value of the designated source. The content of the source is not affected and the original content of the destination, rs_2 , is destroyed.

Format	Label	Command	Argument	Comment
	symbol	<u>LXOR</u>	<u>rs</u> ₂ [<u>rs</u> ₁] [<u>a</u> +k] [r]	

LXOR

Note

If M is chosen for the rs_2 entry, the original content of the X response store register is destroyed when this multiple instruction is executed.

Example 1

LXOR M, Y

<u>Before</u>			<u>After</u>		
X	Y	M	X	Y	M
1	0	0	d	0	0
1	1	0	e	1	1
0	0	1	s	0	1
0	1	1	t	1	0
.	.	.	r	.	.
.	.	.	o	.	.
.	.	.	y	.	.
256 Bits	.	.	d	.	.
			e	.	.
			s	.	.
			t	.	.
			r	.	.
			o	.	.
			y	.	.
			d	.	.
			e	.	.
			s	.	.

Example 2

LXOR Y, 9

<u>Before</u>			<u>After</u>		
	Array Memory	Col 9		Array Memory	Col 9
Y	Col	9	Y	Col	9
0	0	0	0	0	0
0	1	1	1	1	1
1	0	0	1	0	0
1	1	1	0	1	1
.
.
.
256 Bits	.	.	0	0	0
			1	1	1
			1	0	0
			0	1	1
			.	.	.
			.	.	.
			.	.	.
			.	.	.
			.	.	.
			.	.	.

Example 3

LI FP2,X'9'
LXOR X,FP2

<u>Before</u>			<u>After</u>		
	Array Memory	Col 9		Array Memory	Col 9
X	Col	9	X	Col	9
0	0	0	0	0	0
0	1	1	1	1	1
1	0	0	1	0	0
1	1	1	0	1	1
.
.
.
256 Bits	.	.	0	0	0
			1	1	1
			1	0	0
			0	1	1
			.	.	.
			.	.	.
			.	.	.
			.	.	.
			.	.	.
			.	.	.

LXORN

Load Logical Exclusive OR Complemented

This instruction will load the response store register, rs_2 , with the logical exclusive OR of itself and the one's complement of the designated source. The content of the source is not affected and the original content of the destination, rs_2 , is destroyed in the execution of the load.

Format

Label	Command	Argument	Comment
symbol	<u>LXORN</u>	rs_2 , $\begin{bmatrix} rs_1 \\ \underline{\underline{a\pm k}} \\ r \end{bmatrix}$	

• Label

Any valid symbol or blank.

• Command

LXORN

• Argument

Two entries are required. The first entry is the destination; the second entry is the source. As shown, there are three distinct types of source expressions. The brackets are not a part of the possible argument field terms.

• • rs_2

The destination response store register.

Valid entries:

X - X response store register
 Y - Y response store register
 M - M response store register

• • rs_1

The source response store register.

Valid entries:

X - X response store register
 Y - Y response store register
 M - M response store register

• • $a\pm k$

'a' may be a constant or a symbol; k is an optional constant modifier. k is legal only when 'a' is present as a symbol. If 'a' was defined as a field via a DF instruction, the most-significant bit position is the value used. This term represents a source bit position in all words of enabled associative memory. The value of $a\pm k$ should be $0 \leq a\pm k \leq 255$.

• • r

A field pointer register. If this form is used, the response store register, rs_2 , is loaded indirectly through this register. This register contains the address of the source bit column.

Valid entries:

PF1 Field Pointer 1
 FP1+ Field Pointer 1 with a post-increment
 FP1- Field Pointer 1 with a post-decrement
 FP2 Field Pointer 2
 FP2+ Field Pointer 2 with a post-increment
 FP2- Field Pointer 2 with a post-decrement
 FP3 Field Pointer 3
 FP3+ Field Pointer 3 with a post-increment
 FP3- Field Pointer 3 with a post-decrement

LXORN

Note

If M is chosen for the rs₂ entry, the original content of the X response store register is destroyed when this multiple instruction is executed.

Example 1

LXORN M, Y

<u>Before</u>			<u>After</u>		
X	Y	M	X	Y	M
1	0	0	d	0	1
1	1	0	e	1	0
0	0	1	s	0	0
0	1	1	t	1	1
.	.	.	r	.	.
.	.	.	o	.	.
.	.	.	y	.	.
256 Bits			d	.	.

Example 2

LXORN Y, 9

<u>Before</u>			<u>After</u>		
Y	Array Memory	Col 9	Y	Array Memory	Col 9
0	0	0	1	0	0
0	1	1	0	1	0
1	0	0	0	0	0
1	1	1	1	1	1
.
.
.
256 Bits			.	.	.

Example 3

TAG DF 9, 4

.

.

.

.

LI FP3, TAG

LXORN X, FP3

<u>Before</u>			<u>After</u>		
X	Array Memory	Col 9	X	Array Memory	Col 9
0	0	0	1	0	0
0	1	1	0	1	0
1	0	0	0	0	0
1	1	1	1	1	1
.
.
.
256 Bits			.	.	.

LC

Load Common Register from an Associative Memory Word

This instruction will load the Common register, right-justified with a field of the associative memory word whose address is in the link pointer (FP1, FP2).

Format	Label	Command	Argument	Comment
	symbol	<u>LC</u>	<u>a</u>	

- Label Any valid symbol or blank.
- Command LC
- Argument One entry is required, an associative memory field expression.

• • a There are two ways of denoting a field expression:

- 1) 'a' may be in the form

$b\pm i$

where b must be a symbol, and i is an optional constant modifier. b should have been previously defined in a DF instruction. b represents the most-significant bit position and the number of contiguous bits occupied by a field in associative memory. The optional constant modifier, i, modifies only the most-significant bit position of the field.

- 2) 'a' may be in the form

$(b, i)\pm j$

where b may be a constant or a symbol and represents the most-significant bit position of a field. If b was defined as a field via a previous DF instruction, the most-significant bit position is the value used. i must be a constant and represents the number of contiguous bits occupied by the field. j is an optional constant modifying only the most-significant bit position of the field.

Note 1

The link pointer (FP1 and FP2 registers) must be loaded with the address of the particular associative memory word prior to executing this instruction. Loading the link pointer is generally accomplished by use of the FIND, STEP, or RESVFST instruction.

Note 2

If the array memory field length is less than 32 bits, the most-significant bit positions of the Common register are cleared to zero.

Note 3

If the array memory field length is greater than 32 bits, the most significant bits are truncated and the instruction is flagged with a T on the listing.

Note 4

The X response store register is destroyed if shifting is required.

LC

Example

FIND

LC (10, 6)

.

.

.

Y	Array Memory						
	Bit Column						
	10	11	12	13	14	15	
Word 0	0	0	0	0	1	0	1
Word 1	0	1	1	1	0	1	0
.
.
Word n	1	1	0	0	1	1	1
.
.
Word 255	1	1	1	0	0	1	0

FP1

FP2

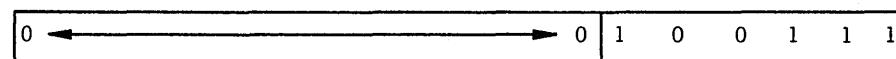
Address of Word n

Common Register

0

25 26

31



LCM

Load a Common Register Field From an Associative Memory Word

This instruction will load a field, a_1 , in the Common register with a field, a_2 , from the word of associative memory whose address is in the link pointer (FP1, FP2). All other bits in the Common register remain unchanged.

Format	Label	Command	Argument	Comment
	symbol	<u>LCM</u>	a_1, a_2	

- Label Any valid symbol or blank.

- Command LCM

- Argument Two entries are required. The first entry is the destination, a field in the Common register, the second entry is the source, a field in a word of associative memory.

- • a_1, a_2 There are two ways of denoting a field expression:

- 1) a_1 or a_2 may be in the form
 $b\pm i$

where b must be a symbol, and i is an optional constant modifier. b should have been previously defined in a DF instruction. b represents the most-significant bit position and the number of contiguous bits occupied by a field in either the Common register or associative memory. The optional constant modifier, i, modifies only the most-significant bit position.

- 2) a_1 or a_2 may be in the form
 $(b, i)\pm j$

where b may be a constant or a symbol and represents the most-significant bit position of a field. If b was defined as a field via a previous DF instruction, the most-significant bit position is the value used. i must be a constant and represents the number of contiguous bits occupied by the field. j is an optional constant modifying only the most significant bit position of the field.

Note 1

The link pointer (FP1 and FP2 registers) must be loaded with the address of the particular associative memory word prior to executing this instruction. Loading the link pointer is generally accomplished by use of the FIND, STEP, or RESVFST instruction.

Note 2

If the associative memory field length is less than the Common register field length, a W (warning flag) is noted on the listing. In this case the associative memory field will be loaded, right justified in the Common register field, and all remaining bits are unchanged.

Note 3

If the associative memory field length is greater than the Common register field length, a T (truncation flag) is noted on the listing. In this case the most-significant bits of the associative memory field are truncated.

Note 4

The X response store register is destroyed if shifting is required.

LCM

Example

FIND

LCM (15, 6), (10, 6)

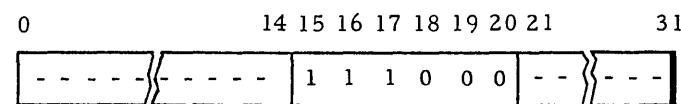
•
•
•

Y	Array Memory Bit Column						
	10	11	12	13	14	15	
Word 0	0	0	1	0	1	0	1
Word 1	0	1	0	1	0	1	0
•	•	•	•	•	•	•	•
•	•	•	•	•	•	•	•
Word n	1	1	1	1	0	0	0
•	•	•	•	•	•	•	•
•	•	•	•	•	•	•	•
Word 255	1	0	0	1	1	0	0

FP1 FP2

Address of Word n

Common Register



SET**Set Response Store Register**

This instruction will set the designated response store, rs, to all ones.

Format

<u>Label</u>	<u>Command</u>	<u>Argument</u>	<u>Comment</u>
symbol	<u>SET</u>	<u>rs</u>	

- Label Any valid symbol or blank.
- Command SET
- Argument One entry is required.

• • rs The designated response store to be set by the instruction.

Valid entries:

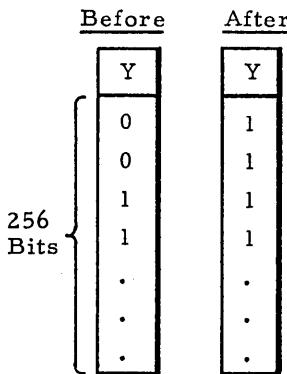
X - X response store register
Y - Y response store register
M - M response store register

Note

If M is chosen for the rs entry, the original content of the X response store register is destroyed when this multiple instruction is executed.

Example

SET Y



CLR**Clear Response Store Register**

This instruction will clear the designated response store, rs, to all zeroes.

Format

Label	Command	Argument	Comment
symbol	<u>CLR</u>	<u>rs</u>	

- Label Any valid symbol or blank.

- Command CLR

- Argument One entry is required.

- • rs The designated response store to be cleared by the instruction.

Valid entries:

X - X response store
Y - Y response store
M - M response store

Note

If M is chosen for the rs entry, the original content of the X response store register is destroyed when this multiple instruction is executed.

Example

CLR M

<u>Before</u>		<u>After</u>	
M	X	M	X
0	0	0	d
0	1	0	e
1	0	0	s
1	1	0	t
.	.	.	r
.	.	.	o
.	.	.	y
256 Bits			e
			d

ROT

Rotate Response Store Registers

This instruction will rotate the selected response store register right, end around.

Format

Label	Command	Argument	Comment
symbol	<u>ROT</u>	<u>rs, a₁±k₁, a₂±k₂</u>	

- Label Any valid symbol or blank.

- Command ROT

- Argument

- • rs The response store register to be rotated.

Valid entries:

X - X response store register
Y - Y response store register
M - M response store register

- • a₁±k₁ The number of end-around bit positions to be rotated. a₁ may be either a constant or a symbol: k₁ is an optional constant modifier. A negative value indicates a left end-around rotate from least-significant bit position toward a more significant bit position. A positive value indicates a right end-around rotate from the most-significant bit position toward a less significant bit position. The absolute value of the rotate constant must be less than the value of the modulus a₂±k₂.

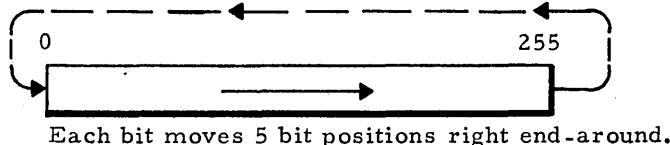
- • a₂±k₂ The modulus to be rotated. This optional term defines the length of the equal sections within the response store register. a₂ may be either a constant or a symbol: k₂ is an optional constant modifier. The value of this term must be a power of 2, such that $1 \leq a_2 \pm k_2 \leq 128$. A default value of 256 is assumed.

Note

If the M response store register is chosen, the X response store register is destroyed.

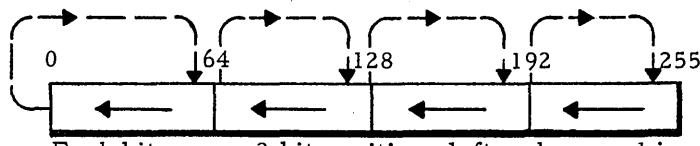
Example 1

ROT Y, 5



Example 2

ROT Y, -2, 64



STORES

This group of associative instructions allows the programmer to store the response store registers or the Common register into an associative memory bit column or an associative memory field respectively. All instructions dealing with response store registers, and/or associative memory fields or bit columns only affect those associative array memory modules enabled via the Array Select register. The response store registers and associative array memory modules disabled via the Array Select register remain unchanged.

<u>Mnemonic</u>	<u>Instructions</u>
S	Store Response Store Into Associative Memory
SM	Store Response Store Masked Into Associative Memory
SN	Store Complement Into Associative Memory
SNM	Store Complement Masked Into Associative Memory
SOR	Store Logical Inclusive OR Into Associative Memory
SORM	Store Logical Inclusive OR, MASKED Into Associative Memory
SORN	Store Logical Inclusive OR, Complemented Into Associative Memory
SORNM	Store Logical Inclusive OR, Complemented, Masked Into Associative Memory
SAND	Store Logical AND Into Associative Memory
SANDM	Store Logical AND Masked Into Associative Memory
SANDN	Store Logical AND Complemented Into Associative Memory
SANDNM	Store Logical AND, Complemented, Masked Into Associative Memory
SC	Store Common Register Into Associative Memory
SCW	Store Common Register Into Associative Word

5

Store Response Store Into Associative Memory

This instruction will store the content of the designated response store register, rs, into the specified bit column of enabled associative memory. The content of the source response store is not affected, and the original content of the bit column is destroyed.

Format	Label	Command	Argument	Comment
	symbol	<u>S</u>	<u>rs,</u> <u>a±k</u> <u>r</u>	

- Label Any valid symbol or blank.
 - Command S
 - Argument Two entries are required. The first entry is the source; the second entry is the destination. As shown, there are two distinct ways of specifying the destination bit column. The brackets are not a part of the argument field terms.
 - • rs The source response store register.

• • rs The source response store register.

Valid entries:

X - X response store
Y - Y response store
M - M response store

- **a_k** 'a' may be a constant or a symbol; k is an optional constant modifier. k is legal only when 'a' is present as a symbol. If 'a' was defined as a field via a DF instruction, the most-significant bit position is the value used. This term represents a destination bit position in all words of enabled associative memory. The value of a_k should be 0 ≤ a_k ≤ 255.

• • r A field pointer register which may be post-incremented or post-decremented. If this form is used, the response store register, rs, is stored indirectly through this register. This register contains the address of the destination bit column.

Valid entries:

FP1	Field Pointer 1
FP1+	Field Pointer 1 with a post-increment
FP1-	Field Pointer 1 with a post-decrement
FP2	Field Pointer 2
FP2+	Field Pointer 2 with a post-increment
FP2-	Field Pointer 2 with a post-decrement
FP3	Field Pointer 3
FP3+	Field Pointer 3 with a post-increment
FP3-	Field Pointer 3 with a post-decrement

S

Example 1

TAG DF 9, 4

•
•
•
•
S Y, TAG

Before		After	
Y	Array Memory Column 9	Y	Array Memory Column 9
256 Bits	0	0	0
	0	1	0
	1	0	1
	1	1	1
	•	•	•
	•	•	•
	•	•	•
	•	•	•
	•	•	•
	•	•	•

Example 2

TAG DF 9, 4

•
•
•
•
S X, TAG -5

Before		After	
X	Array Memory Column 4	X	Array Memory Column 4
256 Bits	0	0	0
	0	1	0
	1	0	1
	1	1	1
	•	•	•
	•	•	•
	•	•	•
	•	•	•
	•	•	•
	•	•	•

Store Response Store Masked Into Associative Memory

This instruction will store the content of the designated response store register, rs, into the specified bit column of enabled associative memory in all words whose M response store bit is set. The content of the source response store is not affected, and the original content of the bit column is destroyed in those words of associative memory whose M response store bit is set.

Format

Label	Command	Argument	Comment
symbol	<u>SM</u>	rs, [a±k] r	

• Label

Any valid symbol or blank.

• Command

SM

• Argument

Two entries are required. The first entry is the source; the second entry is the destination. As shown there are two distinct ways of specifying the destination bit column. The brackets are not a part of the argument field terms.

• • rs

The source response store register.

Valid entries:

X - X response store
Y - Y response store
M - M response store

• • a±k

'a' may be a constant or a symbol: k is an optional constant modifier. k is legal only when 'a' is present as a symbol. If 'a' was defined as a field via a DF instruction, the most-significant bit position is the value used. This term represents a destination bit position in all selected words of enabled associative memory. The value of a±k should be $0 \leq a \pm k \leq 255$.

• • r

A field pointer register which may be post-incremented or post-decremented. If this form is used, the response store register, rs, is stored indirectly through this register. This register contains the address of the destination bit column.

Valid entries:

FP1 Field Pointer 1
FP1+ Field Pointer 1 with a post-increment
FP1- Field Pointer 1 with a post-decrement
FP2 Field Pointer 2
FP2+ Field Pointer 2 with a post-increment
FP2- Field Pointer 2 with a post-decrement
FP3 Field Pointer 3
FP3+ Field Pointer 3 with a post-increment
FP3- Field Pointer 3 with a post-decrement

SM

Example TAG DF 152, 40
 . .
 . .
 . .
SM X, TAG

Before			After		
X	M	Array Memory Col 152	X	M	Array Memory Col 152
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	1	0
0	1	1	0	1	0
1	0	0	1	0	0
1	0	1	1	0	1
1	1	0	1	1	1
1	1	1	1	1	1
.
.
.

SN

Store Complement Into Associative

This instruction will store the one's complement of the value of the designated response store register, rs, into the specified bit column of enabled associative memory. The content of the source response store is not affected, and the original content of the bit column is destroyed.

Format	Label	Command	Argument	Comment
	symbol	<u>SN</u>	<u>rs</u> , $\left[\begin{smallmatrix} a \pm k \\ r \end{smallmatrix} \right]$	

- Label Any valid symbol or blank.
- Command SN
- Argument Two entries are required. The first entry is the source; the second entry is the destination. As shown, there are two distinct ways of specifying the destination bit column. The brackets are not a part of the argument field terms.
- • rs The source response store register.
- Valid entries:
Y - Y response store
M - M response store
- • a±k 'a' may be a constant or a symbol: k is an optional constant modifier. k is legal only when 'a' is present as a symbol. If 'a' was defined as a field via a DF instruction, the most-significant bit position is the value used. This term represents a destination bit position in all words of enabled associative memory. The value of a±k should be $0 \leq a \pm k \leq 255$.
- • r A field pointer register which may be post-incremented or post-decremented. If this form is used, the response store register, rs, is stored indirectly through this register. This register contains the address of the destination bit column.
- Valid entries:

FP1	Field Pointer 1
FP1+	Field Pointer 1 with a post-increment
FP1-	Field Pointer 1 with a post-decrement
FP2	Field Pointer 2
FP2+	Field Pointer 2 with a post-increment
FP2-	Field Pointer 2 with a post-decrement
FP3	Field Pointer 3
FP3+	Field Pointer 3 with a post-increment
FP3-	Field Pointer 3 with a post-decrement

SN

Note

The original content of the X response store register is destroyed when this multiple instruction is executed.

Example

TAG DF 9, 4
· ·
· ·
· ·
SN M, TAG

<u>Before</u>			<u>After</u>		
X	M	Array Memory Col 9	X	M	Array Memory Col 9
1	0	0	d	0	1
1	0	1	e	0	1
0	1	0	s	1	0
0	1	1	t	1	0
·	·	·	r	·	·
·	·	·	o	·	·
·	·	·	y	·	·
			e	·	·
			d	·	·

SNM

Store Complement Masked Into Associative Memory

This instruction will store the one's complement of the value of the Y response store register into the specified bit column of enabled associative memory in all words of associative memory whose M response store bit is set. The content of the Y response store is not affected and the original content of the bit column is destroyed in those words of associative memory whose M response store bit is set.

Format

Label	Command	Argument	Comment
symbol	<u>SNM</u>	$Y_{\left[\begin{array}{c} a \pm k \\ r \end{array} \right]}$	

• Label

Any valid symbol or blank.

• Command

SNM

• Argument

Two entries are required. The first entry is the source; the second entry is the destination. As shown, there are two distinct ways of specifying the destination bit column. The brackets are not a part of the argument field terms.

• • Y

Required and only valid entry.

• • a±k

'a' may be a constant or a symbol; k is an optional constant modifier. k is legal only when 'a' is present as a symbol. If 'a' was defined as a field via a DF instruction, the most-significant bit position is the value used. This term represents a destination bit position in all selected words of enabled associative memory. The value of a±k should be $0 \leq a \pm k \leq 255$.

• • r

A field pointer register which may be post-incremented or post-decremented. If this form is used, the response store register, rs, is stored indirectly through this register. This register contains the address of the destination bit column.

Valid entries:

- FP1 Field Pointer 1
- FP1+ Field Pointer 1 with a post-increment
- FP1- Field Pointer 1 with a post-decrement
- FP2 Field Pointer 2
- FP2+ Field Pointer 2 with a post-increment
- FP2- Field Pointer 2 with a post-decrement
- FP3 Field Pointer 3
- FP3+ Field Pointer 3 with a post-increment
- FP3- Field Pointer 3 with a post-decrement

SNM

Note

The original content of the X response store register is destroyed when this multiple instruction is executed.

Example

TAG DF 152,40
· ·
· ·
SNM Y, TAG

Before				After			
X	Y	M	Array Memory Col. 152	X	Y	M	Array Memory Col. 152
-	0	0	0	d	0	0	0
-	0	0	1	e	0	0	1
-	0	1	0	s	0	1	1
-	0	1	1	t	0	1	1
-	1	0	0	r	1	0	0
-	1	0	1	o	1	0	1
-	1	1	0	y	1	1	0
-	1	1	1	e	1	1	0
·	·	·	·	d	·	·	·
·	·	·	·	·	·	·	·
·	·	·	·	·	·	·	·

SOR

Store Logical Inclusive OR Into Associative Memory

This instruction will logical inclusive OR the contents of the designated response store register, rs, and the bit column of enabled associative memory, and store the resultant value into the bit column. The content of the source response store, rs, is not affected, and the original content of the bit column is destroyed.

Format	Label	Command	Argument	Comment
	symbol	<u>SOR</u>	<u>r</u> _s , [<u>a</u> \pm <u>k</u>] <u>r</u>	

- Label Any valid symbol or blank.
 - Command SOR
 - Argument Two entries are required. The first entry is the source; the second entry is the destination. As shown, there are two distinct ways of specifying the destination bit column. The brackets are not a part of the argument field terms.
 - • rs The source response store register.
 - Valid entries:
 - Y - Y response store
 - M - M response store
 - • a±k 'a' may be a constant or a symbol; k is an optional constant modifier. k is legal only when 'a' is present as a symbol. If 'a' was defined as a field via a DF instruction, the most-significant bit position is the value used. This term represents a destination bit position in all words of enabled associative memory. The value of a±k should be $0 \leq a \pm k \leq 255$.
 - • r A field pointer register which may be post-incremented or post-decremented. If this form is used, the response store register, rs, is stored indirectly through this register. This register contains the address of the destination bit column.
 - Valid entries:
 - FP1 Field Pointer 1
 - FP1+ Field Pointer 1 with a post-increment
 - FP1- Field Pointer 1 with a post decrement
 - FP2 Field Pointer 2
 - FP2+ Field Pointer 2 with a post-increment
 - FP2- Field Pointer 2 with a post-decrement
 - FP3 Field Pointer 3
 - FP3+ Field Pointer 3 with a post-increment
 - FP3- Field Pointer 3 with a post-decrement

SOR

Note

The original content of the X response store register is destroyed when this multiple instruction is executed.

Example

TAG	DF	9, 4
.	.	.
.	.	.
.	.	.
SOR		Y, TAG-1

Before

Array Memory Column 8		
X	Y	
1	0	0
1	0	1
0	1	0
0	1	1
.	.	.
.	.	.
.	.	.

256
Bits

After

Array Memory Column 8		
X	Y	
d	0	0
e	0	1
s	1	1
t	1	1
r	.	.
o	.	.
y	.	.
e		
d		

SORM

Store Logical Inclusive OR, Masked

This instruction will logical inclusive OR the contents of the Y response store register and the designated bit column of enabled associative memory, and store the resultant value into the designated bit column in all words of associative memory whose M response store bit is set. The content of the Y response store is not affected and the original content of the bit column is destroyed in those words of associative memory whose M response store bit is set.

Format	Label	Command	Argument	Comment
	symbol	<u>SORM</u>	<u>Y</u> [<u>a+k</u>] <u>r</u>	

- Label Any valid symbol or blank.
 - Command SORM
 - Argument Two entries are required. The first entry is the source; the second entry is the destination. As shown, there are two distinct ways of specifying the destination bit column. The brackets are not a part of the argument field terms.
 - • Y Required and only valid entry.
 - • a±k 'a' may be a constant or a symbol; k is an optional constant modifier. k is legal only when 'a' is present as a symbol. If 'a' was defined as a field via a DF instruction, the most-significant bit position is the value used. This term represents a destination bit position in all selected words of enabled associative memory. The value of a±k should be $0 \leq a \pm k \leq 255$.
 - • r A field pointer register which may be post-incremented or post-decremented. If this form is used, the response store register, rs, is stored indirectly through this register. This register contains the address of the destination bit column.

Valid entries:

 - FP1 Field Pointer 1
 - FP1+ Field Pointer 1 with a post-increment
 - FP1- Field Pointer 1 with a post-decrement
 - FP2 Field Pointer 2
 - FP2+ Field Pointer 2 with a post-increment
 - FP2- Field Pointer 2 with a post-decrement
 - FP3 Field Pointer 3
 - FP3+ Field Pointer 3 with a post-increment
 - FP3- Field Pointer 3 with a post-decrement

SORM

Note

The original content of the X response store register is destroyed when this multiple instruction is executed.

Example

SORM Y, 200

Before				After			
X	Y	M	Array Memory Col 200	X	Y	M	Array Memory Col. 200
-	0	0	0	d	0	0	0
-	0	0	1	e	0	0	1
-	0	1	0	s	0	1	0
-	0	1	1	t	0	1	1
-	1	0	0	r	1	0	0
-	1	0	1	o	1	0	1
-	1	1	0	y	1	1	1
-	1	1	1	e	1	1	1
.	.	.	.	d	.	.	.
.
.

SORN

Store Logical Inclusive OR Complemented Into Associative Memory

This instruction will logical inclusive OR the one's complement of the contents of the designated response store register, rs, with the specified bit column of enabled associative memory, and store the resultant value into the designated bit column of all words. The content of the source response store is not affected, and the original content of the bit column is destroyed.

Format	Label	Command	Argument	Comment
	symbol	<u>SORN</u>	<u>rs,</u> [<u>a±k</u>] <u>r</u>	

- Label Any valid symbol or blank.
 - Command SORN
 - Argument Two entries are required. The first entry is the source; the second entry is the destination. As shown there are two distinct ways of specifying the destination bit column. The brackets are not a part of the argument field terms.
 - • rs The source response store register.
 - Valid entries:
 - Y - Y response store
 - M - M response store
 - • a±k 'a' may be a constant or a symbol; k is an optional constant modifier. k is legal only when 'a' is present as a symbol. If 'a' was defined as a field via a DF instruction, the most-significant bit position is the value used. This term represents a destination bit position in all words of enabled associative memory. The value of a±k should be $0 \leq a \pm k \leq 255$.
 - • r A field pointer register, which may be post-incremented or post-decremented. If this form is used, the response store register, rs, is stored indirectly through this register. This register contains the address of the destination bit column.

Valid entries:

FP1	Field Pointer 1
FP1+	Field Pointer 1 with a post-increment
FP1-	Field Pointer 1 with a post-decrement
FP2	Field Pointer 2
FP2+	Field Pointer 2 with a post-increment
FP2-	Field Pointer 2 with a post-decrement
FP3	Field Pointer 3
FP3+	Field Pointer 3 with a post-increment
FP3-	Field Pointer 3 with a post-decrement

SORN

Note

The original content of the X response store register is destroyed when this multiple instruction is executed.

Example

SORN		M, X'0'
<u>Before</u>		<u>After</u>
X	M	Array Memory Column 0
1	0	0
1	0	1
0	1	0
0	1	1
.	.	.
.	.	.
.	.	.
256 Bits	d	0
	e	0
	s	1
	t	1
	r	.
	o	.
	y	.
	e	.
	d	.

SORNM

Store Logical Inclusive OR, Complemented, Masked Into Associative Memory

This instruction will logical inclusive OR the one's complement of the contents of the Y response store register with the specified bit column of enabled associative memory. The resultant value is then stored into the designated bit column in all words of associative memory whose M response store bit is set. The content of the Y response store is not affected, and the original content of the bit column is destroyed in those words of associative memory whose M response store bit is set.

Format	Label	Command	Argument	Comment
	symbol	<u>SORNM</u>	<u>Y, [a±k r]</u>	

- Label Any valid symbol or blank.
- Command SORNM
- Argument Two entries are required. The first entry is the source; the second entry is the destination. As shown there are two distinct ways of specifying the destination bit column. The brackets are not a part of the argument field terms.
- • Y Required and only valid entry.
- • a±k 'a' may be a constant or a symbol; k is an optional constant modifier. k is legal only when 'a' is present as a symbol. If 'a' was defined as a field via a DF instruction, the most-significant bit position is the value used. This term represents a destination bit position in all selected words of enabled associative memory. The value of a±k should be $0 \leq a \pm k \leq 255$.
- • r A field pointer register, which may be post-incremented or post-decremented. If this form is used, the response store register, rs, is stored indirectly through this register. This register contains the address of the destination bit column.

Valid entries:

FP1	Field Pointer 1
FP1+	Field Pointer 1 with a post-increment
FP1-	Field Pointer 1 with a post-decrement
FP2	Field Pointer 2
FP2+	Field Pointer 2 with a post-increment
FP2-	Field Pointer 2 with a post-decrement
FP3	Field Pointer 3
FP3+	Field Pointer 3 with a post-increment
FP3-	Field Pointer 3 with a post-decrement

SORNM

Note

The original content of the X response store register is destroyed when this multiple instruction is executed.

Example

SORNM Y, 0

<u>Before</u>				<u>After</u>
X	Y	M	Array Memory Col. 0	X
-	0	0	0	d
-	0	0	1	e
-	0	1	0	s
-	0	1	1	t
-	1	0	0	r
-	1	0	1	o
-	1	1	0	y
-	1	1	1	e
.	.	.	.	d
.
.

SAND

Store Logical AND Into Associative Memory

This instruction will logical AND the contents of the designated response store register, rs, with the specified bit column of enabled associative memory, and store the resultant value into the bit column. The content of the source response store is not affected, and the original content of the bit column is destroyed.

Format

Label	Command	Argument	Comment
symbol	<u>SAND</u>	rs, [_{a±k} _r]	

- Label

Any valid symbol or blank.

- Command

SAND

- Argument

Two entries are required. The first entry is the source; the second entry is the destination. As shown, there are two distinct ways of specifying the destination bit column. The brackets are not a part of the argument field terms.

- • rs

The source response store register.

Valid entries:

Y - Y response store
M - M response store

- • a±k

'a' may be a constant or a symbol; k is an optional constant modifier. k is legal only when 'a' is present as a symbol. If 'a' was defined as a field via a DF instruction, the most-significant bit position is the value used. This term represents a destination bit position in all words of enabled associative memory. The value of a±k should be $0 \leq a \pm k \leq 255$.

- • r

A field pointer register, which may be post-incremented or post-decremented. If this form is used, the response store register, rs, is stored indirectly through this register. This register contains the address of the destination bit column.

Valid entries:

FP1	Field Pointer 1
FP1+	Field Pointer 1 with a post-increment
FP1-	Field Pointer 1 with a post-decrement
FP2	Field Pointer 2
FP2+	Field Pointer 2 with a post-increment
FP2-	Field Pointer 2 with a post-decrement
FP3	Field Pointer 3
FP3+	Field Pointer 3 with a post-increment
FP3-	Field Pointer 3 with a post-decrement

SAND

Note

The original content of the X response store register is destroyed when this multiple instruction is executed.

Example

TAG DF 9,4

• •
• •
• •
SAND Y, TAG

Before

X	Y	Array Memory Column 9
1	0	0
1	0	1
0	1	0
0	1	1
•	•	•
•	•	•
•	•	•

256
Bits

After

X	Y	Array Memory Column 9
d	0	0
e	0	0
s	1	0
t	1	1
r	•	•
o	•	•
y	•	•
e	•	•
d	•	•

SANDM

Store Logical AND Masked Into Associative Memory

This instruction will logical AND the contents of the Y response store register and the specified bit column of enabled associative memory. The resultant value will then be stored into the designated bit column in all words of associative memory whose M response store bit is set. The content of the Y response store is not affected, and the original content of the bit column is destroyed in those words of associative memory whose M response store bit is set.

Format	Label	Command	Argument	Comment
	symbol	SANDM	$\underline{Y}, \left[\begin{array}{c} a \pm k \\ r \end{array} \right]$	

- Label Any valid symbol or blank.
- Command SANDM
- Argument Two entries are required. The first entry is the source; the second entry is the destination. As shown, there are two distinct ways of specifying the destination bit column. The brackets are not a part of the argument field terms.
- • Y Required and the only valid entry.
- • a±k 'a' may be a constant or a symbol; k is an optional constant modifier. k is legal only when 'a' is present as a symbol. If 'a' was defined as a field via a DF instruction, the most-significant bit position is the value used. This term represents a destination bit position in all selected words of enabled associative memory. The value of a±k should be $0 \leq a \pm k \leq 255$.
- • r A field pointer register which may be post-incremented or post-decremented. If this form is used, the response store register, rs, is stored indirectly through this register. This register contains the address of the destination bit column.

Valid entries:

FP1	Field Pointer 1
FP1+	Field Pointer 1 with a post-increment
FP1-	Field Pointer 1 with a post-decrement
FP2	Field Pointer 2
FP2+	Field Pointer 2 with a post-increment
FP2-	Field Pointer 2 with a post-decrement
FP3	Field Pointer 3
FP3+	Field Pointer 3 with a post-increment
FP3-	Field Pointer 3 with a post-decrement

SANDM

Note

The original content of the X response store register is destroyed when this multiple instruction is executed.

Example

SANDM Y, 10

			<u>Before</u>	<u>After</u>
			X Y M	Array Memory Column 10
256 Bits	-	0	0	0
	-	0	0	1
	-	0	1	0
	-	0	1	1
	-	1	0	0
	-	1	0	1
	-	1	1	0
	-	1	1	1



SANDN

Store Logical AND Complemented Into Associative Memory

This instruction will logical AND the one's complement of the contents of the designated response store register, rs, with the specified bit column of enabled associative memory. The resultant value is then stored into the designated bit column in all words of associative memory. The content of the Y response store is not affected and the original content of the bit column is destroyed.

Format	Label	Command	Argument	Comment
	symbol	<u>SANDN</u>	<u>rs,</u> <u>a±k</u> <u>r</u>	

- Label Any valid symbol or blank.
 - Command SANDN
 - Argument Two entries are required. The first entry is the source; the second entry is the destination. As shown, there are two distinct ways of specifying the destination bit column. The brackets are not a part of the argument field terms.
 - • rs The source response store register.
 - Valid entries:
 - Y - Y response store
 - M - M response store
 - • a±k 'a' may be a constant or a symbol; k is an optional constant modifier. k is legal only when 'a' is present as a symbol. If 'a' was defined as a field via a DF instruction, the most-significant bit position is the value used. This term represents a destination bit position in all words of enabled associative memory. The value of a±k should be $0 \leq a \pm k \leq 255$.
 - • r A field pointer register which may be post-incremented or post-decremented. If this form is used, the response store register, rs, is stored indirectly through this register. This register contains the address of the destination bit column.

Valid entries:

FP1	Field Pointer 1
FP1+	Field Pointer 1 with a post-increment
FP1-	Field Pointer 1 with a post-decrement
FP2	Field Pointer 2
FP2+	Field Pointer 2 with a post-increment
FP2-	Field Pointer 2 with a post-decrement
FP3	Field Pointer 3
FP3+	Field Pointer 3 with a post-increment
FP3-	Field Pointer 3 with a post-decrement

SANDN

Note

The original content of the X response store register is destroyed when this multiple instruction is executed.

Example 1

TAG DF 9, 4
· ·
· ·
· ·
SANDN Y, TAG

<u>Before</u>			<u>After</u>		
		Array Memory Column 9			Array Memory Column 9
X	Y		X	Y	
256 Bits	1	0	d	0	0
	1	0	e	0	1
	0	1	s	1	0
	0	1	t	1	0
	·	·	r	·	·
	·	·	o	·	·
	·	·	y	·	·
			e		
			d		

SANDNM

Store Logical AND, Complemented, Masked Into Associative Memory

This instruction will logical AND the one's complement of the contents of the Y response store register with the specified bit column of enabled associative memory. The resultant value is then stored into the designated bit column in all words of associative memory whose M response store bit is set. The content of the Y response store is not affected, and the original content of the bit column is destroyed in those words of associative memory whose M response store bit is set.

Format	Label	Command	Argument	Comment
	symbol	<u>SANDNM</u>	<u>Y,</u> $\left[\begin{smallmatrix} a\pm k \\ r \end{smallmatrix} \right]$	

- Label

Any valid symbol or blank.

- Command

SANDNM

- Argument

Two entries are required. The first entry is the source; the second entry is the destination. As shown, there are two distinct ways of specifying the destination bit column. The brackets are not a part of the argument field terms.

- • Y

Required and only valid entry.

- • a±k

'a' may be a constant or a symbol; k is an optional constant modifier. k is legal only when 'a' is present as a symbol. If 'a' was defined as a field via a DF instruction, the most-significant bit position is the value used. This term represents a destination bit position in all selected words of enabled associative memory. The value of a±k should be $0 \leq a\pm k \leq 255$.

- • r

A field pointer register, which may be post-incremented or post-decremented. If this form is used, the response store register, rs, is loaded indirectly through this register. This register contains the address of the destination bit column.

Valid entries:

FP1	Field Pointer 1
FP1+	Field Pointer 1 with a post-increment
FP1-	Field Pointer 1 with a post-decrement
FP2	Field Pointer 2
FP2+	Field Pointer 2 with a post-increment
FP2-	Field Pointer 2 with a post decrement
FP3	Field Pointer 3
FP3+	Field Pointer 3 with a post-increment
FP3-	Field Pointer 3 with a post-decrement

SANDNM

Note

The original content of the X response store register is destroyed when this multiple instruction is executed.

Example

SANDMN Y, 0

<u>Before</u>				<u>After</u>
X	Y	M	Array Memory Column 0	X
-	0	0	0	d
-	0	0	1	e
-	0	1	0	s
-	0	1	1	t
-	1	0	0	r
-	1	0	1	o
-	1	1	0	y
-	1	1	1	e
.	.	.	.	d
.
.

SC

Store Common Register into Associative Memory

This instruction will store a Common register field, a_1 , into a field, a_2 , of all words of enabled associative memory whose M response store bit is set.

Format	Label	Command	Argument	Comment
	symbol	<u>SC</u>	a_1, a_2	

- Label Any valid symbol or blank.
- Command SC
- Argument Two entries are required. The first entry is the source, a field in the Common register; the second entry is the destination, a field in words of associative memory.
- • a_1, a_2 There are two ways of denoting a field expression:
 - 1) a_1 or a_2 may be in the form
 $b \pm i$
where b must be a symbol, and i is an optional constant modifier. b should have been previously defined in a DF instruction. b represents the most significant bit position and the number of contiguous bits occupied by a field in either the Common register or associative memory. The optional constant modifier, i, modifies only the most-significant bit position.
 - 2) a_1 or a_2 may be in the form
 $(b, i) \pm j$
where b may be a constant or a symbol and represents the most-significant bit position of a field. If b was defined as a field via a previous DF instruction, the most-significant bit position is the value used. i must be a constant and represents the number of contiguous bits occupied by the field. j is an optional constant, modifying only the most-significant bit position of the field.

SC

- Note 1 If the Common register field length is less than the associative memory field length, a W (warning flag) is noted on the listing. In this case, the Common register field will be stored right justified into the associative memory field.
- Note 2 If the Common register field length is greater than the associative memory field length, a T (truncation flag) is noted on the listing. In this case, the most significant bits of the Common register field are truncated.
- Note 3 The content of the X response store register is destroyed. Also, the following field definition registers are used: FP1, FP2, and FL1.

Example

SC (0, 3), (10, 3)

		Array Memory Bit Column		
		10	11	12
256 Bits	M	1	0	1
	1	1	0	1
	1	1	0	1
	0	unchanged		
	0	unchanged		
	1	1	0	1
	1	1	0	1

Common Register

0		31	
1	0	1	-----

SCW

Store Common Register into Associative Word

This instruction will store a Common register field, a_1 , into a field, a_2 , of one word of associative memory whose address is in the link pointer (FP1, FP2). All other words in the associative memory remain unchanged.

Format

Label	Command	Argument	Comment
symbol	SCW	a_1, a_2	

• Label

Any valid symbol or blank.

• Command

SCW

• Argument

Two entries are required. The first entry is the source, a field in the Common register; the second entry is the destination, a field in a word of associative memory.

• a_1, a_2

There are two ways of denoting a field expression:

- 1) a_1 or a_2 may be in the form
 $b\pm i$

where b must be a symbol, and i is an optional constant modifier. b should have been previously defined in a DF instruction. b represents the most-significant bit position and the number of contiguous bits occupied by a field in either the Common register or associative memory. The optional constant modifier, i, modifies only the most-significant bit position.

- 2) a_1 or a_2 may be in the form
 $(b, i)\pm j$

where b may be a constant or a symbol and represents the most-significant bit position of a field. If b was defined as a field via a previous DF instruction, the most-significant bit position is the value used. i must be a constant and represents the number of contiguous bits occupied by the field. j is an optional constant, modifying only the most-significant bit position of the field.

Note 1

The link pointer (FP1 and FP2 registers) must be loaded with the address of the particular word of associative memory prior to execution of the instruction. Loading the link pointer is generally accomplished by the use of the FIND, STEP, or RESVFST instruction.

Note 2

If the Common register field length is less than the associative memory field length, a W (warning flag) is noted on the listing. In this case, the Common register field will be stored right justified into the associative memory field.

Note 3

If the Common register field length is greater than the associative memory field length, a T (truncation flag) is noted on the listing. In this case, the most significant bits of the Common register field are truncated.

Note 4

The content of the X response store register is destroyed. The content of the Y response store register will be destroyed if field-alignment shifting is required.

SCW

Example

FIND

SCW $(0, 3), (10, 3)$
·
·
·

FP1	FP2
ADDR of WORD 3	

Before

X	Y	Array Memory Bit Column		
		10	11	12
1	0			
0	0			
0	0	.	.	.
1	1	.	.	.
0	1	.	.	.
.
.
.
0	0	.	.	.

After

		Array Memory Bit Column		
X	Y	10	11	12
d	d		unchanged	
e	e		unchanged	
s	s		unchanged	
t	t	1	0	1
r	r		unchanged	
o	o	.	.	.
y	y	:	:	:
e	e	.	.	.
d	d		unchanged	

Common Register

0

31

1 0 1 - - - - - - - - - - - - - - -

SEARCHES

These associative instructions allow the programmer to search for a particular set of conditions in associative memory. All instructions dealing with response store registers and/or associative memories affect only those associative array memory modules enabled via the Array Select register. The response store registers and associative array memory modules disabled via the Array Select register remain unchanged. Except for MAXF and MINF, the most-significant bit of all fields is considered to be the sign bit.

<u>Mnemonic</u>	<u>Instructions</u>
FIND	Find the First Bit Set in Y Response Store
STEP	Step to First Y Set and Clear It
RESVFST	Step to First Y Set and Clear All Others
EQC	Equal to Common Register Field
EQF	Equal Fields
NEC	Not Equal To Common Register Field
NEF	Not Equal Fields
GTC	Greater Than Common Register Field
GTF	Greater Than Fields
GEC	Greater Than or Equal To Common Register Field
GEF	Greater Than or Equal Fields
LTC	Less Than Common Register Field
LTF	Less Than Fields
LEC	Less Than or Equal Common Register Field
LEF	Less Than or Equal Fields
MAXF	Maximum Fields
MINF	Minimum Fields

FIND

Find the First Bit Set in Y Response Store

The instruction loads FP1 with the array address of the first array module containing a Y response store register bit set to one. FP2 is then loaded with the bit address of the first Y response store register bit set to one.

Format	Label	Command	Argument	Comment
	symbol	<u>FIND</u>		

- Label Any valid symbol or blank.
- Command FIND
- Argument No entries required.

STEP

Step to First Y Set and Clear It

This instruction loads FP1 with the array address of the first array module containing a Y response store register bit set to one. FP2 is then loaded with the bit address of the first Y response store register bit set to one. This selected first bit will then be cleared to zero.

Format	Label	Command	Argument	Comment
	symbol	<u>STEP</u>		

- Label Any valid symbol or blank.
- Command STEP
- Argument No entries required.

RESVFST**Step to First Y Set and Clear All Others**

This instruction loads FP1 with the array address of the first array module containing a Y response store register bit set to one. FP2 is then loaded with the bit address of the first Y response store register bit set to one. This selected first bit will remain set to one, but all other bits in the Y response store register are cleared to zero.

Format	Label	Command	Argument	Comment
	symbol	<u>RESVFST</u>		

- Label Any valid symbol or blank.
- Command RESVFST
- Argument No entries required.

EQC**Equal to Common Register Field**

For the field a_1 in each word of associative memory, this instruction will set the corresponding Y response store register bit if, and only if, the following is true:

Conditions

- 1) The particular array is enabled in the Array Select register so it may participate in the search.
- 2) The M response store register bit is set for the particular word participating in the search.
- 3) The search criteria is met; array field a_1 is equal to Common register field a_2 .

Format

Label	Command	Argument	Comment
symbol	<u>EQC</u>	a_1, a_2	

- Label Any valid symbol or blank.
- Command EQC
- Argument Two entries are required. The first entry, a_1 , is a field in associative memory; the second entry, a_2 , is a field in the Common register. The lengths of the fields must be equal and greater than one.
- • a_1, a_2 There are two ways of denoting a field expression.

- 1) a_1 or a_2 may be in the form

$b\pm i$

where b must be a symbol, and i is an optional constant modifier. b should have been previously defined in a DF instruction. b represents the most-significant bit position and the number of contiguous bits occupied by a field in either the Common register or associative memory. The optional constant modifier, i, modifies only the most significant bit position.

- 2) a_1 or a_2 may be in the form

$(b, i)\pm j$

where b may be a constant or a symbol and represents the most-significant bit position of a field. If b was defined as a field via a previous DF instruction, the most-significant bit position is the value used. i must be a constant and represents the number of contiguous bits occupied by the field. j is an optional constant, modifying only the most-significant bit position of the field.

Note

Register values after the search:

- 1) FL1 - Zero
- 2) FP1 - Address of the most-significant bit of a_2 .
- 3) FP3 - Address of the most-significant bit of a_1 .

EQF**Equal Fields**

This instruction will set the Y response store register bit for each word of associative memory if, and only if, the following is true:

Conditions

- 1) The particular array is enabled in the Array Select register so it may participate in the search.
- 2) The M response store register bit is set for the particular word participating in the search.
- 3) The search criteria is met; array field a_1 of word i is equal to array field a_2 of word i .

Format

Label	Command	Argument	Comment
symbol	<u>EQF</u>	a_1, a_2	

- Label Any valid symbol or blank.
- Command EQF
- Argument Two entries are required. Both entries represent fields in associative memory that are compared with each other. The lengths of the fields must be equal and greater than one.
- • a_1, a_2 There are two ways of denoting a field expression:
 - 1) a_1 or a_2 may be in the form
 $b\pm i$
where b must be symbol, and i is an optional constant modifier. b should have been previously defined in a DF instruction. b represents the most-significant bit position and the number of contiguous bits occupied by a field in associative memory. The optional constant modifier, i, modifies only the most-significant bit position.
 - 2) a_1 or a_2 may be in the form
 $(b, i)\pm j$
where b may be a constant or a symbol and represents the most-significant bit position of a field. If b was defined as a field via a previous DF instruction, the most-significant bit position is the value used. i must be a constant and represents the number of contiguous bits occupied by the field. j is an optional constant, modifying only the most-significant bit position of the field.

Note**Register values after search:**

- 1) FLL - Zero
- 2) FP1 - Address of the most-significant bit of a_2 .
- 3) FP3 - Address of the most-significant bit of a_1 .

The X response store register is utilized.

NEC

Not Equal To Common Register Field

For the field a_1 in each word of associative memory, this instruction will set the corresponding Y response store register bit if, and only if, the following is true:

Conditions

- 1) The particular array is enabled in the Array Select register so it may participate in the search.
- 2) The M response store register bit is set for the particular word participating in the search.
- 3) The search criteria is met; array field a_1 is not equal to Common register field a_2 .

Format

Label	Command	Argument	Comment
symbol	<u>NEC</u>	a_1, a_2	

• Label

Any valid symbol or blank.

• Command

NEC

• Argument

Two entries are required. The first entry, a_1 , is a field in associative memory; the second entry, a_2 , is a field in the Common register. The lengths of the fields must be equal and greater than one.

• • a_1, a_2

There are two ways of denoting a field expression:

- 1) a_1 or a_2 may be in the form

$b \pm i$

where b must be a symbol, and i is an optional constant modifier. b should have been previously defined in a DF instruction. b represents the most-significant bit position and the number of contiguous bits occupied by a field in either the Common register or associative memory. The optional constant modifier, i, modifies only the most-significant bit position.

- 2) a_1 or a_2 may be in the form

$(b, i) \pm j$

where b may be a constant or a symbol and represents the most-significant bit position of a field. If b was defined as a field via a previous DF instruction, the most-significant bit position is the value used. i must be a constant and represents the number of contiguous bits occupied by the field. j is an optional constant, modifying only the most-significant bit position of the field.

Note

Register values after the search:

- 1) FLL - Zero
- 2) FP1 - Address of the most significant bit of a_2 .
- 3) FP3 - Address of the most significant bit of a_1 .

NEF**Not Equal Fields**

This instruction will set the Y response store register bit for each word of associative memory if, and only if, the following is true:

Conditions

- 1) The particular array is enabled in the Array Select register so it may participate in the search.
- 2) The M response store register bit is set for the particular word participating in the search.
- 3) The search criteria is met; array field a_1 of word i is not equal to array field a_2 of word i .

Format

Label	Command	Argument	Comment
symbol	<u>NEF</u>	a_1, a_2	

- Label Any valid symbol or blank.
- Command NEF
- Argument Two entries are required. Both entries represent fields in associative memory that are compared with each other. The lengths of the fields must be equal and greater than one.
- • a_1, a_2 There are two ways of denoting a field expression:
 - 1) a_1 or a_2 may be in the form
 $b\pm i$
where b must be a symbol, and i is an optional constant modifier. b should have been previously defined in a DF instruction. b represents the most-significant bit position and the number of contiguous bits occupied by a field in associative memory. The optional constant modifier, i, modifies only the most-significant bit position.
 - 2) a_1 or a_2 may be in the form
 $(b, i)\pm j$
where b may be a constant or a symbol and represents the most-significant bit position of a field. If b was defined as a field via a previous DF instruction, the most-significant bit position is the value used. i must be a constant and represents the number of contiguous bits occupied by the field. j is an optional constant, modifying only the most-significant bit position of the field.

Note**Register values after search:**

- 1) FLL - Zero
- 2) FP1 - Address of the most-significant bit of a_2 .
- 3) FP3 - Address of the most-significant bit of a_1 .

GTC**Greater Than Common Register Field**

For the field a_1 in each word of associative memory, this instruction will set the corresponding Y response store register bit if, and only if, the following is true:

Conditions

- 1) The particular array is enabled in the Array Select register so it may participate in the search.
- 2) The M response store register bit is set for the particular word participating in the search.
- 3) The search criteria is met; array field a_1 is greater than Common register field a_2 .

Format

Label	Command	Argument	Comment
symbol	<u>GTC</u>	$a_1 \sqcup a_2$	

- Label Any valid symbol or blank.
- Command GTC
- Argument Two entries are required. The first entry, a_1 , is a field in associative memory; the second entry, a_2 , is a field in the Common register. The lengths of the fields must be equal and greater than one.
- • a_1, a_2 There are two ways of denoting a field expression:
 - 1) a_1 or a_2 may be in the form
 $b\pm i$
where b must be a symbol, and i is an optional constant modifier. b should have been previously defined in a DF instruction. b represents the most-significant bit position and the number of contiguous bits occupied by a field in either the Common register or associative memory. The optional constant modifier, i, modifies only the most-significant bit position.
 - 2) a_1 or a_2 may be in the form
 $(b, i)\pm j$
where b may be a constant or a symbol and represents the most-significant bit position of a field. If b was defined as a field via a previous DF instruction, the most-significant bit position is the value used. i must be a constant and represents the number of contiguous bits occupied by the field. j is an optional constant, modifying only the most-significant bit position of the field.

Note**Register values after the search:**

- 1) FL1 - Zero
- 2) FP1 - Address of the most-significant bit of a_2 .
- 3) FP3 - Address of the most-significant bit of a_1 .

GTF

Greater Than Fields

This instruction will set the Y response store register bit for each word of associative memory if, and only if, the following is true:

Conditions

- 1) The particular array is enabled in the Array Select register so it may participate in the search.
- 2) The M response store register bit is set for the particular word participating in the search.
- 3) The search criteria is met; array field a_1 of word_i is greater than array field a_2 of word_i.

Format

Label	Command	Argument	Comment
symbol	<u>GTF</u>	a_1, a_2	

- Label Any valid symbol or blank.
- Command GTF
- Argument Two entries are required. Both entries represent fields in associative memory that are compared with each other. The lengths of the fields must be equal and greater than one.
- • a_1, a_2 There are two ways of denoting a field expression:
 - 1) a_1 or a_2 may be in the form
 $b\pm i$
where b must be a symbol, and i is an optional constant modifier. b should have been previously defined in a DF instruction. b represents the most-significant bit position and the number of contiguous bits occupied by a field in associative memory. The optional constant modifier, i, modifies only the most-significant bit position.
 - 2) a_1 or a_2 may be in the form
 $(b, i)\pm j$
where b may be a constant or a symbol and represents the most-significant bit position of a field. If b was defined as a field via a previous DF instruction, the most-significant bit position is the value used. i must be a constant and represents the number of contiguous bits occupied by the field. j is an optional constant modifying only the most-significant bit position of the field.

Note

Register values after search:

- 1) FL1 - Zero
- 2) FP1 - Address of the most-significant bit of a_2 .
- 3) FP3 - Address of the most-significant bit of a_1 .

The X response store register is utilized.

GEC

Greater Than or Equal To Common Register Field

For the field a_1 in each word of associative memory, this instruction will set the corresponding Y response store register bit if, and only if, the following is true:

Conditions

- 1) The particular array is enabled in the Array Select register so it may participate in the search.
 - 2) The M response store register bit is set for the particular word participating in the search.
 - 3) The search criteria is met; array field a_1 is greater than or equal to Common register field a_2 .

Format	Label	Command	Argument	Comment
	symbol	<u>GEC</u>	<u>a</u> ₁ , <u>a</u> ₂	

- Label Any valid symbol or blank.
 - Command GEC
 - Argument Two entries are required. The first entry, a_1 , is a field in associative memory; the second entry, a_2 , is a field in the Common register. The lengths of the fields must be equal and greater than one.

• • a_1, a_2 There are two ways of denoting a field expression:

- 1) a_1 or a_2 may be in the form

$b \pm i$

where b must be a symbol, and i is an optional constant modifier. b should have been previously defined in a DF instruction. b represents the most-significant bit position and the number of contiguous bits occupied by a field in either the Common register or associative memory. The optional constant modifier, i, modifies only the most-significant bit position.

- 2) a_1 or a_2 may be in the form

(b, i) \pm j

where b may be a constant or a symbol and represents the most-significant bit position of a field. If b was defined as a field via a previous DF instruction, the most-significant bit position is the value used. i must be a constant and represents the number of contiguous bits occupied by the field. j is an optional constant, modifying only the most-significant bit position of the field.

Note

Register values after the search:

- 1) FL1 - Zero
 - 2) FP1 - Address of the most-significant bit of a_2 .
 - 3) FP3 - Address of the most-significant bit of a_1 .

GEF**Greater Than or Equal Fields**

This instruction will set the Y response store register bit for each word of associative memory if, and only if, the following is true:

Conditions

- 1) The particular array is enabled in the Array Select register so it may participate in the search.
- 2) The M response store register bit is set for the particular word participating in the search.
- 3) The search criteria is met; array field a_1 of word i is greater than or equal to array field a_2 of word i .

Format

Label	Command	Argument	Comment
symbol	<u>GEF</u>	a_1, a_2	

- Label Any valid symbol or blank.
- Command GEF
- Argument Two entries are required. Both entries represent fields in associative memory that are compared with each other. The lengths of the fields must be equal and greater than one.
- • a_1, a_2 There are two ways of denoting a field expression:
 - 1) a_1 or a_2 may be in the form
 $b\pm i$ where b must be a symbol, and i is an optional constant modifier. b should have been previously defined in a DF instruction. b represents the most-significant bit position and the number of contiguous bits occupied by a field in associative memory. The optional constant modifier, i, modifies only the most-significant bit position.
 - 2) a_1 or a_2 may be in the form
 $(b, i)\pm j$ where b may be a constant or a symbol and represents the most-significant bit position of a field. If b was defined as a field via a previous DF instruction, the most-significant bit position is the value used. i must be a constant and represents the number of contiguous bits occupied by the field. j is an optional constant, modifying only the most-significant bit position of the field.

Note

Register values after search:

- 1) FL1 - Zero
- 2) FP1 - Address of the most-significant bit of a_2 .
- 3) FP3 - Address of the most-significant bit of a_1 .

The X response store register is utilized.

LTC

Less Than Common Register Field

For the field a_1 in each word of associative memory, this instruction will set the corresponding Y response store register bit if, and only if, the following is true:

Conditions

- 1) The particular array is enabled in the Array Select register so it may participate in the search.
- 2) The M response store register bit is set for the particular word participating in the search.
- 3) The search criteria is met; array field a_1 is less than Common register field a_2 .

Format

Label	Command	Argument	Comment
symbol	LTC	a_1, a_2	

- Label Any valid symbol or blank.
- Command LTC
- Argument Two entries are required. The first entry, a_1 , is a field in associative memory; the second entry, a_2 , is a field in the Common register. The lengths of the fields must be equal and greater than one.
- • a_1, a_2 There are two ways of denoting a field expression:
 - 1) a_1 or a_2 may be in the form
 $b\pm i$
where b must be a symbol, and i is an optional constant modifier. b should have been previously defined in a DF instruction. b represents the most-significant bit position and the number of contiguous bits occupied by a field in either the Common register or associative memory. The optional constant modifier, i, modifies only the most-significant bit position.
 - 2) a_1 or a_2 may be in the form
 $(b, i)\pm j$
where b may be a constant or a symbol and represents the most-significant bit position of a field. If b was defined as a field via a previous DF instruction, the most-significant bit position is the value used. i must be a constant and represents the number of contiguous bits occupied by the field. j is an optional constant, modifying only the most-significant bit position of the field.

Note

Register values after the search:

- 1) FLL - Zero
- 2) FP1 - Address of the most-significant bit of a_2 .
- 3) FP3 - Address of the most-significant bit of a_1 .

LTF

Less Than Fields

This instruction will set the Y response store register bit for each word of associative memory if, and only if, the following is true:

Conditions

- 1) The particular array is enabled in the Array Select register so it may participate in the search.
- 2) The M response store register bit is set for the particular word participating in the search.
- 3) The search criteria is met; array field a_1 of word i is less than array field a_2 of word i .

Format

Label	Command	Argument	Comment
symbol	<u>LTF</u>	a_1, a_2	

- Label Any valid symbol or blank.
- Command LTF
- Argument Two entries are required. Both entries represent fields in associative memory that are compared with each other. The lengths of the fields must be equal and greater than one.
- • a_1, a_2 There are two ways of denoting a field expression:
 - 1) a_1 or a_2 may be in the form
 $b\pm i$
where b must be a symbol, and i is an optional constant modifier. b should have been previously defined in a DF instruction. b represents the most-significant bit position and the number of contiguous bits occupied by a field in associative memory. The optional constant modifier, i, modifies only the most-significant bit position.
 - 2) a_1 or a_2 may be in the form
 $(b, i)\pm j$
where b may be a constant or a symbol and represents the most-significant bit position of a field. If b was defined as a field via a previous DF instruction, the most-significant bit position is the value used. i must be a constant and represents the number of contiguous bits occupied by the field. j is an optional constant, modifying only the most-significant bit position of the field.

Note

Register values after search:

- 1) FL1 - Zero
- 2) FP1 - Address of the most-significant bit of a_2 .
- 3) FP3 - Address of the most-significant bit of a_1 .

The X response store register is utilized.

LEC

Less Than or Equal Common Register Field

For the field a_1 in each word of associative memory, this instruction will set the corresponding Y response store register bit if, and only if, the following is true:

Conditions

- 1) The particular array is enabled in the Array Select register so it may participate in the search.
- 2) The M response store register bit is set for the particular word participating in the search.
- 3) The search criteria is met; array field a_1 is less than or equal to Common register field a_2 .

Format

Label	Command	Argument	Comment
symbol	<u>LEC</u>	a_1, a_2	

• Label

Any valid symbol or blank.

• Command

LEC

• Argument

Two entries are required. The first entry, a_1 , is a field in associative memory; the second entry, a_2 , is a field in the Common register. The lengths of the fields must be equal and greater than one.

• • a_1, a_2

There are two ways of denoting a field expression:

- 1) a_1 or a_2 may be in the form

$b\pm i$

where b must be a symbol, and i is an optional constant modifier. b should have been previously defined in a DF instruction. b represents the most-significant bit position and the number of contiguous bits occupied by a field in either the Common register or associative memory. The optional constant modifier, i, modifies only the most-significant bit position.

- 2) a_1 or a_2 may be in the form

$(b, i)\pm j$

where b may be a constant or a symbol and represents the most-significant bit position of a field. If b was defined as a field via a previous DF instruction, the most-significant bit position is the value used. i must be a constant and represents the number of contiguous bits occupied by the field. j is an optional constant, modifying only the most-significant bit position of the field.

Note

Register values after the search:

- 1) FLL - Zero
- 2) FP1 - Address of the most-significant bit of a_2 .
- 3) FP3 - Address of the most-significant bit of a_1 .

LEF**Less Than or Equal Fields**

This instruction will set the Y response store register bit for each word of associative memory if, and only if, the following is true:

Conditions

- 1) The particular array is enabled in the Array Select register so it may participate in the search.
- 2) The M response store register bit is set for the particular word participating in the search.
- 3) The search criteria is met; array field a_1 of word i is less than or equal to array field a_2 of word i .

Format

Label	Command	Argument	Comment
symbol	<u>LEF</u>	a_1, a_2	

• Label

Any valid symbol or blank.

• Command

LEF

• Argument

Two entries are required. Both entries represent fields in associative memory that are compared with each other. The lengths of the fields must be equal and greater than one.

• • a_1, a_2

There are two ways of denoting a field expression:

- 1) a_1 or a_2 may be in the form

$b\pm i$

where b must be a symbol, and i is an optional constant modifier. b should have been previously defined in a DF instruction. b represents the most-significant bit position and the number of contiguous bits occupied by a field in associative memory. The optional constant modifier, i, modifies only the most-significant bit position.

- 2) a_1 or a_2 may be in the form

$(b, i)\pm j$

where b may be a constant or a symbol and represents the most-significant bit position of a field. If b was defined as a field via a previous DF instruction, the most-significant bit position is the value used. i must be a constant and represents the number of contiguous bits occupied by the field. j is an optional constant, modifying only the most-significant bit position of the field.

Note

Register values after search:

- 1) FLL - Zero
- 2) FP1 - Address of the most-significant bit of a_2 .
- 3) FP3 - Address of the most-significant bit of a_1 .

MAXF**Maximum Fields**

This instruction will compare a field of those words of the associative memory whose M response store bit is set. The Y response store will be set for the word(s) containing the field with the maximum (greatest) unsigned value. The Y response store for all other words will be cleared to zero.

Format

Label	Command	Argument	Comment
symbol	<u>MAXF</u>	a	

- Label Any valid symbol or blank.

- Command MAXF

- Argument One entry is required.

• • a There are two ways of denoting a field expression:

- 1) 'a' may be in the form

b±i

where b must be a symbol, and i is an optional constant modifier. b should have been previously defined in a DF instruction. b represents the most-significant bit position and the number of contiguous bits occupied by a field in associative memory. The optional constant modifier, i, modifies only the most-significant bit position.

- 2) 'a' may be in the form

(b, i)±j

where b may be a constant or a symbol and represents the most significant bit position of a field. If b was defined as a field via a previous DF instruction, the most-significant bit position is the value used. i must be a constant and represents the number of contiguous bits occupied by the field. j is an optional constant, modifying only the most-significant bit position of the field.

Note

Register values after search:

- 1) FLL - Zero
- 2) FP3 - Address of the least-significant bit of 'a'.

The X response store register is utilized.

MINF

Minimum Fields

This instruction will compare a field of those words of associative memory whose M response store bit is set. The Y response store will be set for the word(s) containing the field with the minimum (least) unsigned value. The Y response store for all other words will be cleared to zero.

Format

Label	Command	Argument	Comment
symbol	<u>MINF</u>	<u>a</u>	

- Label Any valid symbol or blank.

- Command MINF

- Argument One entry is required.

- • a There are two ways of denoting a field expression:

- 1) 'a' may be in the form

$b \pm i$

where b must be a symbol, and i is an optional constant modifier. b should have been previously defined in a DF instruction. b represents the most-significant bit position and the number of contiguous bits occupied by a field in associative memroy. The optional constant modifier, i, modifies only the most-significant bit position.

- 2) 'a' may be in the form

$(b, i) \pm j$

where b may be a constant or a symbol and represents the most significant bit position of a field. If b was defined as a field via a previous DF instruction, the most-significant bit position is the value used. i must be a constant and represents the number of contiguous bits occupied by the field. j is an optional constant, modifying only the most-significant bit position of the field.

Note

Register values after the search:

- 1) FLL - Zero
- 2) FP3 - Address of the least-significant bit of 'a'.

The X response store register is utilized.

MOVES

This group of associative instructions allows the programmer to move an array memory field to another array memory field within the same word of associative memory.

This group of instructions will operate only on those associative array memory modules (including response store registers) enabled via the Array Select register. Also, only those words within enabled associative array memory modules whose M response store register bit is set will participate in the execution of the instructions in this group. The most significant bit of all fields is considered to be the sign bit.

<u>Mnemonic</u>	<u>Instructions</u>
MVF	Move Field
MVCF	Move the One's Complement of a Field
MVNF	Move the Negative of a Field
MVAF	Move the Absolute Value of a Field
INCF	Move Field with Increment
DECF	Move Field with Decrement

MVF

Move Field

This instruction will move the contents of field a_1 into field a_2 within the same word for each word of enabled associative memory whose M response store bit is set. The content of the source field is not affected unless overlaid by the destination field. The original content of the destination field is destroyed.

Format	Label	Command	Argument	Comment
	symbol	<u>MVF</u>	a_1, a_2	

- Label Any valid symbol or blank.
- Command MVF
- Argument Two entries are required. The first entry is the source; the second entry is the destination. Both entries represent fields within the same word of associative memory.
- • a_1, a_2 There are two ways of denoting a field expression:

- 1) a_1 or a_2 may be in the form

$b \pm i$

where b must be a symbol, and i is an optional constant modifier. b should have been previously defined in a DF instruction. b represents the most-significant bit position and the number of contiguous bits occupied by a field in associative memory. The optional constant modifier, i , modifies only the most-significant bit position.

- 2) a_1 or a_2 may be in the form

$(b, i) \pm j$

where b may be a constant or a symbol and represents the most-significant bit position of a field. If b was defined as a field via a previous DF instruction, the most-significant bit position is the value used. i must be a constant and represents the number of contiguous bits occupied by the field. j is an optional constant, modifying only the most-significant bit position of the field.

Note

The X response store, FP1, FP3, and FL1 registers are used by this instruction.

MVF

Example

MVF

(2, 3), (10, 3)

		Before								After					
		Array Memory Bit Column								Array Memory Bit Column					
M		2	3	4	10	11	12			2	3	4	10	11	12
256 Bits	0	0	0	0	1	1	0	0	0	0	0	0	1	1	0
	0	0	0	1	1	0	1	0	0	0	1	1	0	0	1
	1	0	1	0	1	0	0	1	0	1	0	1	0	1	0
	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1

MVCF

Move the One's Complement of a Field

This instruction will move the one's complement of the contents of field a_1 into field a_2 within the same word for each word of enabled associative memory whose M response store bit is set. The content of the source field is not affected unless overlaid by the destination field. The original content of the destination field is destroyed.

Format	Label	Command	Argument	Comment
	symbol	MVCF	a_1, a_2	

- Label Any valid symbol or blank.
- Command MVCF
- Argument Two entries are required. The first entry is the source; the second entry is the destination. Both entries represent fields within the same word of associative memory.
- • a_1, a_2 There are two ways of denoting a field expression:
 - 1) a_1 or a_2 may be in the form
 $b\pm i$
 where b must be a symbol, and i is an optional constant modifier. b should have been previously defined in a DF instruction. b represents the most-significant bit position and the number of contiguous bits occupied by a field in associative memory. The optional constant modifier, i, modifies only the most-significant bit position.
 - 2) a_1 or a_2 may be in the form
 $(b, i)\pm j$
 where b may be a constant or a symbol and represents the most-significant bit position of a field. If b was defined as a field via a previous DF instruction, the most-significant bit position is the value used. i must be a constant and represents the number of contiguous bits occupied by the field. j is an optional constant, modifying only the most-significant bit position of the field.

Note

The X response store, FP1, FP3, and FL1 registers are used by this instruction.

MVCF

Example

MVCF $(2, 3), (20, 3)$

After Execution

M	Array Memory Bit Column					
	2	3	4	20	21	22
256 Bits	1	0	0	0	1	1
	1	0	0	1	1	0
	1	0	1	0	1	0
	1	0	1	1	0	0
	1	1	0	0	0	1
	1	1	0	1	0	0
	1	1	1	0	0	0
	1	1	1	1	0	0

MVNF

Move the Negative of a Field

This instruction will move the two's complement of the contents of field a_1 into field a_2 within the same word for each word of enabled associative memory whose M response store bit is set. The content of the source field is not affected unless overlaid by the destination field. The original content of the destination field is destroyed.

Format	Label	Command	Argument	Comment
	symbol	<u>MVNF</u>	<u>a_1, a_2</u>	

- Label Any valid symbol or blank.
- Command MVNF
- Argument Two entries are required. The first entry is the source; the second entry is the destination. Both entries represent fields within the same word of associative memory.
- • a_1, a_2 There are two ways of denoting a field expression:

- 1) a_1 or a_2 may be in the form

$b\pm i$

where b must be a symbol, and i is an optional constant modifier. b should have been previously defined in a DF instruction. b represents the most-significant bit position and the number of contiguous bits occupied by a field in associative memory. The optional constant modifier, i , modifies only the most-significant bit position.

- 2) a_1 or a_2 may be in the form

$(b, i)\pm j$

where b may be a constant or a symbol and represents the most-significant bit position of a field. If b was defined as a field via a previous DF instruction, the most-significant bit position is the value used. i must be a constant and represents the number of contiguous bits occupied by the field. j is an optional constant, modifying only the most-significant bit position of the field.

Note

The X response store, Y response store, FP1, FP3, and FL1 registers are used by this instruction.

MVNF

Example

MVNF $(5, 3), (15, 3)$

After Execution

M	Array Memory Bit Column					
	5	6	7	15	16	17
256 Bits	1	0	0	0	0	0
	1	0	0	1	1	1
	1	0	1	0	1	1
	1	0	1	1	0	1
	1	1	0	0	1	0*
	1	1	0	1	0	1
	1	1	1	0	1	0
	1	1	1	1	0	1

.

* An overflow condition is set in the response store registers.

MVAF**Move the Absolute Value of a Field**

This instruction will move the absolute value of the contents of field a_1 into field a_2 within the same word for each word of enabled associative memory whose M response store bit is set. The content of the source field is not affected unless overlaid by the destination field. The original content of the destination field is destroyed.

Format	Label	Command	Argument	Comment
	symbol	<u>MVAF</u>	a_1, a_2	

- Label Any valid symbol or blank.
- Command MVAF
- Argument Two entries are required. The first entry is the source; the second entry is the destination. Both entries represent fields within the same word of associative memory.
- • a_1, a_2 There are two ways of denoting a field expression:
 - 1) a_1 or a_2 may be in the form
 $b\pm i$
 where b must be a symbol, and i is an optional constant modifier. b should have been previously defined in a DF instruction. b represents the most-significant bit position and the number of contiguous bits occupied by a field in associative memory. The optional constant modifier, i, modifies only the most-significant bit position.
 - 2) a_1 or a_2 may be in the form
 $(b, i)\pm j$
 where b may be a constant or a symbol and represents the most-significant bit position of a field. If b was defined as a field via a previous DF instruction, the most-significant bit position is the value used. i must be a constant and represents the number of contiguous bits occupied by the field. j is an optional constant, modifying only the most-significant bit position of the field.

Note

The X response store, Y response store, FP1, FP3, and FL1 registers are used by this instruction.

MVAF

Example

MVAF $(2, 3), (10, 3)$

M	Array Memory Bit Column					
	2	3	4	10	11	12
256 Bits	1	0	0	0	0	0
	1	0	0	1	0	0
	1	0	1	0	0	1
	1	0	1	1	0	1
	1	1	0	0	1	0*
	1	1	0	1	0	1
	1	1	1	0	0	1
	1	1	1	1	0	1
.						
.						
.						

* An overflow condition is set in the response store registers.

INCE

Move Field with Increment

This instruction will add one to the value of field a_1 and store the incremented value into field a_2 within the same word for each word of enabled associative memory whose M response store bit is set. The content of the source field is not affected unless overlaid by the destination field. The original content of the destination field is destroyed.

Format	Label	Command	Argument	Comment
	symbol	<u>INCF</u>	<u>a</u> 1, <u>a</u> 2	

- Label Any valid symbol or blank.

- Command INCF

- Argument Two entries are required. The first entry is the source; the second entry is the destination. Both entries represent fields within the same word of associative memory.

- • a₁, a₂ There are two ways of denoting a field expression:

- 1) a_1 or a_2 may be in the form

where b must be a symbol, and i is an optional constant modifier. b should have been previously defined in a DF instruction. b represents the most-significant bit position and the number of contiguous bits occupied by a field in associative memory. The optional constant modifier, i, modifies only the most-significant bit position.

- 2) a_1 or a_2 may be in the form

$(b, i) \pm j$

where b may be a constant or a symbol and represents the most-significant bit position of a field. If b was defined as a field via a previous DF instruction, the most-significant bit position is the value used. i must be a constant and represents the number of contiguous bits occupied by the field. j is an optional constant, modifying only the most-significant bit position of the field.

- ## Note

X response store, Y response store, FL1, FP2, FP3, and R0 registers are used by this instruction.

INCF

Example

INCF (2, 3), (10, 3)

After Execution

M	Array Memory Bit Column					
	2	3	4	10	11	12
256 Bits	1	0	0	0	0	1
	1	0	0	1	0	0
	1	0	1	0	0	1
	1	0	1	1	1	0
	1	1	0	0	1	1
	1	1	0	1	1	0
	1	1	1	0	1	1
	1	1	1	1	0	0

- * An overflow condition is set in the response store registers.

DECF

Move Field with Decrement

This instruction will subtract one from the value of field a_1 and store the decremented value into field a_2 within the same word for each word of enabled associative memory whose M response store bit is set. The content of the source field is not affected unless overlaid by the destination field. The original content of the destination field is destroyed.

Format	Label	Command	Argument	Comment
	sybmol	<u>DECF</u>	a_1, a_2	

- Label Any valid symbol or blank.
- Command DECF
- Argument Two entries are required. The first entry is the source; the second entry is the destination. Both entries represent fields within the same word of associative memory.
- • a_1, a_2 There are two ways of denoting a field expression:
 - 1) a_1 or a_2 may be in the form
 $b \pm i$
 where b must be a symbol, and i is an optional constant modifier. b should have been previously defined in a DF instruction. b represents the most-significant bit position and the number of contiguous bits occupied by a field in associative memory. The optional constant modifier, i, modifies only the most-significant bit position.
 - 2) a_1 or a_2 may be in the form
 $(b, i) \pm j$
 where b may be a constant or a symbol and represents the most-significant bit position of a field. If b was defined as a field via a previous DF instruction, the most-significant bit position is the value used. i must be a constant and represents the number of contiguous bits occupied by the field. j is an optional constant, modifying only the most-significant bit position of the field.

Note

X response store, Y response store, FL1, FP2, FP3, and R0 registers are used by this instruction.

DECF

Example

DECF (2, 3), (10, 3)

M	Array Memory Bit Column					
	2	3	4	10	11	12
256 Bits	1	0	0	0	1	1
	1	0	0	1	0	0
	1	0	1	0	0	1
	1	0	1	1	0	1
	1	1	0	0	1	1*
	1	1	0	1	0	0
	1	1	1	0	1	0
	1	1	1	1	1	0
.	
.	
.	

* An overflow condition is set in the response store registers.

ARITHMETICS

This group of associative instructions allows the programmer to perform arithmetic operations between associative memory fields, and between a Common register field with an associative memory field.

This group of instructions will operate only on those associative array memory modules (including response store registers) enabled via the Array Select register. Also, only those words within enabled associative array memory modules whose M response store register bit is set will participate in the execution of the instructions in this group. The most significant bit of all fields is considered to be the sign bit.

<u>Mnemonic</u>	<u>Instructions</u>
ADC	Add Common Register to Field
ADF	Add Field to Field
SBC	Subtract Common Register from Field
SBF	Subtract Field from Field
MPC	Multiply Field by Common Register
MPF	Multiply Field by Field
DVF	Divide Field by Field

ADC

Add Common Register to Field

This instruction will add field a_2 of the Common register (addend) to field a_1 of word_i in associative memory (augend) and then store the resultant sum into field a_3 of word_i. Only those words of associative memory whose M response store bit is set will participate in this instruction. The original content of the addend field a_2 is undisturbed. The content of the augend field a_1 will be undisturbed unless the sum field a_3 overlays it.

Format	Label	Command	Argument	Comment
	symbol	<u>ADC</u>	a_1, a_2, a_3	

- Label Any valid symbol or blank.

- Command ADC

- Argument Three entries are required. The first entry represents the augend and is a field in associative memory. The second entry is the addend and is a field in the Common register. These two fields are added together and the sum is stored into the third entry, a field in associative memory.

• • a_1, a_2, a_3 There are two ways of denoting a field expression:

- 1) $a_1, a_2,$ or a_3 may be in the form
 $b \pm i$

where b must be a symbol, and i is an optional constant modifier. b should have been previously defined in a DF instruction. b represents the most-significant bit position and the number of contiguous bits occupied by a field in either the Common register or associative memory. The optional constant modifier, i, modifies only the most-significant bit position of the field.

- 2) $a_1, a_2,$ or a_3 may be in the form
 $(b, i) \pm j$

where b may be a constant or a symbol and represents the most-significant bit position of a field. If b was defined as a field via a previous DF instruction, the most-significant bit position is the value used. i must be a constant and represents the number of contiguous bits occupied by the field. j is an optional constant, modifying only the most-significant bit position of the field.

Note X response store, Y response store, FL1, FP1, FP2, FP3, and R0 registers are used by this instruction.

ADC

Example

ADC {0, 3}, {5, 3}, {11, 3}

After Execution

M	Array Memory Bit Column					
	0	1	2	11	12	13
256 Words	1	0	0	0	0	1
	1	0	1	0	0	1
	1	1	0	0	1	0
	1	1	1	0	1	1
	1	0	1	1	1	0*
	1	1	0	1	1	0
	1	1	1	1	0	0

*An overflow condition will be set in the response store registers.

Common Register

0 4 5 6 7 8

31



ADF

Add Field to Field

This instruction will add field a_1 of word_i to field a_2 of word_i and store the resultant sum into field a_3 of word_i. Only those words of associative memory whose M response store bit is set will participate in this instruction. The original content of the source fields a_1 and a_2 will remain undisturbed unless overlaid by the sum field a_3 .

Format	Label	Command	Argument	Comment
	symbol	<u>ADF</u>	a_1, a_2, a_3	

- Label Any valid symbol or blank.
- Command ADF
- Argument Three entries are required. Each represents a field in associative memory. The first field a_1 represents the augend; the second field a_2 represents the addend; and the third field a_3 represents the sum.
- • a_1, a_2, a_3 There are two ways of denoting a field expression:

- 1) $a_1, a_2,$ or a_3 may be in the form

$b \pm i$

where b must be a symbol, and i is an optional constant modifier. b should have been previously defined in a DF instruction. b represents the most-significant bit position and the number of contiguous bits occupied by a field in associative memory. The optional constant modifier, i, modifies only the most-significant bit position.

- 2) $a_1, a_2,$ or a_3 may be in the form

$(b, i) \pm j$

where b may be a constant or a symbol and represents the most-significant bit position of a field. If b was defined as a field via a previous DF instruction, the most-significant bit position is the value used. i must be a constant and represents the number of contiguous bits occupied by the field. j is an optional constant, modifying only the most-significant bit position of the field.

Note

X response store, Y response store, FL1, FP1, FP2, FP3, and R0 registers are used by this instruction.

ADF

Example

ADF (0, 3), (5, 3), (10, 3)

After Execution

M	Array Memory Bit Column											
	0	1	2	5	6	7	10	11	12			
256 Words	1	0	0	0	0	0	0	0	0			
	1	0	1	0	0	0	1	0	1	1		
	1	1	0	0	0	1	0	1	0			
	1	1	1	0	0	1	1	0	0	1		
	1	0	0	1	1	0	0	1	0	1		
	1	0	1	1	0	1	0	1	0	0		
	1	1	0	1	1	1	0	0	1	1*		
	1	1	1	1	1	1	1	1	0			

* An overflow condition will be set in the response store registers.

SBC

Subtract Common Register from Field

This instruction will subtract field a_2 of the Common register (subtrahend) from field a_1 of word_i in associative memory (minuend) and then store the resultant difference into field a_3 of word_i. Only those words of associative memory whose M response store bit is set will participate in this instruction. The original content of the Common register field is undisturbed. The content of field a_1 will be undisturbed unless the difference field a_3 overlays it.

Format	Label	Command	Argument	Comment
	symbol	<u>SBC</u>	a_1, a_2, a_3	

- Label Any valid symbol or blank.
- Command SBC
- Argument Three entries are required. The first entry represents a field in associative memory and is the minuend. The second entry represents a field in the Common register and is the subtrahend. The third entry represents a field in associative memory and is the difference of the minuend minus the subtrahend.
- • a_1, a_2, a_3 There are two ways of denoting a field expression:
 - 1) $a_1, a_2,$ or a_3 may be in the form
 $b\pm i$
 where b must be a symbol, and i is an optional constant modifier. b should have been previously defined in a DF instruction. b represents the most-significant bit position and the number of contiguous bits occupied by a field in either the Common register or associative memory. The optional constant modifier, i , modifies only the most-significant bit position.
 - 2) $a_1, a_2,$ or a_3 may be in the form
 $(b, i)\pm j$
 where b may be a constant or a symbol and represents the most-significant bit position of a field. If b was defined as a field via a previous DF instruction, the most-significant bit position is the value used. i must be a constant and represents the number of contiguous bits occupied by the field. j is an optional constant, modifying only the most-significant bit position of the field.

Note

X response store, Y response store, FL1, FP1, FP2, FP3, and R0 registers are used by this instruction.

SBC

Example

SBC (5, 3), (8, 3), (11, 3)

After Execution

M	Array Memory Bit Column					
	5	6	7	11	12	13
256 Words	1	0	0	1	1	1
	1	0	1	0	0	0
	1	0	1	1	0	1
	1	1	0	0	0	1
	1	1	0	1	0	1
	1	1	1	0	0	0
	1	1	1	1	0	1
	1	0	0	0	1	0

*An overflow condition will be set in the response store registers.

Common Register

0 7 8 9 10 11

31



SBF

Subtract Field from Field

This instruction will subtract field a_2 of word_i from field a_1 of word_i and store the resultant difference into field a_3 of word_i. Only those words of associative memory whose M response store bit is set will participate in this instruction. The original content of the source fields a_1 and a_2 will remain undisturbed unless overlaid by the difference field a_3 .

Format	Label	Command	Argument	Comment
	symbol	<u>SBF</u>	$\underline{a}_1 \underline{a}_2 \underline{a}_3$	

- Label Any valid symbol or blank.
- Command SBF
- Argument Three entries are required. Each entry represents a field in associative memory. The first field, a_1 , represents the minuend; the second field, a_2 , represents the subtrahend; and the third field, a_3 , represents the difference.
- • a_1, a_2, a_3 There are two ways of denoting a field expression:
 - 1) a_1, a_2 , or a_3 may be in the form
 $b \pm i$
 where b must be a symbol, and i is an optional constant modifier. b should have been previously defined in a DF instruction. b represents the most-significant bit position and the number of contiguous bits occupied by a field in associative memory. The optional constant modifier, i, modifies only the most-significant bit position.
 - 2) a_1, a_2 , or a_3 may be in the form
 $(b, i) \pm j$
 where b may be a constant or a symbol and represents the most-significant bit position of a field. If b was defined as a field via a previous DF instruction, the most-significant bit position is the value used. i must be a constant and represents the number of contiguous bits occupied by the field. j is an optional constant, modifying only the most-significant bit position of the field.

Note

X response store, Y response store, FL1, FP1, FP2, FP3, and R0 registers are used by this instruction.

SBF

Example

SBF (5, 3), (8, 3), (11, 3)

After Execution

M	Array Memory Bit Column								
	5	6	7	8	9	10	11	12	13
256 Words	1	0	0	0	0	0	0	0	0
	1	0	0	0	0	1	1	1	1
	1	0	0	0	1	1	1	0	1
	1	1	1	0	0	1	0	0	0
	1	0	1	0	0	0	1	0	1
	1	1	1	1	1	1	0	0	0
	1	1	0	1	0	1	1	0	*
	1	1	1	1	1	0	0	0	1

*An overflow condition will be set in the response store registers.

MPC

Multiply Field by Common Register

This instruction will multiply associative memory field a_1 of word_i (multiplicand) by field a_2 of the Common register (multiplier) and store the product into associative memory field a_3 of word_i. Only those words in associative memory whose M response store bit is set will participate in the multiplication. The original content of the multiplier field a_2 and the multiplicand field is undisturbed i.e., the product field a_3 must not overlay the multiplicand field a_1 .

Format	Label	Command	Argument	Comment
	symbol	<u>MPC</u>	a_1, a_2, a_3	

- Label Any valid symbol or blank.
- Command MPC
- Argument Three entries are required. The first entry represents a field in associative memory and is the multiplicand. The second entry represents a field in the Common register and is the multiplier. The third entry represents a field in associative memory and is the product. The product field width must equal the sum of the multiplier and multiplicand field widths.
- • a_1, a_2, a_3
 - 1) $a_1, a_2, \text{ or } a_3$ may be in the form
 $b\pm i$
 where b must be a symbol, and i is an optional constant modifier. b should have been previously defined in a DF instruction. b represents the most-significant bit position and the number of contiguous bits occupied by a field in either the Common register or associative memory. The optional constant modifier, i, modifies only the most-significant bit position.
 - 2) $a_1, a_2, \text{ or } a_3$ may be in the form
 $(b, i)\pm j$
 where b may be a constant or a symbol and represents the most-significant bit position of a field. If b was defined as a field via a previous DF instruction, the most-significant bit position is the value used. i must be a constant and represents the number of contiguous bits occupied by the field. j is an optional constant, modifying only the most-significant bit position of the field.

Note

X response store, Y response store, FL1, FP1, FP2, FP3, FL2, FPE, and R0 registers are used by this instruction.

MPC

Example

MPC (0, 3), (5, 3), (8, 6)

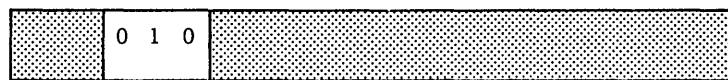
After Execution

M	Array Memory Bit Column									
	0	1	2	8	9	10	11	12	13	
256 Words	1	0	0	1	0	0	0	0	1	0
	1	0	1	0	0	0	0	1	0	0
	1	0	1	1	0	0	0	1	1	0
	1	1	0	0	1	1	1	0	0	0
	1	1	0	1	1	1	0	1	0	
	1	1	1	0	1	1	1	1	0	
	1	1	1	1	1	1	1	1	0	
	
	

Common Register

0 4 5 6 7 8

31



MPF

Multiply Field by Field

This instruction will multiply field a_1 of word_i by field a_2 of word_i, and store the resultant product into field a_3 of word_i. Only those words of the associative memory whose M response store bit is set will participate in this instruction. The original content of the multiplicand field a_1 must remain intact, i.e., it cannot be overlaid by the product field a_3 . The original content of the multiplier field a_2 may be overlaid by the product field a_3 .

Format	Label	Command	Argument	Comment
	symbol	<u>MPF</u>	<u>a</u> ₁ , <u>a</u> ₂ , <u>a</u> ₃	

- Label Any valid symbol or blank.

- Command MPF

- Argument Three entries are required. The first entry represents a field in associative memory and is the multiplicand. The second entry represents a field in the associative memory and is the multiplier. The third entry represents a field in associative memory and is the product. The product field must equal the width of the sum of the multiplier and multiplicand fields.

- • a₁, a₂, a₃ There are two ways of denoting a field expression:

- 1) a_1 , a_2 , or a_3 may be in the form

$b \pm i$

where b must be a symbol, and i is an optional constant modifier. b should have been previously defined in a DF instruction. b represents the most-significant bit position and the number of contiguous bits occupied by a field in associative memory. The optional constant modifier, i, modifies only the most-significant bit position.

- 2) a_1 , a_2 , or a_3 may be in the form

$(b, i) \pm j$

where *b* may be a constant or a symbol and represents the most-significant bit position of a field. If *b* was defined as a field via a previous DF instruction, the most-significant bit position is the value used. *i* must be a constant and represents the number of contiguous bits occupied by the field. *j* is an optional constant, modifying only the most-significant bit position of the field.

- ## Note

X response store, Y response store, FL1, FP1, FP2, FP3, FL2, FPE, and R0 registers are used by this instruction.

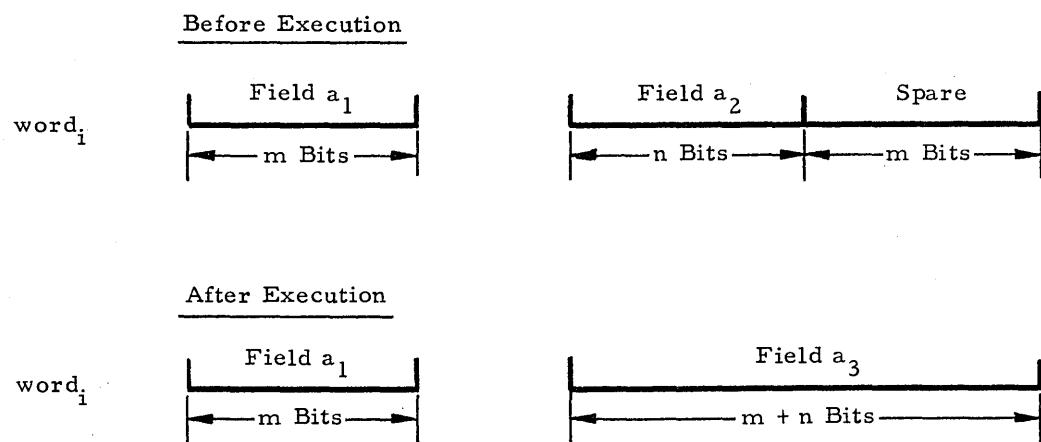
Example 1

MPF (0, 2), (2, 3), (5, 5)

		Array Memory Bit Column								
		M	0 1	2 3 4	5 6 7 8 9					
256 Words	1		1 0	0 0 0	0 0 0 0 0					
	1		1 1	0 0 1	1 1 1 1 1					
	1		1 1	0 1 0	1 1 1 1 0					
	1		0 1	0 1 1	0 0 0 1 1					
	1		1 0	1 0 0	0 1 0 0 0					
	1		1 1	1 0 1	0 0 0 1 1					
	1		1 0	1 1 0	0 0 1 0 0					
	1		0 1	1 1 1	1 1 1 1 1					
					
.					
.					

Example 2

Overlaying the multiplier field a_2 must be handled carefully by the programmer. Array memory storage may be condensed (minimum bit positions) in the following manner if there are at least m "spare" bits to the right of the multiplier field as shown:



The number of spare bits must be equal to the length of the multiplicand field a_1 .

DVF

Divide Field by Field

This instruction will divide field a_1 of word_i by field a_2 of word_j. Only those words of associative array whose M response store bit is set will participate in the divide instruction. The quotient and the remainder are stored into field a_3 of word_i, with the remainder being right justified and having the same length and sign as the divisor, a_2 . The quotient is stored adjacent to the remainder and must have a length of 2 or more. The contents of the divisor must not be overlaid by the quotient-remainder field a_3 .

Overflow Check

Unlike other arithmetic routines, DVF does not check for overflow unless specifically requested in the command field. When requested, the overflow check is made prior to performing the divide. The associative memory words where overflow would occur will have their M response store bit cleared to zero and therefore will not participate in the divide. After the divide, the M is restored, and the possible overflow condition is recorded in the response store registers.

Format

Label	Command	Argument	Comment
symbol	DVF, b	a_1, a_2, a_3	

- Label

Any valid symbol or blank.

- Command

DVF

- • b

b may be a constant, a symbol, or a symbol plus or minus an optional constant modifier. If b was defined as a field via a DF instruction, the most-significant bit position is the value used. This term represents a scratch bit column position in all words of enabled associative arrays and is used to save the original content of the M response store register. The value of b should be $0 \leq b \leq 255$.

- Argument

Three entries are required. Each represents a field in associative memory. The first field a_1 represents the dividend; the second field a_2 represents the divisor; and the third field a_3 represents the quotient-remainder. The field length of the quotient-remainder field must be at least two bit positions longer than the divisor field and at least one bit position longer than the dividend.

- • a_1, a_2, a_3

There are two ways of denoting a field expression:

- 1) a_1, a_2 , or a_3 may be in the form

$b \pm i$

where b must be a symbol, and i is an optional constant modifier. b should have been previously defined in a DF instruction. b represents the most-significant bit position and the number of contiguous bits occupied by a field in associative memory. The optional constant modifier, i, modifies only the most-significant bit position.

- • a_1, a_2, a_3
(cont)

2) a_1, a_2 , or a_3 may be in the form
 $(b, i)\pm j$

where b may be a constant or a symbol and represents the most-significant bit position of a field. If b was defined as a field via a previous DF instruction, the most-significant bit position is the value used. i must be a constant and represents the number of contiguous bits occupied by the field. j is an optional constant, modifying only the most-significant bit position of the field.

Overlapping

The most efficient field layout is overlapping. This technique can save execution time as well as memory. The least-significant bit of the dividend and quotient-remainder fields should have the same address, and the length of field a_3 must be at least one bit column wider than a_1 . The address of the least-significant bit of a_3 and a_1 must be the same to reduce execution time.

Note

X response store, Y response store, FL1, FP1, FP2, FP3, FL2, FPE, and BL registers are used by this instruction.

Example 1

Dividend and divisor not affected:

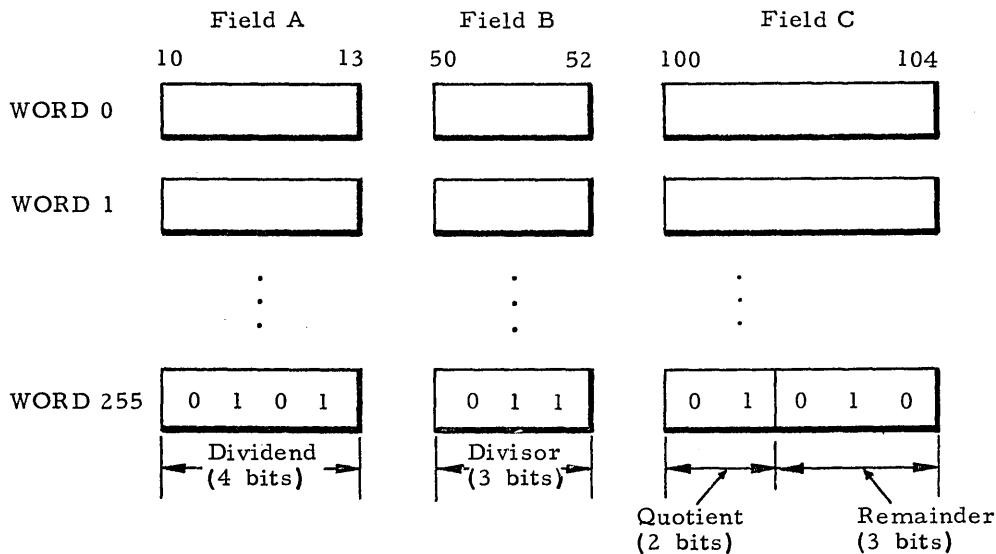
A	DF	10, 4
B	DF	50, 3
C	DF	100, 5

.

.

.

DVF A, B, C



DVF

Example 2

Most efficient memory layout:

CO EQU 4 OVERFLOW SCRATCH BIT

DIVISOR DF 5, 5 DIVISOR

DIVIDEND DF 100, 10 DIVIDEND

QUOTREM DF 99, 11 QUOTIENT, REMAINDER

DVF, CO DIVIDEND, DIVISOR, QUOTREM

In this example overflow will occur in a word if the quotient requires more than 6 bits.

Example 3

Most efficient memory layout with no overflow condition

DIVISOR DF 5, 5 DIVISOR

DIVIDEND DF 100, 10 DIVIDEND

QUOTREM DF 95,15 QUOTIENT, REMAINDER

DVF DIVIDEND, DIVISOR, QUOTREM

**CONTROL
AND
TEST**

This group of instructions allows the programmer to control and test the AP control.

<u>Mnemonic</u>	<u>Instructions</u>
INT	Interrupt Control and Test
ILOCK	Interlock Control and Test
WAIT	Deactivate the AP

INT

Interrupt Control and Test

This instruction will generate an interrupt and/or interrogate the current state of an interrupt according to the value of the argument field expression $a_2^{\pm k_2}$. The interrupt number is denoted by the expression $a_1^{\pm k_1}$.

Format	Label	Command	Argument	Comment
	symbol	<u>INT</u> , $a_1^{\pm k_1}$	$a_2^{\pm k_2}$	

- Label Any valid symbol or blank

- Command INT

- • $a_1^{\pm k_1}$ a_1 may be either a constant or a symbol whose value may be optionally modified by plus or minus the constant k_1 .

- • • Associative Processor Interrupts

Valid Entries:	
Interrupt Number	Vector Address in Bulk Core
X'1'	X'8001'
X'1'	X'8002'
.	.
.	.
X'F'	X'800F'

- • • Sequential Processor Interrupts

Valid Entries:	
Interrupt Number	Vector Address in Sequential Processor
O'300'	O'300'
O'304'	O'304'
O'310'	O'310'
.	.
.	.
O'334'	O'334'

- Argument

- • $a_2^{\pm k_2}$ a_2 may be either a constant or a symbol whose value may be optionally modified by plus or minus the constant k_2 .

Legal Values for $a_2^{\pm k_2}$:

- 0 Unconditionally disable the interrupt.
- 1 Skip the next instruction if interrupt is enabled, and then unconditionally disable the interrupt
- 2 Skip the next instruction if interrupt is disabled, and then unconditionally disable the interrupt
- 3 Unconditionally skip the next instruction and then disable the interrupt

INT

Legal Values for $a_2 \pm k_2$: (cont)

- 5 Skip the next instruction if the interrupt is enabled.
- 6 Skip the next instruction if the interrupt is disabled.
- 7 Unconditionally skip the next instruction.
- 8 Unconditional complement of current state.
- 9 Skip the next instruction if interrupt is enabled, and then unconditionally complement current state.
- 10 Skip the next instruction if interrupt is disabled, and then unconditionally complement current state.
- 11 Unconditionally skip the next instruction, and then unconditionally complement current state.
- 12 Unconditionally enable the interrupt.
- 13 Skip the next instruction if the interrupt is enabled, and then unconditionally enable the interrupt.
- 14 Skip the next instruction if the interrupt is disabled, and then unconditionally enable the interrupt.
- 15 Unconditionally skip the next instruction and then enable the interrupt.

ILOCK

Interlock Control and Test

This instruction will set or reset the specified interlock number $a_1 \pm k_1$ and/or interrogate the current state of this interlock number according to the value of the expression $a_2 \pm k_2$.

Format

Label	Command	Argument	Comment
symbol	<u>ILOCK</u> , $a_1 \pm k_1$	$a_2 \pm k_2$	

- Label Any valid symbol or blank
 - Command ILOCK
 - • $a_1 \pm k_1$ a_1 may be either a constant or a symbol whose value may be optionally modified by plus or minus the constant k_1 . The value of $a_1 \pm k_1$ must be in the range $0 \leq a_1 \pm k_1 \leq 63$. These interlocks have no predetermined meaning. The programmer can assign any meaning to any interlock.
 - Argument
 - • $a_2 \pm k_2$ a_2 may be either a constant or a symbol whose value may be optionally modified by plus or minus the constant k_2 . The value of $a_2 \pm k_2$ determines the action taken on the specified interlock number $a_1 \pm k_1$.
- Legal Values for $a_2 \pm k_2$:
- 0 Unconditionally reset the interlock.
 - 1 Skip the next instruction if set, and then unconditionally reset the interlock number.
 - 2 Skip the next instruction if reset, and then unconditionally reset the interlock number.
 - 3 Unconditionally skip the next instruction and then reset the interlock number.
 - 4 No operation.
 - 5 Skip if the interlock number is set.
 - 6 Skip if the interlock number is reset.
 - 7 Unconditionally skip the next instruction.
 - 8 Unconditionally complement current state.
 - 9 Skip the next instruction if set, and then unconditionally complement current state.
 - 10 Skip the next instruction if reset, and then unconditionally complement current state.
 - 11 Unconditionally skip the next instruction and then unconditionally complement current state.
 - 12 Unconditionally set the interlock number.
 - 13 Skip the next instruction if the interlock is set, and then unconditionally set the interlock number.
 - 14 Skip the next instruction if the interlock is reset, and then unconditionally set the interlock number.
 - 15 Unconditionally skip the next instruction, and then set the interlock number.

WAIT

Deactivate the AP

This instruction will cause the associative processor to go inactive.

Format

Label	Command	Argument	Comment
symbol	<u>WAIT</u>		

- Label Any valid symbol or blank.
- Command WAIT
- Argument No entry is required.

**PAGER
INSTRUCTIONS**

These instructions allow the programmer to utilize the page memories.

<u>Mnemonic</u>	<u>Instructions</u>
STR TSG	Start Segment
END SG	End Segment
MVSG	Move a Page Segment
MVSGI	Move a Page Segment Immediate
PAGER	Pager Control

STR TSG**Start Segment**

This instruction marks the beginning of a page segment by reinitializing the Execution Location Counter as specified in the Command Field.

Format

Label	Command	Argument	Comment
symbol	<u>STRTSG</u> , a±k		

- Label Any valid symbol or blank. This will be the name of the following segment.
- Command STRTSG
- • a±k 'a' may be either a constant or a symbol whose value may be optionally modified by plus or minus the constant k. Moreover, 'a' may be one of the following special symbols: PAGE0, PAGE1, PAGE2. The value of the expression a±k initializes the Execution Location Counter and represents where succeeding assembled APPLE instructions are to be loaded and then executed.
- Argument No entries required.

ENDSG**End Segment**

This instruction marks the end of a page segment.

Format

Label	Command	Argument	Comment
symbol	<u>ENDSG</u>		

- Label Any valid symbol or blank.
- Command ENDSG
- Argument No entries required.

Note

Nested STRTSG—ENDSG pairs are illegal.

Move a Page Segment

This instruction will command the Pager to move a segment of instructions referenced by the Memory Address $a_2 \pm k_2$ if the Pager is not busy with a previous move. If the Pager is busy, AP Control will wait until the previous move is completed before initiating this move.

Format	Label	Command	Argument	Comment
	symbol	<u>MVSG, $a_1 \pm k_1$</u>	<u>$a_2 \pm k_2$</u>	

- Label Any valid symbol or blank.

- ## • Command MVSG

- • $a_1 \pm k_1$ a_1 may be either a constant or a symbol whose value may be optionally modified by plus or minus the constant k_1 . Moreover, a_1 may be one of the following special symbols: PAGE0, PAGE1, PAGE2. This term corresponds to the same term in the STR TSG mnemonic. Its value is used to tell the Pager where to begin storing the program segment.

- Argument One term is required.

- $a_2^{\pm k_2}$ a_2 may be either a constant or a symbol whose value may be optionally modified by plus or minus the constant k_2 . The value of this term should reference the address of a STRTSG mnemonic in bulk core.

MVSGI**Move a Page Segment Immediately**

This instruction will command the Pager to move a segment of instructions referenced by $a_2 \pm k_2$ to the Page Memory address $a_1 \pm k_1$ immediately. If the AP Control encounters the MVSGI instruction while the Pager is busy with a previous move, it will interrupt the Pager and initiate the new move immediately. The remainder of the previous move is forgotten. If the Pager is not busy when the AP control encounters an MVSGI instruction, it acts like an MVSG instruction.

Format	Label	Command	Argument	Comment
	symbol	<u>MVSGI</u> , $a_1 \pm k_1$	$a_2 \pm k_2$	

- Label Any valid symbol or blank.

- Command MVSGI

- • $a_1 \pm k_1$
 a_1 may be either a constant or a symbol whose value may be optionally modified by plus or minus the constant k_1 . Moreover, a_1 may be one of the following special symbols: PAGE0, PAGE1, PAGE2. This term corresponds to the same term in the STRTSG mnemonic. Its value is used to tell the Pager where to begin storing the program segment.

- Argument One term is required.

- • $a_2 \pm k_2$
 a_2 may be either a constant or a symbol whose value may be optionally modified by plus or minus the constant k_2 . The value of this term should reference the address of a STRTSG mnemonic in bulk core.

PAGER**Pager Control**

This instruction will command and/or interrogate the state of the Pager.
The Pager can be considered to be on (busy) or off (not busy).

Format

Label	Command	Argument	Comment
symbol	<u>PAGER</u>	<u>a±k</u>	

- Label Any valid symbol or blank.
- Command PAGER
- Argument One entry is required.
- • a±k 'a' may be either a constant or a symbol whose value may be optionally modified by plus or minus the constant k.

Legal Values of a±k:

- 0 Unconditionally turn Pager off.
- 1 Skip the next instruction if Pager is on, and then unconditionally turn Pager off.
- 2 Skip the next instruction if Pager is off, and then unconditionally turn Pager off.
- 3 Unconditionally skip the next instruction and then turn Pager off.
- 4 No operation.
- 5 Skip the next instruction if Pager is on.
- 6 Skip the next instruction if Pager is off.
- 7 Unconditionally skip the next instruction.
- 8 Unconditionally complement current state.
- 9 Skip the next instruction if Pager is on, and then unconditionally complement current state.
- 10 Skip the next instruction if Pager is off, and then unconditionally complement current state.
- 11 Unconditionally skip the next instruction, and then unconditionally complement current state.
- 12 Unconditionally turn Pager on.
- 13 Skip the next instruction if Pager is on, and then unconditionally turn Pager on.
- 14 Skip the next instruction if Pager is off, then unconditionally turn Pager on.
- 15 Unconditionally skip the next instruction, and then turn Pager on.

CHAPTER 3

SUPERVISOR CALLS

INTRODUCTION

STARAN Program Supervisor (SPS) provides services to supplement the APPLE language such as managing input/output, handling errors, and controlling STARAN processors. This chapter describes all services available to APPLE programs executed by the associative processor.

SPS consists of two program modules which are resident in the sequential control memory at execution time. Module zero (SPS0) manages the sequential controller and its associated peripherals. Module one (SPS1) manages the remaining STARAN processors. SPS0 and SPS1 together manage the entire stand-alone configuration; that is, the sequential controller and its I/O devices, the associative processor, the Pager, the Parallel I/O unit* and additional Custom I/O, as implemented for a specific installation.

The main purpose of SPS is to make input/output operations possible. A programmer can use the supervisor through the APPLE SVC (Supervisor Call) mnemonic by specifying the particular service desired, a buffer, and some I/O device.

SLOT NUMBERS

The convention is not to refer to devices directly, but to the slots assigned them (a slot is similar to a logical unit, see figure 3-1). An I/O request is made on a slot which, in turn, refers to a device. (Slots 0, 1, and 2 are anchored to the keyboard, teleprinter, and high-speed reader. While it is necessary to refer to these slots, any attempt to reassign them will cause an error message to be printed at execution time.) Slot assignments are recorded in the Device Assignment Table (DAT) (figure 3-1). More than one slot may be assigned to a device, but only one device can be attached to a slot. Additional devices will be added to satisfy requirements of specific customers (for example - disk units, magnetic tape units, remote terminals, etc).

STARAN registers, associative memory, and control memory are referenced in SPS calls as though they were devices. They are given negative device codes which may be used to attach them to slots (see figure 3-1). SPS provides for data transfer between these associative processor elements, or between them and any peripheral device.

* Parallel I/O unit is an optional STARAN feature.

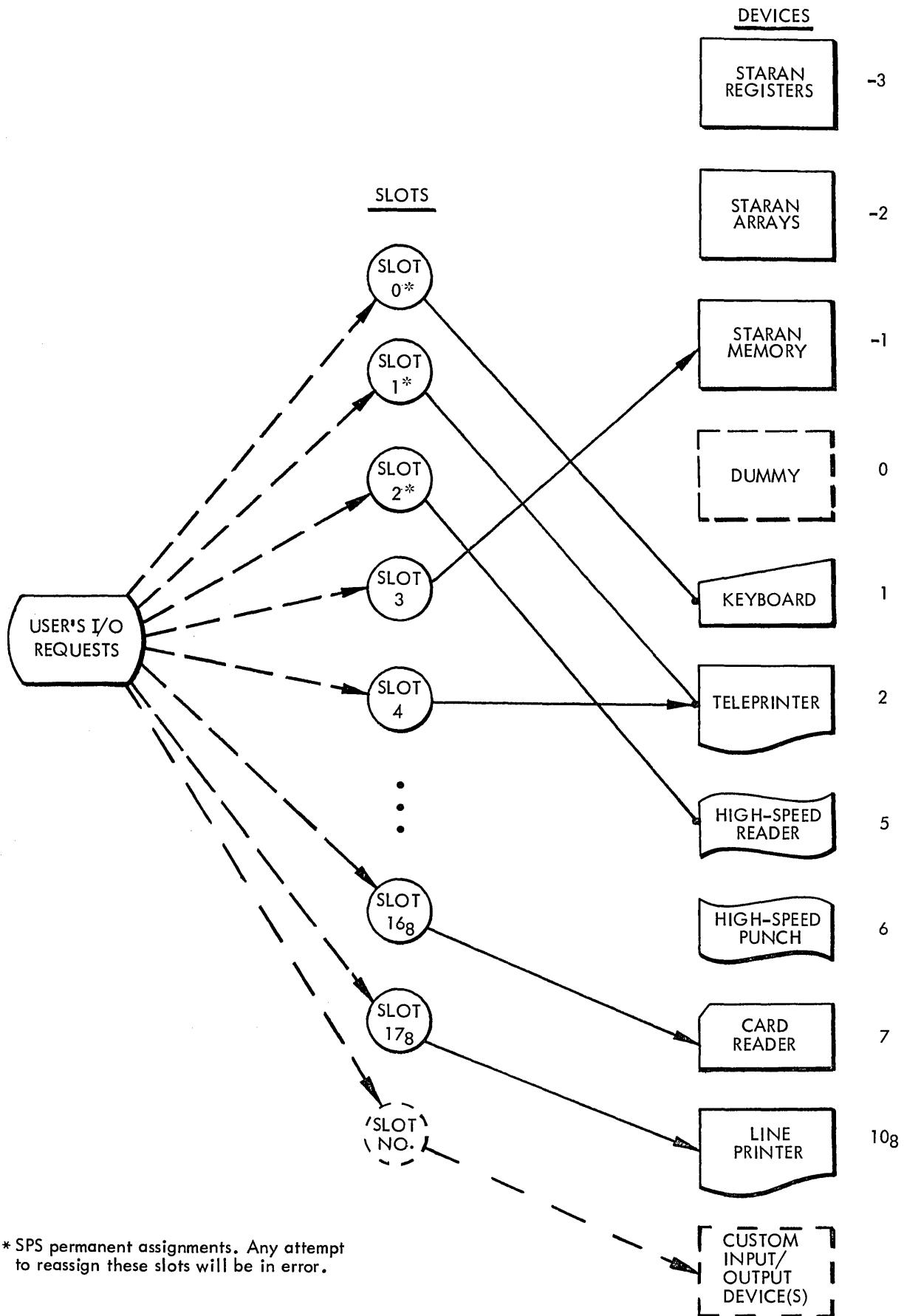


Figure 3-1. Device Assignment Table (DAT)

**INSTRUCTION
DESCRIPTION**

This section of the manual is concerned with the description of two mnemonics and their possible variations. Their basic format is:

**BUFFER
PSEUDO-OP
FORMAT**

Label	Command	Argument	Comment
symbol	<u>BUFFER</u>	<u>a₁</u> , <u>a₂</u> , <u>a₃</u> , a ₄ , a ₅ , a ₆ , a ₇	

and

**SUPERVISOR
CALL
FORMAT**

Label	Command	Argument	Comment
symbol	<u>SVC</u>	a ₁ , a ₂ , a ₃	

Note

The terms in the argument field for these mnemonics will be described in more detail later.

BUFFER

All I/O functions included in this basic manual, except Parallel I/O, require a buffer area (usually either in the HSDB or in BULK core memory) to contain the data as they are input or output. The purpose of the BUFFER mnemonic is to create for the SPS I/O routine required buffer-area header information describing in more detail the exact nature of the intended I/O process. For example, data input from the high-speed paper tape unit may be in the form of ASCII characters (formatted or unformatted) or pure binary values (formatted or unformatted). These cases will be described in more detail later. In other words there is more than one way to input or output data on a given device, and the purpose of the BUFFER pseudo-op is to fully describe the desired method.

SVC

There are currently twelve different variations (SPS services) of the SVC mnemonic. The services will be expanded for future requirements as necessary for Customized Input/Output features. Some of the variations require a corresponding BUFFER mnemonic counterpart to fully describe the nature of the I/O operation to the system. The twelve SVC functions are:

SPS
SERVICES
OR
CALLS

Source Statement Format

<u>Function</u>	<u>Label</u>	<u>Com- mand</u>	<u>Argument</u>	<u>Comment</u>
Attach	symbol	SVC	1, slot-number, device-code-address	
Read	symbol	SVC	9, slot-number, buffer-address	
Write	symbol	SVC	10, slot-number, buffer-address	
Reset	symbol	SVC	2	
Free	symbol	SVC	5, slot-number	
Exit	symbol	SVC	7	
Restart	symbol	SVC	8	
Timer	symbol	SVC	13, timer-number, interrupt number, time-value	
Int	symbol	SVC	14, interrupt-number	
I Setup	symbol	SVC	15, interrupt-number, status, done-address	
Pager Control	symbol	SVC	18, command, start-address	
PI/O Control	symbol	SVC	19, command, start-address	

Each of the above SVC instructions will be described separately in the following pages. Note the distinguishing feature of each SVC instruction is the first term in the argument field. All argument field terms shown are required. Each term in the argument field may be in the form

a±k

where 'a' may be either a constant or a symbol optionally modified by plus or minus the constant k.

ATTACH

Attach an SPS Slot Number to an I/O Device

This function allows the programmer to assign SPS slot numbers to different I/O devices at execution time (see figure 3-1). Note that slot numbers 0, 1, and 2 always refer to the keyboard, teleprinter, and high-speed reader, respectively. These slot numbers may not be reassigned to different devices.

Format	Label	Command	Argument	Comment
	symbol	<u>SVC</u>	<u>1, slot-number, device-code-address</u>	

- Label Any valid symbol or blank.
- Command SVC
- Argument The argument field consists of three entries.
 - • 1 The value of the first entry (must be equal to one) denotes an attach function.
 - • Slot Number The value of the second entry (must be a value between 0 and 17_8) denotes one of the SPS slot numbers (see figure 3-1).
 - • Device-Code-Address The third entry represents an address (must be in either HSDB or in BULK core memory) which contains a device code value.

<u>Value (Octal)</u>	<u>Device</u>
-3	STARAN Registers
-2	STARAN Associative Memory
-1	STARAN Control Memory
0	Dummy
1	Keyboard (KBD)
2	Teleprinter (TTY)
3	Low-Speed Reader (LSR)
4	Low-Speed Punch (LSP)
5	High-Speed Reader (HSR)
6	High-Speed Punch (HSP)
7	Card Reader (CDR)
10	Line Printer (LPT)

STARAN
Special
Device
Codes
For
ATTACH
Function

Note that STARAN storage is given device codes. To allow access to parts of STARAN not directly addressable, the memory elements are formally treated as devices. SVC calls utilizing STARAN device codes require corresponding specially formatted BUFFER pseudo-op mnemonics.

• STARAN
Control
Memory

When STARAN control memory is referenced by an SVC instruction, it should be for the purpose of a memory-to-memory transfer of a block of instructions or data within STARAN Control Memory, which consists of the Page, HSDB, and Bulk core memories.

For an SVC instruction referencing a slot assigned to device -1 (STARAN Control Memory) the form of the corresponding BUFFER mnemonic is:

• • Format	Label	Command	Argument	Comment
	symbol	<u>BUFFER</u>	<u>address, 0, byte-count</u>	

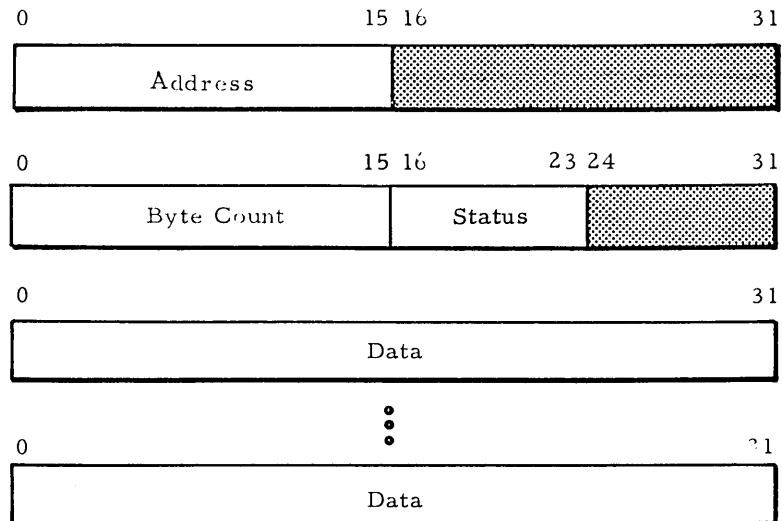
• • • Label Any valid symbol or blank.

• • • Command BUFFER

• • • Argument The argument field consists of three terms. The value of the first term represents any STARAN control memory address and is the destination address. The value of the second term must be zero. The value of the third term represents the byte-count of the number of bytes of data following the BUFFER pseudo-op involved in the memory-to-memory transfer.

The above BUFFER pseudo-op will generate the following two 32-bit words of memory:

• • • BUFFER
Format
For
Device - 1



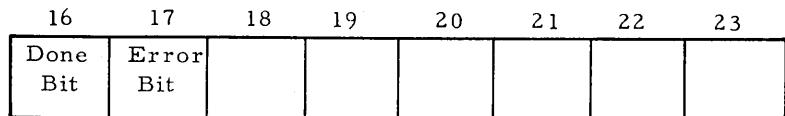
As shown the upper half of the first word will contain the value of the first item of the argument field. The upper half of the second word will contain the value of the number of bytes of data involved in the memory-to-memory transfer. The bytes of data must be contained in the words of memory immediately following the BUFFER pseudo-op mnemonic as shown.

• • • Byte
Count

This value represents the actual number of bytes of data to be moved to or from the buffer area that follows.

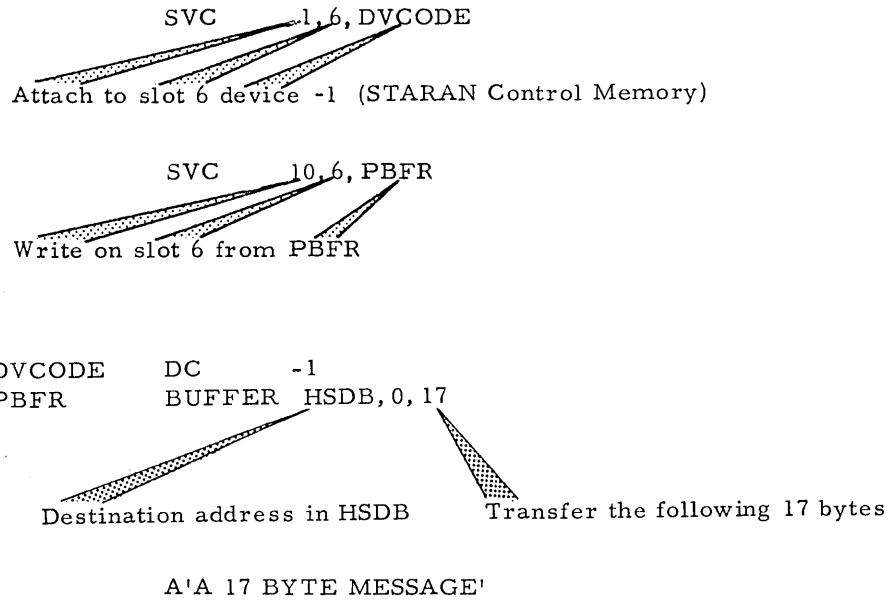
• • • Status
Byte

Bits of the Status Byte shown in bits 16-23 of the second word set by SPS upon completion of the memory-to-memory transfer are bits 16 and 17:



The Done bit will be set to a value of one by SPS upon completion of the operation. The ERROR bit will be set to a value of one by SPS if an error occurred during the operation. If an error occurred before the memory-to-memory transfer was completed, the Byte Count value in the upper half of the second word will contain the actual number of bytes of data transferred by SPS before the operation was terminated. Otherwise the Byte Count value will remain intact.

• • Example



Execution of the above two SVC instructions will result in the memory-to-memory transfer of the above 17 bytes of ASCII characters to the HSDB.

• STARAN
Associative
Memory

When STARAN Associative Memory is referenced by an SVC instruction, it should be for the purpose of a memory-to-memory transfer between associative memory and a buffer located either in the HSDB or Bulk core memories. For an SVC instruction referencing a slot assigned to device -2 (STARAN Associative Memory) the form of the corresponding BUFFER mnemonic is:

• • Format

Label	Command	Argument	Comment
symbol	<u>BUFFER</u>	<u>maxsize, address-mode, byte count, a1, a2, b1, b2</u>	

• • • Label

Any valid symbol or blank.

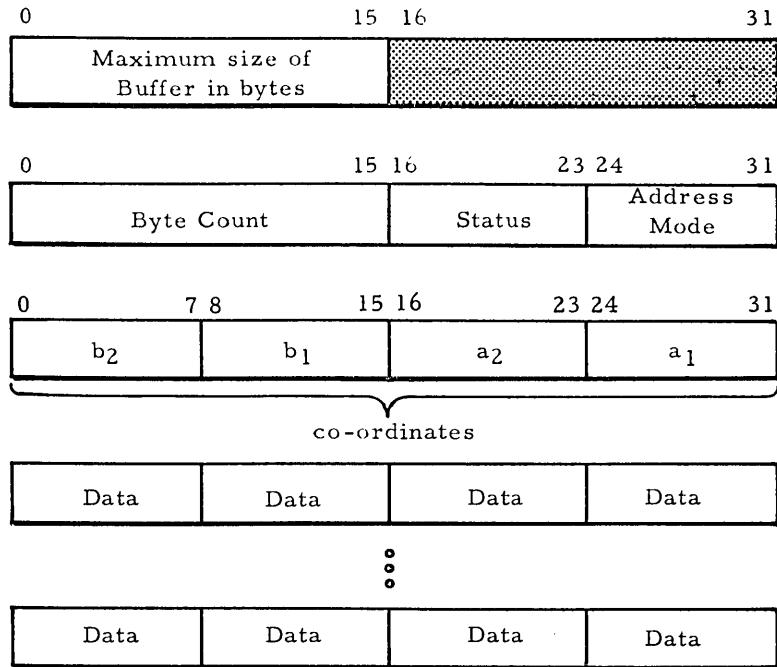
• • • Command

BUFFER

• • • Argument

The argument field consists of seven terms. Before describing these terms consider the three words of object-code generated by this particular BUFFER pseudo-op mnemonic (the words of data are not generated by this mnemonic).

• • • BUFFER
Format
For
Device -2



• • • Maximum
Size

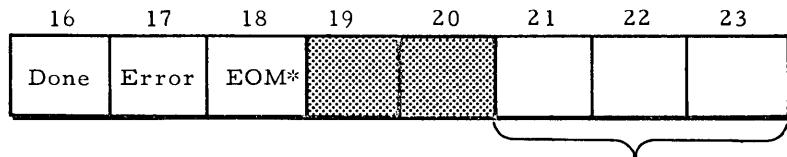
The upper half of the first word will contain the value of the first term of the argument field. This value represents the maximum size (in bytes) of the data words portion of the buffer and also the maximum allowable number of bytes of data that may be moved to or from the buffer.

• • • Byte
Count

The upper half of the second word will contain the value of the third term of the argument field. This value represents the actual number of bytes of data to be moved to or from the buffer area that follows. If for some reason an error condition occurs before a buffer transfer is completed, this value will be modified to reflect the actual number of data transfers.

• • • Status
Byte

Bits 16-23 of the second word contain the Status Byte. The bit value in this byte are set and maintained by SPS.



* End of Medium or Power Off or Out of Tape, etc.

Nonfatal Error Code

2=Checksum error

3=Long-line error

** Formatted Binary Data

4=Improper Mode (FBIN**)

*** End of Tape Code

5=EOT*** will occur if an ASCII EOT code is read outside of a FBIN block of data.

• • • • Done Bit

This bit will be set by SPS upon completion of the buffer transfer regardless of whether or not an error occurred.

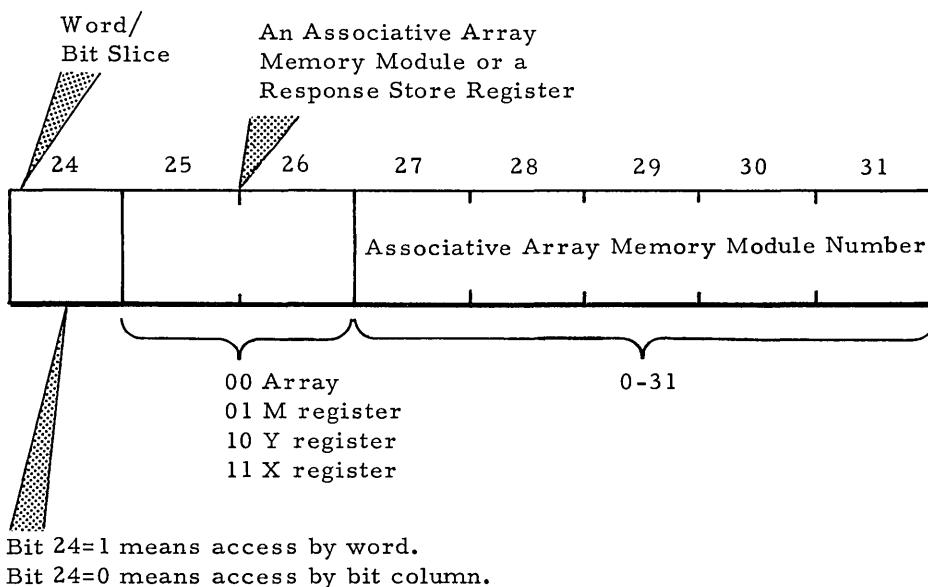
• • • • Error

This bit will be set by SPS if an error is detected during the buffer transfer operation. Examine bits 18 and 21-23 for the exact nature of the error.

• • • Address Mode Byte

Bits 24-31 of the second word comprise the Address Mode Byte and is loaded with the value of the second term in the argument field.

The interpretation of the bit and sub-field values of the byte by SPS is:



• • • Array Co-ordinates

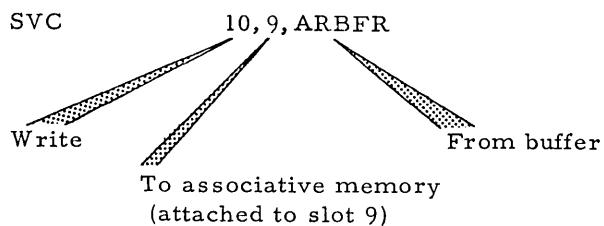
The third word of the buffer header information is loaded with the associative memory coordinate values as shown from the fourth through seventh terms in the argument field of the BUFFER pseudo-op mnemonic. a_1 and a_2 define the starting and ending "line" which may be bit column numbers or word row numbers, depending on bit 24 of the Address Mode; i.e., $0 \leq a_1 \leq a_2 \leq 255$. b_1 and b_2 are the starting and ending byte numbers for the data; i.e., $0 \leq b_1 \leq b_2 \leq 31$. If a response store register instead of associative memory is specified, a maximum of 32 bytes of data can be transferred.

• • Example

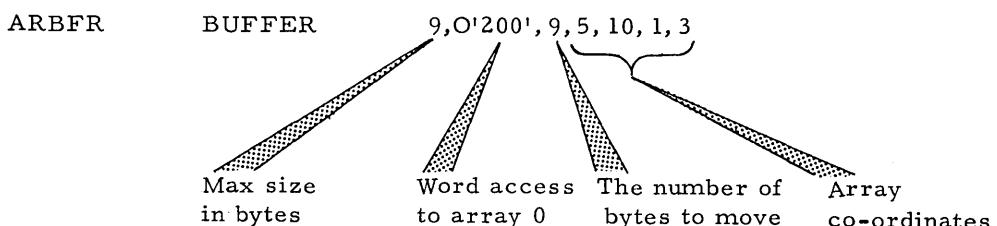
Writing Into Associative Memory

If slot 9 has been attached to associative memory (device code -2),
the following statements will load array 0 as shown.

• • • Supervisor
Call
Statement



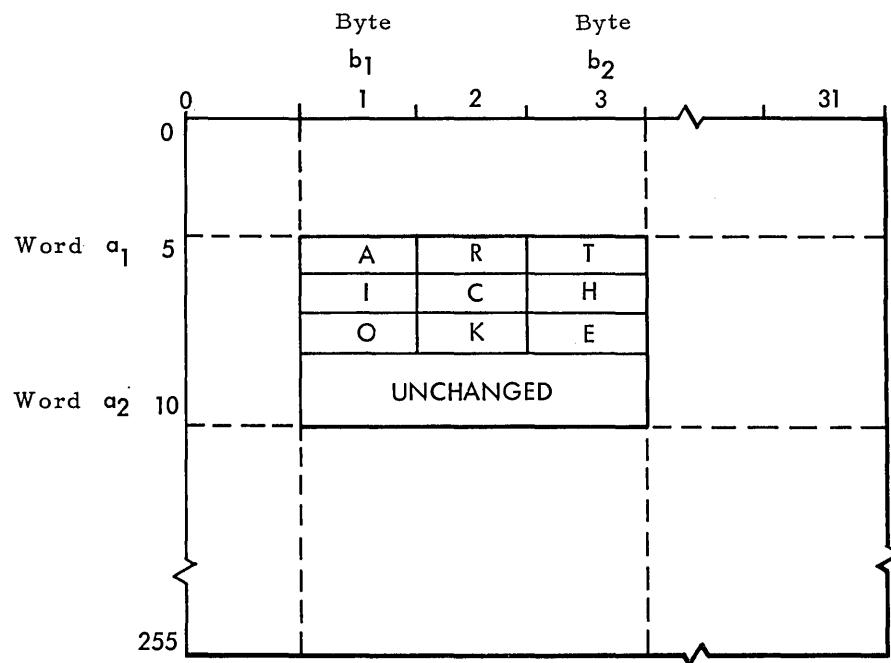
• • • Corresponding
BUFFER
Pseudo-op
Statement



• • • Buffer
Data

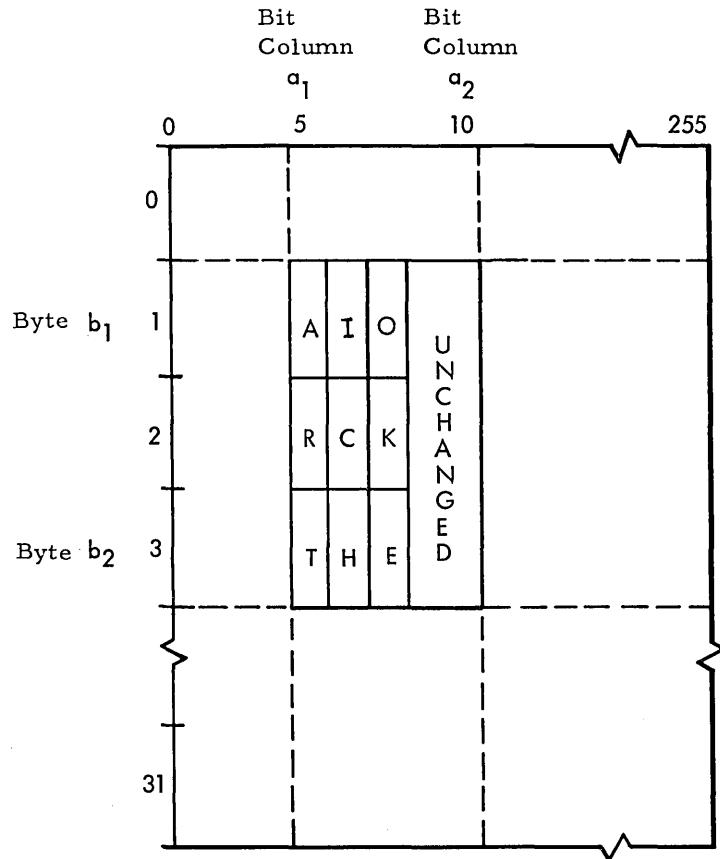
A'ARTICHOKE' ASCII character string

• • • Associative
Memory
Map,
Word
Mode



a₁ and a₂ define the starting and ending "line", 0 ≤ a₁ ≤ a₂ ≤ 255. b₁ and b₂ are the starting and ending byte numbers for the data, 0 ≤ b₁ ≤ b₂ ≤ 31.

- • • Associative Memory Map, Bit Column Mode



- STARAN Registers

When STARAN registers are referenced by an SVC instruction, it should be for the purpose of obtaining the value of those STARAN registers accessible only by the sequential controller. For an SVC READ instruction referencing a slot assigned to device -3 (STARAN Registers) the form of the corresponding BUFFER mnemonic is:

• • Format	Label	Command	Argument	Comment
	symbol	<u>BUFFER</u>	<u>register-code</u>	

- • • Label

Any valid symbol or blank.

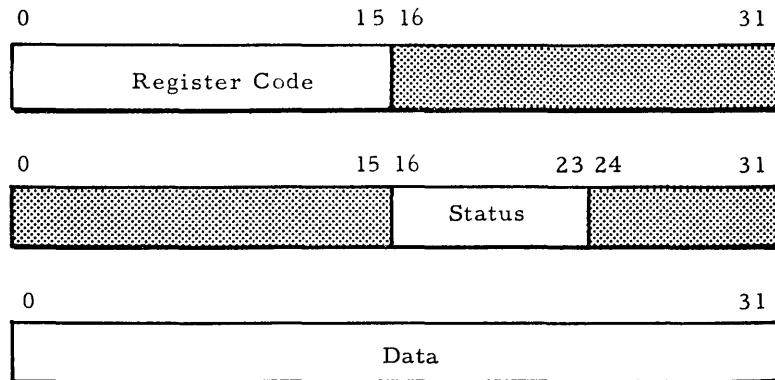
- • • Command

BUFFER

- • • Argument

The argument field consists of one term. Before describing this term consider the two words of object code generated by this particular BUFFER pseudo-op mnemonic (the word of data is not generated by this mnemonic).

• • Buffer
Format
For
Device -3



• • • Register
Code

The upper half of the first word will be loaded with the value of the argument field term. This value represents the code assigned to a particular register.

<u>STARAN Main Frame</u>		
Register	Register Code	Length (bytes)
Instruction	0	4
Pager Put	1	2
Pager Get	2	2
Pager Count	3	2
Start Loop Marker	4	2
End Loop Marker	5	2
<u>PI/O Unit* Main Frame</u>		
Instruction	6	4
Start Loop Marker	7	2
End Loop Marker	8	2
Buffer Register	9	4
Buffer Control Word No. 1	10	2
Buffer Control Word No. 2	11	2
Buffer Control Word No. 3	12	2
Block Length Counter	13	2
Data Pointer	14	2
Program Counter	15	2
Field Length Counter	16	1
Field Pointer No. 1	17	1
Field Pointer No. 2	18	1
Field Pointer No. 3	19	1
STARAN Address Mode	20	2
PI/O Address Mode	21	2
Performance Timer	22	2
Performance Counter	23	2

* The PI/O Unit is a STARAN option.

• • • Status Byte

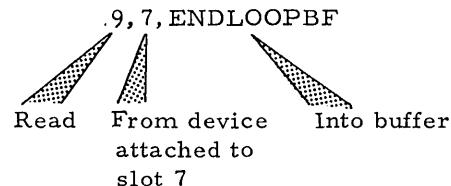


The status byte occupies bits 16-23 of the second word of the BUFFER instruction and is maintained by SPS. Only the Done and Error bits are of significance.

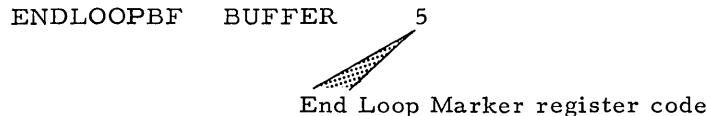
• • Example

Obtain the current value of the End Loop Marker register. Assure slot 7 has been assigned to STARAN registers (device code -3).

• • • Supervisor Call Statement



• • • Corresponding BUFFER Pseudo-op Statement



ENDLOOPRG DC 0 Buffer storage word

The value of the End Loop Marker register may be obtained from the contents of the one word buffer ENDLOOPRG.

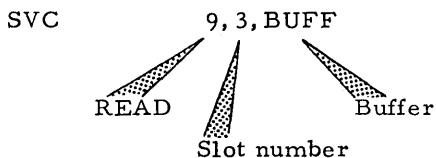
READ

This SVC function permits the transfer of data from a valid input device to a buffer area in Bulk core or the High-Speed Data Buffer. The input device is referenced by a slot number. Detail on memory to memory transfers for special AP devices is described in the ATTACH section.

Format	Label	Command	Argument	Comment
	symbol	<u>SVC</u>	<u>9, slot-number, buffer-address</u>	

Example

Assume slot number 3 has been attached to the high speed paper tape reader.



This example will cause data to be read from the high speed paper tape reader and stored into the buffer area defined by **BUFF** (buffer area in Bulk core or High Speed Data Buffer). **BUFF** must be defined in a **BUFFER** Pseudo-Op instruction.

WRITE

This SVC function permits the transfer of data from a buffer area in Bulk core or the High-Speed Data Buffer to a valid output device. The device is referenced by a slot number. Detail on memory-to-memory transfers for special AP devices is described in the ATTACH section.

Format	Label	Command	Argument	Comment
	symbol	SVC	<u>10, slot-number, buffer-address</u>	

- Label Any valid symbol or blank.
- Command SVC
- Argument Three entries are required.
 - • 10 This entry may be a constant or a symbol whose value is 10.
 - • Slot-Number This entry may be a constant or a symbol whose value is the slot-number assigned to an output device.
 - • Buffer-Address This entry may be a constant or a symbol representing the address of the associated buffer which is set-up in a BUFFER Pseudo-Op instruction. (READ/WRITE BUFFER Pseudo-Op instruction is described in a later section.)

Example

Assume slot number O'17' has been attached to the line printer.



This example will cause the contents of BUFF (buffer area in Bulk core or High-Speed Data Buffer) to be printed on the line printer. BUFF must be defined in a BUFFER Pseudo-Op instruction.

READ/
WRITE
BUFFER
PSEUDO-OP

The BUFFER Pseudo-Op instruction sets up a properly formatted buffer area in control memory for the READ and WRITE supervisor call functions. Detail on special AP device buffers is described in the ATTACH section.

Format	Label	Command	Argument	Comment
	symbol	<u>BUFFER</u>	<u>max-size</u> <u>address</u> <u>register code</u>	

- Label Any valid symbol or blank.
- Command BUFFER
- Argument The first three entries are required for all READ /WRITE operations except the AP register transfers (device code -3) which requires only the first entry. The remaining four entries are required for Associative memory data transfers.
- • Max-size This entry may be a constant or symbol whose value represents the maximum size of the buffer in bytes.
- • Address This entry may be a constant or symbol representing a Bulk core or High-Speed Data Buffer address in a memory to memory transfer.
- • Register Code This entry may be a constant or symbol representing an AP register code for data transfers involving AP registers.
- • Mode This entry may be a constant or symbol whose value represents the mode of data transfer. The mode entry will always be zero for a memory to memory transfer since mode is not applicable.

Value	Mode
3	Unformatted Binary (UBIN)
2	Unformatted ASCII (UASCII)
1	Formatted Binary (FBIN)
0	Formatted ASCII (FASCII)

- • • Unformatted Binary Eight bit bytes are transferred as specified by the buffer byte count. This mode is suitable for paper tape readers and punches. Possible errors are Error bit set and End of Medium (EOM)

- • • Unformatted ASCII Seven bits per byte are transferred as specified by the buffer byte count. This mode is suitable for keyboard, teleprinter, and line printer. Possible errors are Error bit set, End of Medium, Checksum, and Long line.

- • • Formatted Binary This format is used primarily for paper tape I/O (used by APPLE.) For output SPS blocks the data as follows:
 - 2018 } The header for the APPLE assembler output
 - 000 }
 - xxx } The byte count of the block, equal to the number
 - xxx } of data bytes plus 4.
 - yyy } Data bytes
 - yyy }
 - ⋮ }
 - zzz } The checksum, the two's complement of the sum
 - zzz } of all the preceding bytes in the block (Sum plus
 - checksum equals zero)

- When a block with a header 201000 (i.e. APPLE block) is encountered, the number of bytes transferred is the block byte count -4. The data are followed by a checksum. On output, SPS creates the header, the block byte-count, and the checksum.

- • • Formatted ASCII Seven bits per byte are transferred until a terminating character is encountered. Terminating characters are, a line feed (012), a form feed (014), or a carriage return (015). These characters will end transmission of data. Possible errors are Error bit set, End of medium, and Long line.

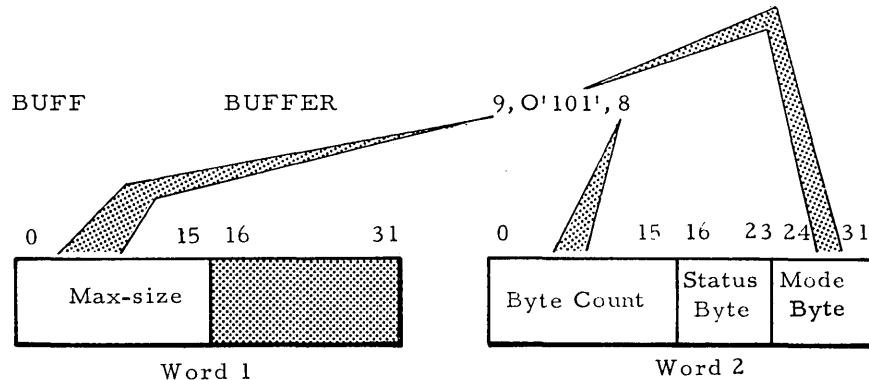
- • Byte Count Actual number of data bytes in the block to be transferred.

- • a_1, a_2, b_1, b_2 These entries are used only when an associative memory data transfer is performed. a_1 and a_2 define the start and end of the array words or bit slices (ranges: $0 \leq a_1 \leq a_2 \leq 255$). b_1 and b_2 define the start and end byte numbers for the data (range: $0 \leq b_1 \leq b_2 \leq 31$). If a response store register is specified, up to 32 bytes will be moved in or out of the buffer.

Example

The following example illustrates the buffer format for transfer of data to or from an I/O device.

- Buffer Pseudo-Op



- Buffer Header

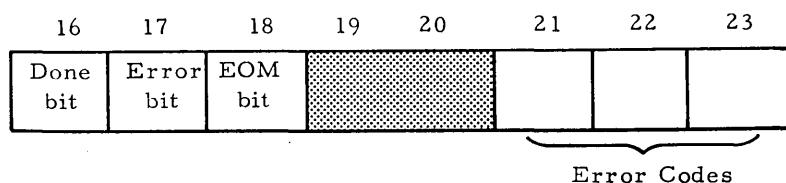
- • Max-Size

The maximum size, in bytes, of the buffer is contained in bits 0 to 15 of the first 32 bit buffer word.

- • Byte Count

This is the actual number of bytes of data in the buffer to be transferred.

- • Status Byte



- • • Done Bit

Bit 16=1 indicates a transfer to or from the buffer is complete.

- • • Error Bit

Bit 17=1 indicates that the device in use has signalled an error in the status register.

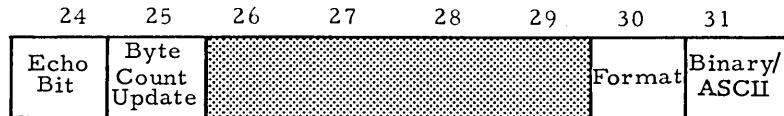
- • • EOM Bit

Bit 18=1 indicates an end of medium has been detected.

- • • Error Codes

<u>Error code</u>	<u>Error</u>
2	Checksum
3	Long line
4	Improper mode
5	End of tape marker detected

• • Mode
Byte



11 unformatted binary
01 formatted binary
10 unformatted ASCII
00 formatted ASCII

• • • Echo
Bit
Normally data entered through the keyboard are printed on the teleprinter (referred to as echo). If bit 7=1 in the mode byte, the echo is inhibited. This bit applies only to keyboard input.

• • • Byte
Count
Update
SPS revises the byte count in the buffer header after each I/O call using that buffer. This is not always desirable. For example, an error during attempted printing of a message might zero the byte count. Subsequent calls using that buffer would output zero bytes. Bit 6=1 will inhibit the updating and should be used only for a read operation.

• • • Format
Bit 30 indicates formatted or unformatted data mode. If bit 30 is zero, the mode is formatted, if bit 30 is equal to 1, the mode is unformatted.

• • • Binary/
ASCII
A one in bit 31 indicates binary mode; a zero in bit 31 indicates ASCII mode.

**RESTART
PROGRAM**

This statement will completely reinitialize the STARAN and return control to a sequential control program (refer to STARAN Systems Programmer's Reference Manual)

Format

Label	Command	Argument	Comment
symbol	<u>SVC</u>	<u>8</u>	

- Label Any valid symbol or blank.
- Command SVC
- Argument The argument must be an expression whose value is 8.

RESET

To reinitialize peripheral devices, issue RESET. This has the effect of RESTART, but control will return to the following instruction, not to a restart address. It is not usually necessary to RESET since conditions that call for reinitializing usually call for restarting, too.

Format

Label	Command	Argument	Comment
symbol	<u>SVC</u>	<u>2</u>	

- Label Any valid symbol or blank.
- Command SVC
- Argument Any expression whose value is 2 denotes the reset function.

FREE
DEVICE
For
NEW
TASK

The device attached to the designated slot will be made ready to start a new I/O process. If the device is busy, its current I/O process will be halted and cannot be resumed.

This service is most useful for getting an urgent - and usually terminal - message in or out.

Format	Label	Command	Argument	Comment
	symbol	SVC	<u>5, slot-number</u>	

- Label Any valid symbol or blank.

- Command SVC

- Argument Two entries are required.

- • 5 The first entry may be any expression whose value is 5; this denotes the FREE function of the supervisor call.

- • Slot-Number The second entry may be any expression whose value is between 0 and 15 (see figure 3-1).

Example KILL SVC 5, 0

The statement named KILL will cause the device attached to slot 0 (the teleprinter) to halt any current output and make it ready for a WRITE request.

EXIT
TO
SUPERVISOR

A STARAN program may return control to the supervisor with
this command.

Format	Label	Command	Argument	Comment
	symbol	<u>SVC</u>	<u>7</u>	

- Label Any valid symbol or blank.
- Command SVC
- Argument Any expression whose value is 7 denotes the EXIT function of the supervisor call.

TIMER
START

The TIMER statement allows clocking of an interval beginning with the execution of the statement. At the end of the specified interval, STARAN interrupt will be triggered.

Format	Label	Command	Argument	Comment
	symbol	<u>SVC</u>	<u>13, timer-number, interrupt-number, time-value</u>	

Example EXCELS SVC 13, 2, 5, 300

When the above statement is executed, timer 2 will trigger AP interrupt 5 in 1 second (300/300ths).

INTERRUPT
SIGNAL

This command is used to cause the sequential processor to execute a program. SPS simulates sixteen interrupt vectors in the sequential processor. (This is not a STARAN hardware interrupt into the sequential processor, signalled by external-function codes. It is a software facility to make possible user linkages between STARAN and the sequential processor.) When the software interrupt is triggered, the sequential processor will execute a program at the address specified in the I SETUP call (discussed on following page).

Format	Label	Command	Argument	Comment
	symbol	SVC	<u>14, interrupt-number</u>	

- Label Any valid symbol or blank.
- Command SVC
- Argument The argument field consists of two entries.
 - • 14 The first entry must be an expression whose value is 14; this denotes the INT function of the supervisor call.
 - • Interrupt-Number The second entry is an expression whose value must be 0, 1, 2, . . . , 14, 15. It specifies a software interrupt to the sequential controller.

Example SVC 14, 0

This will trigger the interrupt setup as shown in the next section (I SETUP example).

I SETUP

This SVC function creates a software interrupt vector for the sequential controller.

Format

Label	Command	Argument	Comment
symbol	SVC	<u>15, interrupt-number, status,</u> <u>interrupt-vector-address</u>	

- Label Any valid symbol or blank.

- Argument All entries are required.

- • 15 The first entry must be an expression whose value is 15; it denotes the I SETUP function of the supervisor call.

- • Interrupt-Number The second entry is an expression whose value must be 0, 1, 2, ..., 14, or 15. It specifies a software interrupt vector maintained by SPS. (See Staran System Programmer's Reference Manual.)

- • Status The third argument specifies the status to be assumed when the interrupt is signalled. It will be evaluated and taken to be a number from zero to seven.

- • Interrupt-Vector The fourth argument is an expression whose value is a sequential control address in Bulk core memory.

Example EXETER SVC 15, 0, 7, HANDLR

The above line of coding will attach sequential control software interrupt 0 to a routine called HANDLR. Priority 7 will be assumed.

PAGER
CONTROL

This SVC function is used to control certain Pager operations.

Format	Label	Command	Argument	Comment
	symbol	SVC	18, operation, start-address	

- Label Any valid symbol or blank.
- Command SVC
- Argument Two entries are required in all operations except the Start Pager , which requires all three entries.
- • 18 The first entry may be a constant or a symbol whose value is 18.
- • Operation The second entry may be a constant or a symbol with the following values:

<u>Value</u>	<u>Operation</u>
0	Start Pager at start-address
1	Stop Pager
2	Pause Pager
3	Continue Pager from pause

- • Start-Address The start-address is used only in the Start Pager operation (i.e. when the operation value is zero).

Example	TAG	DC	0	
		.	.	
		.	.	
	SVC	18, TAG, X'00'	Start Pager at address 0	
		.	.	
		.	.	
	SVC	18, 1	Stop Pager	

**PI/O
CONTROL***

This SVC function is used to control certain parallel I/O operations.

Format

Label	Command	Argument	Comment
symbol	SVC	19, operation, start-address	

- Label Any valid symbol or blank.

- Command SVC

- Argument Two entries are required in all operations except the Start PI/O , which requires all three entries.

- • 19 The first entry may be a constant or a symbol whose value is 19.

- • Operation The second entry may be a constant or a symbol with the following values:

<u>Value</u>	<u>Operation</u>
0	Start PI/O at start-address
1	Stop PI/O
2	Pause PI/O
3	Continue PI/O from pause

- • Start-Address The start-address is used only in the Start PI/O operation (i.e. when the operation value is zero).

* Parallel I/O is an optional STARAN feature. Other Custom I/O features may be handled similarly.

APPENDIX A

**SUMMARY OF APPLE MNEMONICS
AND
INSTRUCTION FORMATS**

	<u>Mnemonic</u> <u>(Command)</u>	<u>Argument</u>	<u>Instruction</u>	<u>Page</u>
<u>ASSEMBLER DIRECTIVES</u>	<u>START</u> *		Start APPLE source	2-10
	<u>END</u>	<u>a±k</u>	End APPLE source	2-10
	<u>ORG</u>	<u>a±k</u>	Initialize location counter	2-11
	<u>Label EQU</u>	<u>a±k</u>	Equate (Define Symbol)	2-12
	<u>Label DF</u>	<u>a₁±k₁, a₂±k₂</u>	Define Field	2-12
	<u>DS, a±k</u>		Define Storage	2-13
	<u>TOF</u>		Top of Form	2-13
	<u>EVEN</u>		Make location counter EVEN	2-14
	<u>DC, a₁±k₁</u>	<u>a₂±k₂</u>	Define Constant	2-14
	<u>GEN, k₁, ..., k_n</u>	<u>a₁±j₁, ..., a_n±j_n</u>	Generate Machine Instruction	2-15
	<u>NOP</u>		No Operation	2-15
	<u>Axc₁c₂...c_{i-1}c_ix</u>		Character String Generator	2-16
	<u>Exc₁c₂...c_{i-1}c_ix</u>		Character String Generator	2-16
<u>BRANCH INSTRUCTIONS</u>	<u>B</u>	<u>a(r)±k, cd</u>	Unconditional Branch	2-18
	<u>BZ, r₁</u>	<u>a(r₂)±k, cd</u>	Branch if Zero	2-19
	<u>BNZ, r₁</u>	<u>a(r₂)±k, cd</u>	Branch if Not Zero	2-21
	<u>BBS</u>	<u>a(r)±k, cd</u>	Branch if Bit Set	2-23
	<u>BBZ</u>	<u>a(r)±k, cd</u>	Branch if Bit Zero	2-25
	<u>BRS</u>	<u>a(r)±k, cd</u>	Branch if Response	2-27
	<u>BNR</u>	<u>a(r)±k, cd</u>	Branch if No Response	2-28
	<u>BOV</u>	<u>a(r)±k, cd</u>	Branch if Overflow	2-29
	<u>BNOV</u>	<u>a(r)±k, cd</u>	Branch if No Overflow	2-30
	<u>BAL, r₁</u>	<u>a(r₂)±k, cd</u>	Branch and Link	2-31
	<u>RPT, a±k</u>		Repeat	2-33
	<u>LOOP, a₁±k₁</u>	<u>a₂(r)±k₂</u>	Loop	2-34
<u>REGISTER INSTRUCTIONS</u>	<u>LRR, k_s</u>	<u>r₂, r₁</u>	Load Register from Register	2-37
	<u>LI, k_s</u>	<u>r, a±k</u>	Load Register with Immediate Data	2-39
	<u>LR, k_s</u>	<u>r₂, a(r₁)±k, cd</u>	Load Register from Control Memory	2-41
	<u>SR, r_s</u>	<u>r₂, a(r₁)±k, cd</u>	Store Register in Control Memory	2-47
	<u>INCR</u>	<u>r₁, ..., r_n</u>	Increment the Register	2-49
	<u>DECR</u>	<u>r₁, ..., r_n</u>	Decrement the Register	2-50
	<u>LPSW, k_s</u>	<u>a(r)±k, cd</u>	Load Program Status Word	2-51
	<u>SPSW</u>	<u>a(r)±k, cd</u>	Swap Program Status Word	2-53

* Required entries are underlined throughout

Mnemonic (Command)	<u>Argument</u>	<u>Instruction</u>	<u>Page</u>
ASSOCIATIVE INSTRUCTIONS LOAD RESPONSE STORE REGISTERS AND COMMON REGISTER	<u>L</u> $\underline{rs}_2, \left\{ \begin{array}{l} \underline{rs}_1 \\ \underline{a} \pm \underline{k} \\ \underline{r} \end{array} \right\}$	Load Response Store Register	2-56
	<u>LN</u> $\underline{rs}_2, \left\{ \begin{array}{l} \underline{rs}_1 \\ \underline{a} \pm \underline{k} \\ \underline{r} \end{array} \right\}$	Load Complemented	2-58
	<u>LOR</u> $\underline{rs}_2, \left\{ \begin{array}{l} \underline{rs}_1 \\ \underline{a} \pm \underline{k} \\ \underline{r} \end{array} \right\}$	Load Logical OR	2-60
	<u>LORN</u> $\underline{rs}_2, \left\{ \begin{array}{l} \underline{rs}_1 \\ \underline{a} \pm \underline{k} \\ \underline{r} \end{array} \right\}$	Load Logical OR Complemented	2-62
	<u>LAND</u> $\underline{rs}_2, \left\{ \begin{array}{l} \underline{rs}_1 \\ \underline{a} \pm \underline{k} \\ \underline{r} \end{array} \right\}$	Load Logical AND	2-64
	<u>LANDN</u> $\underline{rs}_2, \left\{ \begin{array}{l} \underline{rs}_1 \\ \underline{a} \pm \underline{k} \\ \underline{r} \end{array} \right\}$	Load Logical AND Complemented	2-66
	<u>LXOR</u> $\underline{rs}_2, \left\{ \begin{array}{l} \underline{rs}_1 \\ \underline{a} \pm \underline{k} \\ \underline{r} \end{array} \right\}$	Load Logical Exclusive OR	2-68
	<u>LXORN</u> $\underline{rs}_2, \left\{ \begin{array}{l} \underline{rs}_1 \\ \underline{a} \pm \underline{k} \\ \underline{r} \end{array} \right\}$	Load Logical Exclusive OR Complemented	2-70
	<u>LC</u>	Load Common Register from an Associative Memory Word	2-72
	<u>LCM</u>	Load Common Register Field from an Associative Memory Word	2-74
	<u>SET</u>	Set Response Store Register	2-76
	<u>CLR</u>	Clear Response Store Register	2-77
	<u>ROT</u>	Rotate Response Store Register	2-78

Mnemonic (Command)	Argument	Instruction	Page
STORE RESPONSE STORE REGISTERS AND COMMON REGISTER	<u>S</u> <u>rs,</u> { $\frac{a \pm k}{r}$ }	Store Response Store Into Associative Memory	2-80
	<u>SM</u> <u>rs,</u> { $\frac{a \pm k}{r}$ }	Store Response Store Masked Into Associative Memory	2-82
	<u>SN</u> <u>rs,</u> { $\frac{a \pm k}{r}$ }	Store Complement Into Associative Memory	2-84
	<u>SNM</u> <u>Y,</u> { $\frac{a \pm k}{r}$ }	Store Complement Masked Into Associative Memory	2-86
	<u>SOR</u> <u>rs,</u> { $\frac{a \pm k}{r}$ }	Store Logical Inclusive OR Into Associative Memory	2-88
	<u>SORM</u> <u>Y,</u> { $\frac{a \pm k}{r}$ }	Store Logical Inclusive OR, Masked Into Associative Memory	2-90
	<u>SORN</u> <u>rs,</u> { $\frac{a \pm k}{r}$ }	Store Logical Inclusive OR, Complemented Into Associative Memory	2-92
	<u>SORNM</u> <u>Y,</u> { $\frac{a \pm k}{r}$ }	Store Logical Inclusive OR, Complemented, Masked Into Associative Memory	2-94
	<u>SAND</u> <u>rs,</u> { $\frac{a \pm k}{r}$ }	Store Logical AND Into Associative Memory	2-96
	<u>SANDM</u> <u>Y,</u> { $\frac{a \pm k}{r}$ }	Store Logical AND Masked Into Associative Memory	2-98
	<u>SANDN</u> <u>rs,</u> { $\frac{a \pm k}{r}$ }	Store Logical AND Complemented Into Associative Memory	2-100
	<u>SANDNM</u> <u>Y,</u> { $\frac{a \pm k}{r}$ }	Store Logical AND, Complemented, Masked Into Associative Memory	2-102
	<u>SC</u> <u>a₁,a₂</u>	Store Common Register Into Associative Memory	2-104
	<u>SCW</u> <u>a₁,a₂</u>	Store Common Register Into Associative Word	2-106

SEARCHES	<u>FIND</u>	Find the First Bit Set in Y Response Store	2-109
	<u>STEP</u>	Step to First Y Set and Clear It	2-109
	<u>RESVFST</u>	Step to First Y Set and Clear All Others	2-110
	<u>EQC</u> <u>a₁,a₂</u>	Equal to Common Register Field	2-111
	<u>EQF</u> <u>a₁,a₂</u>	Equal Fields	2-112
	<u>NEC</u> <u>a₁,a₂</u>	Not Equal to Common Register Field	2-113
	<u>NEF</u> <u>a₁,a₂</u>	Not Equal Fields	2-114
	<u>GTC</u> <u>a₁,a₂</u>	Greater Than Common Register Field	2-115
	<u>GTF</u> <u>a₁,a₂</u>	Greater Than Fields	2-116
	<u>GEC</u> <u>a₁,a₂</u>	Greater than or Equal to Common Register Field	2-117
	<u>GEF</u> <u>a₁,a₂</u>	Greater than or Equal Fields	2-118
	<u>LTC</u> <u>a₁,a₂</u>	Less Than Common Register Field	2-119
	<u>LTF</u> <u>a₁,a₂</u>	Less Than Fields	2-120
	<u>LEC</u> <u>a₁,a₂</u>	Less Than or Equal Common Register Field	2-121
	<u>LEF</u> <u>a₁,a₂</u>	Less Than or Equal Fields	2-122
	<u>MAXF</u> <u>a</u>	Maximum Fields	2-123
	<u>MINF</u> <u>a</u>	Minimum Fields	2-124

	<u>Mnemonic</u> <u>(Command)</u>	<u>Argument</u>	<u>Instruction</u>	<u>Page</u>
MOVES	<u>MVF</u>	<u>a₁,a₂</u>	Move Field	2-126
	<u>MVCF</u>	<u>a₁,a₂</u>	Move the One's Complement of a Field	2-128
	<u>MVNF</u>	<u>a₁,a₂</u>	Move the Negative of a Field	2-130
	<u>MVAF</u>	<u>a₁,a₂</u>	Move the Absolute Value of a Field	2-132
	<u>INCF</u>	<u>a₁,a₂</u>	Move Field with Increment	2-134
	<u>DEC F</u>	<u>a₁,a₂</u>	Move Field with Decrement	2-136
ARITHMETICS	<u>ADC</u>	<u>a₁,a₂,a₃</u>	Add Common Register to Field	2-139
	<u>ADF</u>	<u>a₁,a₂,a₃</u>	Add Field to Field	2-141
	<u>SBC</u>	<u>a₁,a₂,a₃</u>	Subtract Common Register from Field	2-143
	<u>SBF</u>	<u>a₁,a₂,a₃</u>	Subtract Field from Field	2-145
	<u>MPC</u>	<u>a₁,a₂,a₃</u>	Multiply Field by Common Register	2-147
	<u>MPF</u>	<u>a₁,a₂,a₃</u>	Multiply Field by Field	2-149
	<u>DVF</u>	<u>a₁,a₂,a₃</u>	Divide Field by Field	2-151
CONTROL AND TEST	<u>INT,a₁±k₁</u>	<u>a₂,±k₂</u>	Interrupt Control and Test	2-155
	<u>ILOCK,a₁±k₁</u>	<u>a₂,±k₂</u>	Interlock Control and Test	2-157
	<u>WAIT</u>		Deactivate the AP	2-158
PAGER INSTRUCTIONS	<u>STR TSG,a±k</u>		Start Segment	2-160
	<u>END SG</u>		End Segment	2-160
	<u>MVSG,a₁±k₁</u>	<u>a₂±k₂</u>	Move a Page Segment	2-161
	<u>MVSG I,a₁±k₁</u>	<u>a₂±k₂</u>	Move a Page Segment Immediate	2-162
	<u>PAGER</u>	<u>a±k</u>	Pager Control	2-163

APPLE I/O
Statements

<u>Mnemonic (Command)</u>	<u>Arguments</u>	<u>Instruction</u>	<u>Page</u>
BUFFER	Maxsize, mode, byte-count	Buffer-header Pseudo-op	3-17
SVC	1, slot-number, device-code-address	Attach Device to Slot	3-5
SVC	2	Reset Peripherals	3-22
SVC	5, slot-number	Free a Device for I/O	3-23
SVC	7	Exit From Program	3-24
SVC	8	Restart Program	3-21
SVC	9, slot-number, buffer-address	Read Into Buffer	3-15
SVC	10, slot-number, buffer-address	Write From Buffer	3-16
SVC	13, timer-number, interrupt-number, time-value	Start a Timer	3-25
SVC	14, interrupt-number	Int-Signal Interrupt	3-26
SVC	15, interrupt-number, status, done-address	I Setup-Interrupt	3-27
SVC	18, operation, start-address	Pager Control	3-28
SVC	19, operation, start-address	PI/O Unit Control	3-29

APPENDIX B

ERROR CODES

ERROR CODES

When APPLE scans source statements to produce the object code, it checks for improper use of the defined grammar. Up to two error codes can be printed in the left hand margin for each statement in error. Error code meanings are listed below.

<u>Error Code</u>	<u>Meaning</u>
A	<u>Addressing error.</u> An address within the instruction is incorrect.
B	<u>Boundary error.</u> An address that should be even (odd) is odd (even).
D	<u>Doubly-defined symbol referenced.</u> Reference is made to a symbol that is defined more than once.
F	<u>Illegal forward reference of a symbol.</u>
I	<u>Illegal character detected.</u>
K	<u>Array address out of range.</u>
L	<u>Lengths of array fields incompatible.</u>
M	<u>Multiple definition of a label.</u> A label is encountered that is identical to a previously encountered label.
P	<u>A page segment boundary syntax error.</u>
Q	<u>Questionable syntax.</u> There are missing arguments or the instruction scan was not completed.
R	<u>Register-type error.</u> An invalid use of or reference to a register has been made.
S	<u>Symbol table overflow.</u> When the quantity of user-defined symbols exceeds the allocated space available in the user's symbol table, the assembler outputs the current source line with the S error code, then returns to the initial dialogue.
T	<u>Truncation error.</u> A number being loaded into a register or storage location is larger than the length the register or location allows.
U	<u>Undefined symbol.</u> An undefined symbol is encountered during the evaluation of an expression. Relative to the expression, the undefined symbol is assigned a value of zero.
V	<u>Value out of range.</u>
W	<u>Warning.</u> Nonstandard usage or procedure. Processing continues.

APPENDIX C

TERMS AND SYMBOLS

AP	Associative Processor
Associative Memory	An associative array memory module consists of two basic components: array storage and response store. Each array contains 65,536 bits, organized as a square 256 words by 256 bits of solid state storage. Array input and output may be either 32 bits or 256 bits in parallel. Input data may be stored into the array through a mask contained in the response store.
Array Selector	The Array Select register establishes those associative array memory modules that are to be active for an associative operation. The Array Select register is 32 bits wide. Each bit position controls one array, i.e., bit 0 corresponds to array 0, bit 1 corresponds to array 1, etc. A value of one in an Array Select register bit position will enable the corresponding array number.
AS	Array Select register (32 bits)
ASH	Array Select register, High half (bits 0-15)
ASL	Array Select register, Low half (bits 16-31)
Associative Array	See Associative Memory
BL	Block length counter (16 bits)
Block Length counter	The block length counter is a 16-bit decrementing counter. The block length counter may be used to control the length of a data block transfer.
Branch and Link registers	A group of registers that occupies dedicated memory locations in the HSDB (addresses 6001_6 to 6071_6). They are used as linkages to subroutines.
Bulk Core	The bulk core memory is a section of AP control memory used to store instructions or data. In the standard STARAN S configuration it contains 16,384 words (32 bits each). Bulk core addresses range from 80001_6 to $BFFF1_6$.
C	Common register (32 bits)
cd	Control Digit

CDR	Card Reader
CH	Common register, High half (bits 0-15)
CL	Common register, Low half (bits 16-31)
Common Register	The Common register is an AP register that contains 32 bits numbered 0 to 31. Bit 0 is the left-most (most-significant) bit. Bit 31 is the right-most (least-significant) bit. The Common register may contain the argument for a search operation performed upon the associative memory, the input data stored into an associative memory, or the input data received from an associative memory in a load operation. Data from an associative memory is loaded into the Common register through a mask generated by the mask generator.
Control Digit	A control digit permits post-incrementing or post-decrementing of the DP register and/or post-decrementing the BL register. It is implemented in instructions with a Control Memory address, using the DP register as a base register.
Control Memory Address	The Control memory address is a symbolic or absolute address in Bulk Core, Page Memory, or the High Speed Data buffer. Valid address ranges are 000_{16} to $7FF_{16}$ and 8000_{16} to BFF_{16} .
Data Pointer	The data pointer is a 16-bit register in AP control that may contain the control memory address for block transfers. The data pointer can be stepped with each transfer within a data block.
DMA	Direct Memory Access
DP	Data Pointer (16 bits)
DP0	Data Pointer, byte 0 (bits 0-7)
DP1	Data Pointer, byte 1 (bits 8-15)
EOM	End of Medium
EOT	End of Tape
Execution Location Counter	The Execution Location counter indicates the address of the instruction when it is executed. This will differ from the Load Location Counter only when program segments are moved to a Page Memory for execution.

FASCII	Formatted ASCII Code
FBIN	Formatted Binary Code
Field Expression	<p>There are two ways of denoting a field expression:</p> <ol style="list-style-type: none"> 1) $b \pm i$ <p>where b must be a symbol, and i is an optional constant modifier. b should have been previously defined in a DF instruction. b represents the most-significant bit position and the number of contiguous bits occupied by a field in either the Common register or associative memory. The optional constant modifier, i, modifies only the most-significant bit position.</p> <ol style="list-style-type: none"> 2) $(b, i) \pm j$ <p>where b may be a constant or a symbol and represents the most significant bit position of a field. If b was defined as a field via a previous DF instruction, the most-significant bit position is the value used. i must be a constant and represents the number of contiguous bits occupied by the field. j is an optional constant modifying only the most-significant bit position of the field.</p>
Field Length Counter	Field length counters are 8-bit AP control registers used to contain the length of data fields. They may be decremented to allow stepping through the bits of a data field. When the counter's contents equal zero, an indication is sent to the AP control for test purposes. There are two field length counters: FL1 and FL2.
Field Pointers	A field pointer is an 8-bit AP control register that generally contains an array bit column or word address. Field pointers may be incremented or decremented to facilitate stepping through data fields. There are four field pointers: FP1, FP2, FP3, and FPE.
FL1	Field Length counter 1 (8 bits)
FL2	Field Length counter 2 (8 bits)
FPE	Field Pointer E (8 bits)
FP1	Field Pointer 1 (8 bits)
FP2	Field Pointer 2 (8 bits)
FP3	Field Pointer 3 (8 bits)

High-Speed Data Buffer	The High-Speed Data Buffer is a section of AP Control memory consisting of fast solid state elements. In the standard configuration of STARAN S it contains 512 words with addresses from 600_{16} to $7FF_{16}$. Since the HSDB can be accessed faster than bulk core, it is a convenient place to store data and instructions that require quick access.
Interlocks	The EXF logic contains 64 stored bits called interlocks. These bits have no predetermined meaning. Software may assign a meaning to an interlock and use it for any purpose. Sixteen interlocks (hex addresses 00 through 0F) can be controlled and sensed manually by panel switches and lights to facilitate communication with an operator. The other 48 interlocks (hex addresses 10 through 3F) can only be sensed and controlled by function codes.
Interrupt MASK	The program status word in the program control logic contains the interrupt mask for the 15 AP control interrupts. All interrupts with numbers greater than the mask are accepted. The interrupt mask is contained in bits 28 through 31 of the program status word.
Link Pointer	The link pointer is registers FP1 and FP2 concatenated together. FP1 contains the address of the selected associative array memory module and FP2 contains the array word or bit column address. The link pointer is commonly used to store the address of the first responder of a search operation.
Load Location Counter	The load location keeps track of the addresses associated with instructions when they are loaded.
LSB	Least Significant Bit (bit 31 of 32-bit word); right-most bit; low order bit.
M	M-Response Store Register; MASK (256 bits)
M Response Store register	The M response store register (MASK) is a 256-bit register contained in the response store element of each associative array memory module. Its special use is to select associative memory words participating in an associative operation.
MASK	M-Response Store Register

Masked Store	Data being stored into associative memory may be stored through a mask which is contained in the M response store register. This is a masked store. Data will be stored only into words that have the corresponding M register bit set. Other words are unchanged.
MDA	Multi-Dimensional Access memory; associative memory
MSB	Most Significant Bit (bit 0 of word); left-most bit; high-order bit.
Page Memory	Three page memories of 512 words each are included in the AP control memory of the standard STARAN S configuration. They are fast memories that should be used for program segments that require frequent usage and/or fast execution. The page memory address ranges are: Page 0 - 000_{16} to $1FF_{16}$; Page 1 - 200_{16} to $3FF_{16}$; Page 2 - 400_{16} to $5FF_{16}$.
Page0	High-speed solid state memory; 512 32-bit words
Page1	High-speed solid state memory; 512 32-bit words
Page2	High-speed solid state memory; 512 32-bit words
PC	Program Counter (16 bits)
PARALLEL INPUT / OUTPUT (OPTIONAL FEATURE)	Each associative array in STARAN can have up to 256 inputs and 256 outputs into the custom I/O cabinet. The basic width of the parallel input/output (PI/O) is 256 n where n is equal to the number of associative arrays in the system (n can have a maximum value of 32). The custom I/O cabinet is capable of buffering and reformatting the data received from any peripheral device to match the width necessary to communicate with the STARAN associative array.
Program Counter	The program counter occupies bits 0-15 of the program status word in AP control. The program counter contains the address of the current instruction being executed. It is normally incremented sequentially through control memory. Its normal sequence may be altered by branch or loop instruction.

Program Status Word

The Program Status Word (PSW) consists of the program counter (PC) (bits 0-15), which contains the address of the current AP control instruction being executed, and the Interrupt Mask (IMASK) (bits 28-31), which contains the current interrupt status.

PSW **Program Status Word**

Resolve

The resolver logic in AP control finds the associative array memory module address and word address of the first responder. The array address is loaded into FP1 and the word address into FP2 (see link pointer). This permits subsequent operations to only affect the first responder.

Responder

A responder is a response store element in an enabled associative array memory module whose Y register bit is set. Generally, responders indicate words satisfying some search criteria. The Y register can be tested for a response or a no-response condition.

Response **See responder**

RS **Response Store**

R0 Branch and Link Register (memory location 600_{16})

R1 Branch and Link Register (memory location 601_{16})

R2 Branch and Link Register (memory location 601_{16})

R3 Branch and Link Register (memory location 603_{16})

R4 Branch and Link Register (memory location 604_{16})

R5 Branch and Link Register (memory location 605_{16})

R6 Branch and Link Register (memory location 606_{16})

R7 Branch and Link Register (memory location 607_{16})

SPS **STARAN Program Supervisor**

SVS **Supervisor call**

UBIN Unformatted Binary

UASCII Unformatted ASCII

X	X Response Store Register (256 bits)
X Response Store Register	The X response store register is a 256-bit register contained in the response store element of each associative array memory module. It may be used as temporary storage of data loaded from the array or stored into the array. It can be combined logically with data from the input network and/or the Y register. It is useful as temporary storage in parallel arithmetic operations or searches.
Y	Y Response Store register (256 bits)
Y Response Store Register	The Y response store register is a 256-bit register contained in the response store element of each associative array memory module. It may be used as temporary storage of data loaded from the array or stored into the array. It can be combined logically with data from the input network. It is useful as temporary storage in parallel arithmetic operations and searches. It is also used as the responder in a resolve operation.

APPENDIX D

HEXADECIMAL/DECIMAL TABLE

HEXADECIMAL-DECIMAL NUMBER CONVERSION TABLE

GENERAL

The table provides for direct conversion of hexadecimal and decimal numbers in these ranges:

Hexadecimal	Decimal
000 to FFF	0000 to 4095

HEXADECIMAL- DECIMAL NUMBER CONVERSION

In the table, the decimal value appears at the intersection of the row representing the most significant hexadecimal digits (16^2 and 16^1) and the column representing the least significant hexadecimal digit (16^0).

Example

$C21_{16}$	=	3105_{10}
HEX	0	1 2
C0	3072	3073 3074
C1	3088	3089 3090
C2	3104	(3105) 3106
C3	3120	3121 3122

For numbers outside the range of the table, add the following values to the table figures:

Hexadecimal	Decimal	Hexadecimal	Decimal
1000	4,096	C000	49,152
2000	8,192	D000	53,248
3000	12,288	E000	57,344
4000	16,384	F000	61,440
5000	20,480	10000	65,536
6000	24,576	20000	131,072
7000	28,672	30000	196,608
8000	32,768	40000	262,144
9000	36,864	50000	327,680
A000	40,960	60000	393,216
B000	45,056	70000	458,752

Example

$$1C21_{16} = 7201_{10}$$

Hexadecimal	Decimal
-------------	---------

$C21$	3105
$+1000$	4096
<hr/>	<hr/>
$1C21$	7201

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	0000	0001	0002	0003	0004	0005	0006	0007	0008	0009	0010	0011	0012	0013	0014	0015
01	0016	0017	0018	0019	0020	0021	0022	0023	0024	0025	0026	0027	0028	0029	0030	0031
02	0032	0033	0034	0035	0036	0037	0038	0039	0040	0041	0042	0043	0044	0045	0046	0047
03	0048	0049	0050	0051	0052	0053	0054	0055	0056	0057	0058	0059	0060	0061	0062	0063
04	0064	0065	0066	0067	0068	0069	0070	0071	0072	0073	0074	0075	0076	0077	0078	0079
05	0080	0081	0082	0083	0084	0085	0086	0087	0088	0089	0090	0091	0092	0093	0094	0095
06	0096	0097	0098	0099	0100	0101	0102	0103	0104	0105	0106	0107	0108	0109	0110	0111
07	0112	0113	0114	0115	0116	0117	0118	0119	0120	0121	0122	0123	0124	0125	0126	0127
08	0128	0129	0130	0131	0132	0133	0134	0135	0136	0137	0138	0139	0140	0141	0142	0143
09	0144	0145	0146	0147	0148	0149	0150	0151	0152	0153	0154	0155	0156	0157	0158	0159
0A	0160	0161	0162	0163	0164	0165	0166	0167	0168	0169	0170	0171	0172	0173	0174	0175
0B	0176	0177	0178	0179	0180	0181	0182	0183	0184	0185	0186	0187	0188	0189	0190	0191
0C	0192	0193	0194	0195	0196	0197	0198	0199	0200	0201	0202	0203	0204	0205	0206	0207
0D	0208	0209	0210	0211	0212	0213	0214	0215	0216	0217	0218	0219	0220	0221	0222	0223
0E	0224	0225	0226	0227	0228	0229	0230	0231	0232	0233	0234	0235	0236	0237	0238	0239
0F	0240	0241	0242	0243	0244	0245	0246	0247	0248	0249	0250	0251	0252	0253	0254	0255
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
10	0256	0257	0258	0259	0260	0261	0262	0263	0264	0265	0266	0267	0268	0269	0270	0271
11	0272	0273	0274	0275	0276	0277	0278	0279	0280	0281	0282	0283	0284	0285	0286	0287
12	0288	0289	0290	0291	0292	0293	0294	0295	0296	0297	0298	0299	0300	0301	0302	0303
13	0304	0305	0306	0307	0308	0309	0310	0311	0312	0313	0314	0315	0316	0317	0318	0319
14	0320	0321	0322	0323	0324	0325	0326	0327	0328	0329	0330	0331	0332	0333	0334	0335
15	0336	0337	0338	0339	0340	0341	0342	0343	0344	0345	0346	0347	0348	0349	0350	0351
16	0352	0353	0354	0355	0356	0357	0358	0359	0360	0361	0362	0363	0364	0365	0366	0367
17	0368	0369	0370	0371	0372	0373	0374	0375	0376	0377	0378	0379	0380	0381	0382	0383
18	0384	0385	0386	0387	0388	0389	0390	0391	0392	0393	0394	0395	0396	0397	0398	0399
19	0400	0401	0402	0403	0404	0405	0406	0407	0408	0409	0410	0411	0412	0413	0414	0415
1A	0416	0417	0418	0419	0420	0421	0422	0423	0424	0425	0426	0427	0428	0429	0430	0431
1B	0432	0433	0434	0435	0436	0437	0438	0439	0440	0441	0442	0443	0444	0445	0446	0447
1C	0448	0449	0450	0451	0452	0453	0454	0455	0456	0457	0458	0459	0460	0461	0462	0463
1D	0464	0465	0466	0467	0468	0469	0470	0471	0472	0473	0474	0475	0476	0477	0478	0479
1E	0480	0481	0482	0483	0484	0485	0486	0487	0488	0489	0490	0491	0492	0493	0494	0495
1F	0496	0497	0498	0499	0500	0501	0502	0503	0504	0505	0506	0507	0508	0509	0510	0511
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
20	0512	0513	0514	0515	0516	0517	0518	0519	0520	0521	0522	0523	0524	0525	0526	0527
21	0528	0529	0530	0531	0532	0533	0534	0535	0536	0537	0538	0539	0540	0541	0542	0543
22	0544	0545	0546	0547	0548	0549	0550	0551	0552	0553	0554	0555	0556	0557	0558	0559
23	0560	0561	0562	0563	0564	0565	0566	0567	0568	0569	0570	0571	0572	0573	0574	0575
24	0576	0577	0578	0579	0580	0581	0582	0583	0584	0585	0586	0587	0588	0589	0590	0591
25	0592	0593	0594	0595	0596	0597	0598	0599	0600	0601	0602	0603	0604	0605	0606	0607
26	0608	0609	0610	0611	0612	0613	0614	0615	0616	0617	0618	0619	0620	0621	0622	0623
27	0624	0625	0626	0627	0628	0629	0630	0631	0632	0633	0634	0635	0636	0637	0638	0639
28	0640	0641	0642	0643	0644	0645	0646	0647	0648	0649	0650	0651	0652	0653	0654	0655
29	0656	0657	0658	0659	0660	0661	0662	0663	0664	0665	0666	0667	0668	0669	0670	0671
2A	0672	0673	0674	0675	0676	0677	0678	0679	0680	0681	0682	0683	0684	0685	0686	0687
2B	0688	0689	0690	0691	0692	0693	0694	0695	0696	0697	0698	0699	0700	0701	0702	0703
2C	0704	0705	0706	0707	0708	0709	0710	0711	0712	0713	0714	0715	0716	0717	0718	0719
2D	0720	0721	0722	0723	0724	0725	0726	0727	0728	0729	0730	0731	0732	0733	0734	0735
2E	0736	0737	0738	0739	0740	0741	0742	0743	0744	0745	0746	0747	0748	0749	0750	0751
2F	0752	0753	0754	0755	0756	0757	0758	0759	0760	0761	0762	0763	0764	0765	0766	0767
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
30	0768	0769	0770	0771	0772	0773	0774	0775	0776	0777	0778	0779	0780	0781	0782	0783
31	0784	0785	0786	0787	0788	0789	0790	0791	0792	0793	0794	0795	0796	0797	0798	0799
32	0800	0801	0802	0803	0804	0805	0806	0807	0808	0809	0810	0811	0812	0813	0814	0815
33	0816	0817	0818	0819	0820	0821	0822	0823	0824	0825	0826	0827	0828	0829	0830	0831
34	0832	0833	0834	0835	0836	0837	0838	0839	0840	0841	0842	0843	0844	0845	0846	0847
35	0848	0849	0850	0851	0852	0853	0854	0855	0856	0857	0858	0859	0860	0861	0862	0863
36	0864	0865	0866	0867	0868	0869	0870	0871	0872	0873	0874	0875	0876	0877	0878	0879
37	0880	0881	0882	0883	0884	0885	0886	0887	0888	0889	0890	0891	0892	0893	0894	0895
38	0896	0897	0898	0899	0900	0901	0902	0903	0904	0905	0906	0907	0908	0909	0910	0911
39	0912	0913	0914	0915	0916	0917	0918	0919	0920	0921	0922	0923	0924	0925	0926	0927
3A	0928	0929	0930	0931	0932	0933	0934	0935	0936	0937	0938	0939	0940	0941	0942	0943
3B	0944	0945	0946	0947	0948	0949	0950	0951	0952	0953	0954	0955	0956	0957	0958	0959
3C	0960	0961	0962	0963	0964	0965	0966	0967	0968	0969	0970	0971	0972	0973	0974	0975
3D	0976	0977	0978	0979	0980	0981	0982	0983	0984	0985	0986	0987	0988	0989	0990	0991
3E	0992	0993	0994	0995	0996	0997	0998	0999	1000	1001	1002	1003	1004	1005	1006	1007
3F	1008	1009	1010	1011	1012	1013	1014	1015	1016	1017	1018	1019	1020	1021	1022	1023

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
40	1024	1025	1026	1027	1028	1029	1030	1031	1032	1033	1034	1035	1036	1037	1038	1039
41	1040	1041	1042	1043	1044	1045	1046	1047	1048	1049	1050	1051	1052	1053	1054	1055
42	1056	1057	1058	1059	1060	1061	1062	1063	1064	1065	1066	1067	1068	1069	1070	1071
43	1072	1073	1074	1075	1076	1077	1078	1079	1080	1081	1082	1083	1084	1085	1086	1087
44	1088	1089	1090	1091	1092	1093	1094	1095	1096	1097	1098	1099	1100	1101	1102	1103
45	1104	1105	1106	1107	1108	1109	1110	1111	1112	1113	1114	1115	1116	1117	1118	1119
46	1120	1121	1122	1123	1124	1125	1126	1127	1128	1129	1130	1131	1132	1133	1134	1135
47	1136	1137	1138	1139	1140	1141	1142	1143	1144	1145	1146	1147	1148	1149	1150	1151
48	1152	1153	1154	1155	1156	1157	1158	1159	1160	1161	1162	1163	1164	1165	1166	1167
49	1168	1169	1170	1171	1172	1173	1174	1175	1176	1177	1178	1179	1180	1181	1182	1183
4A	1184	1185	1186	1187	1188	1189	1190	1191	1192	1193	1194	1195	1196	1197	1198	1199
4B	1200	1201	1202	1203	1204	1205	1206	1207	1208	1209	1210	1211	1212	1213	1214	1215
4C	1216	1217	1218	1219	1220	1221	1222	1223	1224	1225	1226	1227	1228	1229	1230	1231
4D	1232	1233	1234	1235	1236	1237	1238	1239	1240	1241	1242	1243	1244	1245	1246	1247
4E	1248	1249	1250	1251	1252	1253	1254	1255	1256	1257	1258	1259	1260	1261	1262	1263
4F	1264	1265	1266	1267	1268	1269	1270	1271	1272	1273	1274	1275	1276	1277	1278	1279
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
50	1280	1281	1282	1283	1284	1285	1286	1287	1288	1289	1290	1291	1292	1293	1294	1295
51	1296	1297	1298	1299	1300	1301	1302	1303	1304	1305	1306	1307	1308	1309	1310	1311
52	1312	1313	1314	1315	1316	1317	1318	1319	1320	1321	1322	1323	1324	1325	1326	1327
53	1328	1329	1330	1331	1332	1333	1334	1335	1336	1337	1338	1339	1340	1341	1342	1343
54	1344	1345	1346	1347	1348	1349	1350	1351	1352	1353	1354	1355	1356	1357	1358	1359
55	1360	1361	1362	1363	1364	1365	1366	1367	1368	1369	1370	1371	1372	1373	1374	1375
56	1376	1377	1378	1379	1380	1381	1382	1383	1384	1385	1386	1387	1388	1389	1390	1391
57	1392	1393	1394	1395	1396	1397	1398	1399	1400	1401	1402	1403	1404	1405	1406	1407
58	1408	1409	1410	1411	1412	1413	1414	1415	1416	1417	1418	1419	1420	1421	1422	1423
59	1424	1425	1426	1427	1428	1429	1430	1431	1432	1433	1434	1435	1436	1437	1438	1439
5A	1440	1441	1442	1443	1444	1445	1446	1447	1448	1449	1450	1451	1452	1453	1454	1455
5B	1456	1457	1458	1459	1460	1461	1462	1463	1464	1465	1466	1467	1468	1469	1470	1471
5C	1472	1473	1474	1475	1476	1477	1478	1479	1480	1481	1482	1483	1484	1485	1486	1487
5D	1488	1489	1490	1491	1492	1493	1494	1495	1496	1497	1498	1499	1500	1501	1502	1503
5E	1504	1505	1506	1507	1508	1509	1510	1511	1512	1513	1514	1515	1516	1517	1518	1519
5F	1520	1521	1522	1523	1524	1525	1526	1527	1528	1529	1530	1531	1532	1533	1534	1535
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
60	1536	1537	1538	1539	1540	1541	1542	1543	1544	1545	1546	1547	1548	1549	1550	1551
61	1552	1553	1554	1555	1556	1557	1558	1559	1560	1561	1562	1563	1564	1565	1566	1567
62	1568	1569	1570	1571	1572	1573	1574	1575	1576	1577	1578	1579	1580	1581	1582	1583
63	1584	1585	1586	1587	1588	1589	1590	1591	1592	1593	1594	1595	1596	1597	1598	1599
64	1600	1601	1602	1603	1604	1605	1606	1607	1608	1609	1610	1611	1612	1613	1614	1615
65	1616	1617	1618	1619	1620	1621	1622	1623	1624	1625	1626	1627	1628	1629	1630	1631
66	1632	1633	1634	1635	1636	1637	1638	1639	1640	1641	1642	1643	1644	1645	1646	1647
67	1648	1649	1650	1651	1652	1653	1654	1655	1656	1657	1658	1659	1660	1661	1662	1663
68	1664	1665	1666	1667	1668	1669	1670	1671	1672	1673	1674	1675	1676	1677	1678	1679
69	1680	1681	1682	1683	1684	1685	1686	1687	1688	1689	1690	1691	1692	1693	1694	1695
6A	1696	1697	1698	1699	1700	1701	1702	1703	1704	1705	1706	1707	1708	1709	1710	1711
6B	1712	1713	1714	1715	1716	1717	1718	1719	1720	1721	1722	1723	1724	1725	1726	1727
6C	1728	1729	1730	1731	1732	1733	1734	1735	1736	1737	1738	1739	1740	1741	1742	1743
6D	1744	1745	1746	1747	1748	1749	1750	1751	1752	1753	1754	1755	1756	1757	1758	1759
6E	1760	1761	1762	1763	1764	1765	1766	1767	1768	1769	1770	1771	1772	1773	1774	1775
6F	1776	1777	1778	1779	1780	1781	1782	1783	1784	1785	1786	1787	1788	1789	1790	1791
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
70	1792	1793	1794	1795	1796	1797	1798	1799	1800	1801	1802	1803	1804	1805	1806	1807
71	1808	1809	1810	1811	1812	1813	1814	1815	1816	1817	1818	1819	1820	1821	1822	1823
72	1824	1825	1826	1827	1828	1829	1830	1831	1832	1833	1834	1835	1836	1837	1838	1839
73	1840	1841	1842	1843	1844	1845	1846	1847	1848	1849	1850	1851	1852	1853	1854	1855
74	1856	1857	1858	1859	1860	1861	1862	1863	1864	1865	1866	1867	1868	1869	1870	1871
75	1872	1873	1874	1875	1876	1877	1878	1879	1880	1881	1882	1883	1884	1885	1886	1887
76	1888	1889	1890	1891	1892	1893	1894	1895	1896	1897	1898	1899	1900	1901	1902	1903
77	1904	1905	1906	1907	1908	1909	1910	1911	1912	1913	1914	1915	1916	1917	1918	1919
78	1920	1921	1922	1923	1924	1925	1926	1927	1928	1929	1930	1931	1932	1933	1934	1935
79	1936	1937	1938	1939	1940	1941	1942	1943	1944	1945	1946	1947	1948	1949	1950	1951
7A	1952	1953	1954	1955	1956	1957	1958	1959	1960	1961	1962	1963	1964	1965	1966	1967
7B	1968	1969	1970	1971	1972	1973	1974	1975	1976	1977	1978	1979	1980	1981	1982	1983
7C	1984	1985	1986	1987	1988	1989	1990	1991	1992	1993	1994	1995	1996	1997	1998	1999
7D	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013	2014	2015
7E	2016	2017	2018	2019	2020	2021	2022	2023	2024	2025	2026	2027	2028	2029	2030	2031
7F	2032	2033	2034	2035	2036	2037	2038	2039	2040	2041	2042	2043	2044	2045	2046	2047

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
80	2048	2049	2050	2051	2052	2053	2054	2055	2056	2057	2058	2059	2060	2061	2062	2063
81	2064	2065	2066	2067	2068	2069	2070	2071	2072	2073	2074	2075	2076	2077	2078	2079
82	2080	2081	2082	2083	2084	2085	2086	2087	2088	2089	2090	2091	2092	2093	2094	2095
83	2096	2097	2098	2099	2100	2101	2102	2103	2104	2105	2106	2107	2108	2109	2110	2111
84	2112	2113	2114	2115	2116	2117	2118	2119	2120	2121	2122	2123	2124	2125	2126	2127
85	2128	2129	2130	2131	2132	2133	2134	2135	2136	2137	2138	2139	2140	2141	2142	2143
86	2144	2145	2146	2147	2148	2149	2150	2151	2152	2153	2154	2155	2156	2157	2158	2159
87	2160	2161	2162	2163	2164	2165	2166	2167	2168	2169	2170	2171	2172	2173	2174	2175
88	2176	2177	2178	2179	2180	2181	2182	2183	2184	2185	2186	2187	2188	2189	2190	2191
89	2192	2193	2194	2195	2196	2197	2198	2199	2200	2201	2202	2203	2204	2205	2206	2207
8A	2208	2209	2210	2211	2212	2213	2214	2215	2216	2217	2218	2219	2220	2221	2222	2223
8B	2224	2225	2226	2227	2228	2229	2230	2231	2232	2233	2234	2235	2236	2237	2238	2239
8C	2240	2241	2242	2243	2244	2245	2246	2247	2248	2249	2250	2251	2252	2253	2254	2255
8D	2256	2257	2258	2259	2260	2261	2262	2263	2264	2265	2266	2267	2268	2269	2270	2271
8E	2272	2273	2274	2275	2276	2277	2278	2279	2280	2281	2282	2283	2284	2285	2286	2287
8F	2288	2289	2290	2291	2292	2293	2294	2295	2296	2297	2298	2299	2300	2301	2302	2303
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
90	2304	2305	2306	2307	2308	2309	2310	2311	2312	2313	2314	2315	2316	2317	2318	2319
91	2320	2321	2322	2323	2324	2325	2326	2327	2328	2329	2330	2331	2332	2333	2334	2335
92	2336	2337	2338	2339	2340	2341	2342	2343	2344	2345	2346	2347	2348	2349	2350	2351
93	2352	2353	2354	2355	2356	2357	2358	2359	2360	2361	2362	2363	2364	2365	2366	2367
94	2368	2369	2370	2371	2372	2373	2374	2375	2376	2377	2378	2379	2380	2381	2382	2383
95	2384	2385	2386	2387	2388	2389	2390	2391	2392	2393	2394	2395	2396	2397	2398	2399
96	2400	2401	2402	2403	2404	2405	2406	2407	2408	2409	2410	2411	2412	2413	2414	2415
97	2416	2417	2418	2419	2420	2421	2422	2423	2424	2425	2426	2427	2428	2429	2430	2431
98	2432	2433	2434	2435	2436	2437	2438	2439	2440	2441	2442	2443	2444	2445	2446	2447
99	2448	2449	2450	2451	2452	2453	2454	2455	2456	2457	2458	2459	2460	2461	2462	2463
9A	2464	2465	2466	2467	2468	2469	2470	2471	2472	2473	2474	2475	2476	2477	2478	2479
9B	2480	2481	2482	2483	2484	2485	2486	2487	2488	2489	2490	2491	2492	2493	2494	2495
9C	2496	2497	2498	2499	2500	2501	2502	2503	2504	2505	2506	2507	2508	2509	2510	2511
9D	2512	2513	2514	2515	2516	2517	2518	2519	2520	2521	2522	2523	2524	2525	2526	2527
9E	2528	2529	2530	2531	2532	2533	2534	2535	2536	2537	2538	2539	2540	2541	2542	2543
9F	2544	2545	2546	2547	2548	2549	2550	2551	2552	2553	2554	2555	2556	2557	2558	2559
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
A0	2560	2561	2562	2563	2564	2565	2566	2567	2568	2569	2570	2571	2572	2573	2574	2575
A1	2576	2577	2578	2579	2580	2581	2582	2583	2584	2585	2586	2587	2588	2589	2590	2591
A2	2592	2593	2594	2595	2596	2597	2598	2599	2600	2601	2602	2603	2604	2605	2606	2607
A3	2608	2609	2610	2611	2612	2613	2614	2615	2616	2617	2618	2619	2620	2621	2622	2623
A4	2624	2625	2626	2627	2628	2629	2630	2631	2632	2633	2634	2635	2636	2637	2638	2639
A5	2640	2641	2642	2643	2644	2645	2646	2647	2648	2649	2650	2651	2652	2653	2654	2655
A6	2656	2657	2658	2659	2660	2661	2662	2663	2664	2665	2666	2667	2668	2669	2670	2671
A7	2672	2673	2674	2675	2676	2677	2678	2679	2680	2681	2682	2683	2684	2685	2686	2687
A8	2688	2689	2690	2691	2692	2693	2694	2695	2696	2697	2698	2699	2700	2701	2702	2703
A9	2704	2705	2706	2707	2708	2709	2710	2711	2712	2713	2714	2715	2716	2717	2718	2719
AA	2720	2721	2722	2723	2724	2725	2726	2727	2728	2729	2730	2731	2732	2733	2734	2735
AB	2736	2737	2738	2739	2740	2741	2742	2743	2744	2745	2746	2747	2748	2749	2750	2751
AC	2752	2753	2754	2755	2756	2757	2758	2759	2760	2761	2762	2763	2764	2765	2766	2767
AD	2768	2769	2770	2771	2772	2773	2774	2775	2776	2777	2778	2779	2780	2781	2782	2783
AE	2784	2785	2786	2787	2788	2789	2790	2791	2792	2793	2794	2795	2796	2797	2798	2799
AF	2800	2801	2802	2803	2804	2805	2806	2807	2808	2809	2810	2811	2812	2813	2814	2815
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
B0	2816	2817	2818	2819	2820	2821	2822	2823	2824	2825	2826	2827	2828	2829	2830	2831
B1	2832	2833	2834	2835	2836	2837	2838	2839	2840	2841	2842	2843	2844	2845	2846	2847
B2	2848	2849	2850	2851	2852	2853	2854	2855	2856	2857	2858	2859	2860	2861	2862	2863
B3	2864	2865	2866	2867	2868	2869	2870	2871	2872	2873	2874	2875	2876	2877	2878	2879
B4	2880	2881	2882	2883	2884	2885	2886	2887	2888	2889	2890	2891	2892	2893	2894	2895
B5	2896	2897	2898	2899	2900	2901	2902	2903	2904	2905	2906	2907	2908	2909	2910	2911
B6	2912	2913	2914	2915	2916	2917	2918	2919	2920	2921	2922	2923	2924	2925	2926	2927
B7	2928	2929	2930	2931	2932	2933	2934	2935	2936	2937	2938	2939	2940	2941	2942	2943
B8	2944	2945	2946	2947	2948	2949	2950	2951	2952	2953	2954	2955	2956	2957	2958	2959
B9	2960	2961	2962	2963	2964	2965	2966	2967	2968	2969	2970	2971	2972	2973	2974	2975
BA	2976	2977	2978	2979	2980	2981	2982	2983	2984	2985	2986	2987	2988	2989	2990	2991
BB	2992	2993	2994	2995	2996	2997	2998	2999	3000	3001	3002	3003	3004	3005	3006	3007
BC	3008	3009	3010	3011	3012	3013	3014	3015	3016	3017	3018	3019	3020	3021	3022	3023
BD	3024	3025	3026	3027	3028	3029	3030	3031	3032	3033	3034	3035	3036	3037	3038	3039
BE	3040	3041	3042	3043	3044	3045	3046	3047	3048	3049	3050	3051	3052	3053	3054	3055
BF	3056	3057	3058	3059	3060	3061	3062	3063	3064	3065	3066	3067	3068	3069	3070	3071

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
C0	3072	3073	3074	3075	3076	3077	3078	3079	3080	3081	3082	3083	3084	3085	3086	3087
C1	3088	3089	3090	3091	3092	3093	3094	3095	3096	3097	3098	3099	3100	3101	3102	3103
C2	3104	3105	3106	3107	3108	3109	3110	3111	3112	3113	3114	3115	3116	3117	3118	3119
C3	3120	3121	3122	3123	3124	3125	3126	3127	3128	3129	3130	3131	3132	3133	3134	3135
C4	3136	3137	3138	3139	3140	3141	3142	3143	3144	3145	3146	3147	3148	3149	3150	3151
C5	3152	3153	3154	3155	3156	3157	3158	3159	3160	3161	3162	3163	3164	3165	3166	3167
C6	3168	3169	3170	3171	3172	3173	3174	3175	3176	3177	3178	3179	3180	3181	3182	3183
C7	3184	3185	3186	3187	3188	3189	3190	3191	3192	3193	3194	3195	3196	3197	3198	3199
C8	3200	3201	3202	3203	3204	3205	3206	3207	3208	3209	3210	3211	3212	3213	3214	3215
C9	3216	3217	3218	3219	3220	3221	3222	3223	3224	3225	3226	3227	3228	3229	3230	3231
CA	3232	3233	3234	3235	3236	3237	3238	3239	3240	3241	3242	3243	3244	3245	3246	3247
CB	3248	3249	3250	3251	3252	3253	3254	3255	3256	3257	3258	3259	3260	3261	3262	3263
CC	3264	3265	3266	3267	3268	3269	3270	3271	3272	3273	3274	3275	3276	3277	3278	3279
CD	3280	3281	3282	3283	3284	3285	3286	3287	3288	3289	3290	3291	3292	3293	3294	3295
CE	3296	3297	3298	3299	3300	3301	3302	3303	3304	3305	3306	3307	3308	3309	3310	3311
CF	3312	3313	3314	3315	3316	3317	3318	3319	3320	3321	3322	3323	3324	3325	3326	3327
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
D0	3328	3329	3330	3331	3332	3333	3334	3335	3336	3337	3338	3339	3340	3341	3342	3343
D1	3344	3345	3346	3347	3348	3349	3350	3351	3352	3353	3354	3355	3356	3357	3358	3359
D2	3360	3361	3362	3363	3364	3365	3366	3367	3368	3369	3370	3371	3372	3373	3374	3375
D3	3376	3377	3378	3379	3380	3381	3382	3383	3384	3385	3386	3387	3388	3389	3390	3391
D4	3392	3393	3394	3395	3396	3397	3398	3399	3400	3401	3402	3403	3404	3405	3406	3407
D5	3408	3409	3410	3411	3412	3413	3414	3415	3416	3417	3418	3419	3420	3421	3422	3423
D6	3424	3425	3426	3427	3428	3429	3430	3431	3432	3433	3434	3435	3436	3437	3438	3439
D7	3440	3441	3442	3443	3444	3445	3446	3447	3448	3449	3450	3451	3452	3453	3454	3455
D8	3456	3457	3458	3459	3460	3461	3462	3463	3464	3465	3466	3467	3468	3469	3470	3471
D9	3472	3473	3474	3475	3476	3477	3478	3479	3480	3481	3482	3483	3484	3485	3486	3487
DA	3488	3489	3490	3491	3492	3493	3494	3495	3496	3497	3498	3499	3500	3501	3502	3503
DB	3504	3505	3506	3507	3508	3509	3510	3511	3512	3513	3514	3515	3516	3517	3518	3519
DC	3520	3521	3522	3523	3524	3525	3526	3527	3528	3529	3530	3531	3532	3533	3534	3535
DD	3536	3537	3538	3539	3540	3541	3542	3543	3544	3545	3546	3547	3548	3549	3550	3551
DE	3552	3553	3554	3555	3556	3557	3558	3559	3560	3561	3562	3563	3564	3565	3566	3567
DF	3568	3569	3570	3571	3572	3573	3574	3575	3576	3577	3578	3579	3580	3581	3582	3583
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
E0	3584	3585	3586	3587	3588	3589	3590	3591	3592	3593	3594	3595	3596	3597	3598	3599
E1	3600	3601	3602	3603	3604	3605	3606	3607	3608	3609	3610	3611	3612	3613	3614	3615
E2	3616	3617	3618	3619	3620	3621	3622	3623	3624	3625	3626	3627	3628	3629	3630	3631
E3	3632	3633	3634	3635	3636	3637	3638	3639	3640	3641	3642	3643	3644	3645	3646	3647
E4	3648	3649	3650	3651	3652	3653	3654	3655	3656	3657	3658	3659	3660	3661	3662	3663
E5	3664	3665	3666	3667	3668	3669	3670	3671	3672	3673	3674	3675	3676	3677	3678	3679
E6	3680	3681	3682	3683	3684	3685	3686	3687	3688	3689	3690	3691	3692	3693	3694	3695
E7	3696	3697	3698	3699	3700	3701	3702	3703	3704	3705	3706	3707	3708	3709	3710	3711
E8	3712	3713	3714	3715	3716	3717	3718	3719	3720	3721	3722	3723	3724	3725	3726	3727
E9	3728	3729	3730	3731	3732	3733	3734	3735	3736	3737	3738	3739	3740	3741	3742	3743
EA	3744	3745	3746	3747	3748	3749	3750	3751	3752	3753	3754	3755	3756	3757	3758	3759
EB	3760	3761	3762	3763	3764	3765	3766	3767	3768	3769	3770	3771	3772	3773	3774	3775
EC	3776	3777	3778	3779	3780	3781	3782	3783	3784	3785	3786	3787	3788	3789	3790	3791
ED	3792	3793	3794	3795	3796	3797	3798	3799	3800	3801	3802	3803	3804	3805	3806	3807
EE	3808	3809	3810	3811	3812	3813	3814	3815	3816	3817	3818	3819	3820	3821	3822	3823
EF	3824	3825	3826	3827	3828	3829	3830	3831	3832	3833	3834	3835	3836	3837	3838	3839
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
F0	3840	3841	3842	3843	3844	3845	3846	3847	3848	3849	3850	3851	3852	3853	3854	3855
F1	3856	3857	3858	3859	3860	3861	3862	3863	3864	3865	3866	3867	3868	3869	3870	3871
F2	3872	3873	3874	3875	3876	3877	3878	3879	3880	3881	3882	3883	3884	3885	3886	3887
F3	3888	3889	3890	3891	3892	3893	3894	3895	3896	3897	3898	3899	3900	3901	3902	3903
F4	3904	3905	3906	3907	3908	3909	3910	3911	3912	3913	3914	3915	3916	3917	3918	3919
F5	3920	3921	3922	3923	3924	3925	3926	3927	3928	3929	3930	3931	3932	3933	3934	3935
F6	3936	3937	3938	3939	3940	3941	3942	3943	3944	3945	3946	3947	3948	3949	3950	3951
F7	3952	3953	3954	3955	3956	3957	3958	3959	3960	3961	3962	3963	3964	3965	3966	3967
F8	3968	3969	3970	3971	3972	3973	3974	3975	3976	3977	3978	3979	3980	3981	3982	3983
F9	3984	3985	3986	3987	3988	3989	3990	3991	3992	3993	3994	3995	3996	3997	3998	3999
FA	4000	4001	4002	4003	4004	4005	4006	4007	4008	4009	4010	4011	4012	4013	4014	4015
FB	4016	4017	4018	4019	4020	4021	4022	4023	4024	4025	4026	4027	4028	4029	4030	4031
FC	4032	4033	4034	4035	4036	4037	4038	4039	4040	4041	4042	4043	4044	4045	4046	4047
FD	4048	4049	4050	4051	4052	4053	4054	4055	4056	4057	4058	4059	4060	4061	4062	4063
FE	4064	4065	4066	4067	4068	4069	4070	4071	4072	4073	4074	4075	4076	4077	4078	4079
FF	4080	4081	4082	4083	4084	4085	4086	4087	4088	4089	4090	4091	4092	4093	4094	4095

APPENDIX E
OCTAL/DECIMAL TABLE

		0 1 2 3 4 5 6 7								0 1 2 3 4 5 6 7							
		0000 0001 0002 0003 0004 0005 0006 0007								0400 0256 0257 0258 0259 0260 0261 0262 0263							
0000 to 0777 (Octal)		0010 0008 0009 0010 0011 0012 0013 0014 0015								0410 0264 0265 0266 0267 0268 0269 0270 0271							
0511 (Decimal)		0020 0016 0017 0018 0019 0020 0021 0022 0023								0420 0272 0273 0274 0275 0276 0277 0278 0279							
Octal Decimal		0030 0024 0025 0026 0027 0028 0029 0030 0031								0430 0280 0281 0282 0283 0284 0285 0286 0287							
10000 . 4096		0040 0032 0033 0034 0035 0036 0037 0038 0039								0440 0288 0289 0290 0291 0232 0293 0294 0295							
20000 . 8192		0050 0040 0041 0042 0043 0044 0045 0046 0047								0450 0296 0297 0298 0299 0300 0301 0302 0303							
30000 . 12288		0060 0048 0049 0050 0051 0052 0053 0054 0055								0460 0304 0305 0306 0307 0308 0309 0310 0311							
40000 . 16384		0070 0056 0057 0058 0059 0060 0061 0062 0063								0470 0312 0313 0314 0315 0316 0317 0318 0319							
50000 . 20480		0100 0064 0065 0066 0067 0068 0069 0070 0071								0500 0320 0321 0322 0323 0324 0325 0326 0327							
60000 . 24576		0110 0072 0073 0074 0075 0076 0077 0078 0079								0510 0328 0329 0330 0331 0332 0333 0334 0335							
70000 . 28672		0120 0080 0081 0082 0083 0084 0085 0086 0087								0520 0336 0337 0338 0339 0340 0341 0342 0343							
0130 0088 0089 0090 0091 0092 0093 0094 0095		0140 0096 0097 0098 0099 0100 0101 0102 0103								0530 0344 0345 0346 0347 0348 0349 0350 0351							
0150 0104 0105 0106 0107 0108 0109 0110 0111		0160 0112 0113 0114 0115 0116 0117 0118 0119								0550 0360 0361 0362 0363 0364 0365 0366 0367							
0170 0120 0121 0122 0123 0124 0125 0126 0127		0200 0128 0129 0130 0131 0132 0133 0134 0135								0600 0384 0385 0386 0387 0388 0389 0390 0391							
0210 0136 0137 0138 0139 0140 0141 0142 0143		0220 0144 0145 0146 0147 0148 0149 0150 0151								0610 0392 0393 0394 0395 0396 0397 0398 0399							
0230 0152 0153 0154 0155 0156 0157 0158 0159		0240 0160 0161 0162 0163 0164 0165 0166 0167								0620 0400 0401 0402 0403 0404 0405 0406 0407							
0250 0168 0169 0170 0171 0172 0173 0174 0175		0260 0176 0177 0178 0179 0180 0181 0182 0183								0650 0424 0425 0426 0427 0428 0429 0430 0431							
0270 0184 0185 0186 0187 0188 0189 0190 0191		0300 0192 0193 0194 0195 0196 0197 0198 0199								0670 0440 0441 0442 0443 0444 0445 0446 0447							
0310 0200 0201 0202 0203 0204 0205 0206 0207		0320 0208 0209 0210 0211 0212 0213 0214 0215								0700 0448 0449 0450 0451 0452 0453 0454 0455							
0330 0216 0217 0218 0219 0220 0221 0222 0223		0340 0224 0225 0226 0227 0228 0229 0230 0231								0710 0456 0457 0458 0459 0460 0461 0462 0463							
0350 0232 0233 0234 0235 0236 0237 0238 0239		0360 0240 0241 0242 0243 0244 0245 0246 0247								0720 0464 0465 0466 0467 0468 0469 0470 0471							
0370 0248 0249 0250 0251 0252 0253 0254 0255		0400 0512 0513 0514 0515 0516 0517 0518 0519								0730 0472 0473 0474 0475 0476 0477 0478 0479							
		0 1 2 3 4 5 6 7								0 1 2 3 4 5 6 7							
1000 to 1777 (Octal)		1010 0520 0521 0522 0523 0524 0525 0526 0527								1400 0768 0769 0770 0771 0772 0773 0774 0775							
0512 to 1023 (Decimal)		1020 0528 0529 0530 0531 0532 0533 0534 0535								1410 0776 0777 0778 0779 0780 0781 0782 0783							
1030 0536 0537 0538 0539 0540 0541 0542 0543		1040 0544 0545 0546 0547 0548 0549 0550 0551								1420 0784 0785 0786 0787 0788 0789 0790 0791							
1050 0552 0553 0554 0555 0556 0557 0558 0559		1060 0560 0561 0562 0563 0564 0565 0566 0567								1430 0792 0793 0794 0795 0796 0797 0798 0799							
1070 0568 0569 0570 0571 0572 0573 0574 0575		1100 0576 0577 0578 0579 0580 0581 0582 0583								1440 0800 0801 0802 0803 0804 0805 0806 0807							
1110 0584 0585 0586 0587 0588 0589 0590 0591		1120 0592 0593 0594 0595 0596 0597 0598 0599								1450 0808 0809 0810 0811 0812 0813 0814 0815							
1130 0600 0601 0602 0603 0604 0605 0606 0607		1140 0608 0609 0610 0611 0612 0613 0614 0615								1460 0816 0817 0818 0819 0820 0821 0822 0823							
1150 0616 0617 0618 0619 0620 0621 0622 0623		1160 0624 0625 0626 0627 0628 0629 0630 0631								1470 0824 0825 0826 0827 0828 0829 0830 0831							
1170 0632 0633 0634 0635 0636 0637 0638 0639		1200 0640 0641 0642 0643 0644 0645 0646 0647								1500 0832 0833 0834 0835 0836 0837 0838 0839							
1210 0648 0649 0650 0651 0652 0653 0654 0655		1220 0656 0657 0658 0659 0660 0661 0662 0663								1510 0840 0841 0842 0843 0844 0845 0846 0847							
1230 0664 0665 0666 0667 0668 0669 0670 0671		1240 0672 0673 0674 0675 0676 0677 0678 0679								1520 0848 0849 0850 0851 0852 0853 0854 0855							
1250 0680 0681 0682 0683 0684 0685 0686 0687		1260 0688 0689 0690 0691 0692 0693 0694 0695								1530 0856 0857 0858 0859 0860 0861 0862 0863							
1270 0696 0697 0698 0699 0700 0701 0702 0703		1300 0704 0705 0706 0707 0708 0709 0710 0711								1540 0864 0865 0866 0867 0868 0869 0870 0871							
1310 0712 0713 0714 0715 0716 0717 0718 0719		1320 0720 0721 0722 0723 0724 0725 0726 0727								1550 0872 0873 0874 0875 0876 0877 0878 0879							
1330 0728 0729 0730 0731 0732 0733 0734 0735																	

		Octal								Octal										
		0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7			
2000	1024	2000	1024	1025	1026	1027	1028	1029	1030	1031	2400	1280	1281	1282	1283	1284	1285	1286	1287	
to	to	2010	1032	1033	1034	1035	1036	1037	1038	1039	2410	1288	1289	1290	1291	1292	1293	1294	1295	
2777	1535	(Octal)	2020	1040	1041	1042	1043	1044	1045	1046	1047	2420	1296	1297	1298	1299	1300	1301	1302	1303
(Decimal)		2030	1048	1049	1050	1051	1052	1053	1054	1055	2430	1304	1305	1306	1307	1308	1309	1310	1311	
		2040	1056	1057	1058	1059	1060	1061	1062	1063	2440	1312	1313	1314	1315	1316	1317	1318	1319	
		2050	1064	1065	1066	1067	1068	1069	1070	1071	2450	1320	1321	1322	1323	1324	1325	1326	1327	
		2060	1072	1073	1074	1075	1076	1077	1078	1079	2460	1328	1329	1330	1331	1332	1333	1334	1335	
		2070	1080	1081	1082	1083	1084	1085	1086	1087	2470	1336	1337	1338	1339	1340	1341	1342	1343	
Octal	Decimal																			
10000	4096		2100	1088	1089	1090	1091	1092	1093	1094	1095	2500	1344	1345	1346	1347	1348	1349	1350	1351
20000	8192		2110	1096	1097	1098	1099	1100	1101	1102	1103	2510	1352	1353	1354	1355	1356	1357	1358	1359
30000	12288		2120	1104	1105	1106	1107	1108	1109	1110	1111	2520	1360	1361	1362	1363	1364	1365	1366	1367
40000	16384		2130	1112	1113	1114	1115	1116	1117	1118	1119	2530	1368	1369	1370	1371	1372	1373	1374	1375
50000	20480		2140	1120	1121	1122	1123	1124	1125	1126	1127	2540	1376	1377	1378	1379	1380	1381	1382	1383
60000	24576		2150	1128	1129	1130	1131	1132	1133	1134	1135	2550	1384	1385	1386	1387	1388	1389	1390	1391
70000	28672		2160	1136	1137	1138	1139	1140	1141	1142	1143	2560	1392	1393	1394	1395	1396	1397	1398	1399
			2170	1144	1145	1146	1147	1148	1149	1150	1151	2570	1400	1401	1402	1403	1404	1405	1406	1407
			2200	1152	1153	1154	1155	1156	1157	1158	1159	2600	1408	1409	1410	1411	1412	1413	1414	1415
			2210	1160	1161	1162	1163	1164	1165	1166	1167	2610	1416	1417	1418	1419	1420	1421	1422	1423
			2220	1168	1169	1170	1171	1172	1173	1174	1175	2620	1424	1425	1426	1427	1428	1429	1430	1431
			2230	1176	1177	1178	1179	1180	1181	1182	1183	2630	1432	1433	1434	1435	1436	1437	1438	1439
			2240	1184	1185	1186	1187	1188	1189	1190	1191	2640	1440	1441	1442	1443	1444	1445	1446	1447
			2250	1192	1193	1194	1195	1196	1197	1198	1199	2650	1448	1449	1450	1451	1452	1453	1454	1455
			2260	1200	1201	1202	1203	1204	1205	1206	1207	2660	1456	1457	1458	1459	1460	1461	1462	1463
			2270	1208	1209	1210	1211	1212	1213	1214	1215	2670	1464	1465	1466	1467	1468	1469	1470	1471
			2300	1216	1217	1218	1219	1220	1221	1222	1223	2700	1472	1473	1474	1475	1476	1477	1478	1479
			2310	1224	1225	1226	1227	1228	1229	1230	1231	2710	1480	1481	1482	1483	1484	1485	1486	1487
			2320	1232	1233	1234	1235	1236	1237	1238	1239	2720	1488	1489	1490	1491	1492	1493	1494	1495
			2330	1240	1241	1242	1243	1244	1245	1246	1247	2730	1496	1497	1498	1499	1500	1501	1502	1503
			2340	1248	1249	1250	1251	1252	1253	1254	1255	2740	1504	1505	1506	1507	1508	1509	1510	1511
			2350	1256	1257	1258	1259	1260	1261	1262	1263	2750	1512	1513	1514	1515	1516	1517	1518	1519
			2360	1264	1265	1266	1267	1268	1269	1270	1271	2760	1520	1521	1522	1523	1524	1525	1526	1527
			2370	1272	1273	1274	1275	1276	1277	1278	1279	2770	1528	1529	1530	1531	1532	1533	1534	1535
Octal	Decimal																			
3000	1536		3000	1536	1537	1538	1539	1540	1541	1542	1543	3400	1792	1793	1794	1795	1796	1797	1798	1799
to	to		3010	1544	1545	1546	1547	1548	1549	1550	1551	3410	1800	1801	1802	1803	1804	1805	1806	1807
3777	2047	(Octal)	3020	1552	1553	1554	1555	1556	1557	1558	1559	3420	1808	1809	1810	1811	1812	1813	1814	1815
			3030	1560	1561	1562	1563	1564	1565	1566	1567	3430	1816	1817	1818	1819	1820	1821	1822	1823
			3040	1568	1569	1570	1571	1572	1573	1574	1575	3440	1824	1825	1826	1827	1828	1829	1830	1831
			3050	1576	1577	1578	1579	1580	1581	1582	1583	3450	1832	1833	1834	1835	1836	1837	1838	1839
			3060	1584	1585	1586	1587	1588	1589	1590	1591	3460	1840	1841	1842	1843	1844	1845	1846	1847
			3070	1592	1593	1594	1595	1596	1597	1598	1599	3470	1848	1849	1850	1851	1852	1853	1854	1855
			3100	1600	1601	1602	1603	1604	1605	1606	1607	3500	1856	1857	1858	1859	1860	1861	1862	1863
			3110	1608	1609	1610	1611	1612	1613	1614	1615	3510	1864	1865	1866	1867	1868	1869	1870	1871
			3120	1616	1617	1618	1619	1620	1621	1622	1623	3520	1872	1873	1874	1875	1876	1877	1878	1879
			3130	1624	1625	1626	1627	1628	1629	1630	1631	3530	1880	1881	1882	1883	1884	1885	1886	1887
			3140	1632	1633	1634	1635	1636	1637	1638	1639	3540	1888	1889	1890	1891	1892	1893	1894	1895
			3150	1640	1641	1642	1643	1644	1645	1646	1647	3550	1896	1897	1898	1899	1900	1901	1902	1903
			3160	1648	1649	1650	1651	1652	1653	1654	1655	3560	1904	1905	1906	1907	1908	1909	1910	1911
			3170	1656	1657	1658	1659	1660	1661	1662	1663	3570	1912	1913	1914	1915	1916	1917	1918	1919
			3200	1664	1665	1666	1667	1668	1669	1670	1671	3600	1920	1921	1922	1923	1924	1925	1926	1927
			3210	1672	1673	1674	1675	1676	1677	1678	1679	3610	1928	1929	1930	1931	1932	1933	1934	1935
			3220	1680	1681	1682	1683	1684	1685	1686	1687	3620	1936	1937	1938	1939	1940	1941	1942	1943
			3230	1688	1689	1690	1691	1692	1693	1694	1695	3630	1944	1945	1946	1947	1948	1949	1950	1951
			3240	1696	1697	1698	1699	1700	1701	1702	1703	3640	1952	1953	1954	1955	1956	1957	1958	1959
			3250	1704	1705	1706	1707	1708	1709	1710	1711	3650	1960	1961	1962	1963	1964	1965	1966	1967
			3260	1712	1713	1714	1715	1716	1717	1718	1719	3660	1968	1969	1970	1971	1972	1973	1974	1975
			3270	1720	1721	1722	1723	1724	1725	1726	1727	3670	1976	1977	1978	1979	1980	1981	1982	1983
			3300	1728	1729	1730	1731	1732	1733	17										

		Octal							Octal											
		0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7			
4000	2048	4000	2048	2049	2050	2051	2052	2053	2054	2055	4400	2304	2305	2306	2307	2308	2309	2310	2311	
to	to	4010	2056	2057	2058	2059	2060	2061	2062	2063	4410	2312	2313	2314	2315	2316	2317	2318	2319	
4777	2559	(Octal)	4020	2064	2065	2066	2067	2068	2069	2070	2071	4420	2320	2321	2322	2323	2324	2325	2326	2327
(Decimal)		4030	2072	2073	2074	2075	2076	2077	2078	2079	4430	2328	2329	2330	2331	2332	2333	2334	2335	
		4040	2080	2081	2082	2083	2084	2085	2086	2087	4440	2336	2337	2338	2339	2340	2341	2342	2343	
		4050	2088	2089	2090	2091	2092	2093	2094	2095	4450	2344	2345	2346	2347	2348	2349	2350	2351	
Octal	Decimal	4060	2096	2097	2098	2099	2100	2101	2102	2103	4460	2352	2353	2354	2355	2356	2357	2358	2359	
10000	4096	4070	2104	2105	2106	2107	2108	2109	2110	2111	4470	2360	2361	2362	2363	2364	2365	2366	2367	
20000	8192		4100	2112	2113	2114	2115	2116	2117	2118	2119	4500	2368	2369	2370	2371	2372	2373	2374	2375
30000	12288		4110	2120	2121	2122	2123	2124	2125	2126	2127	4510	2376	2377	2378	2379	2380	2381	2382	2383
40000	16384		4120	2128	2129	2130	2131	2132	2133	2134	2135	4520	2384	2385	2386	2387	2388	2389	2390	2391
50000	20480		4130	2136	2137	2138	2139	2140	2141	2142	2143	4530	2392	2393	2394	2395	2396	2397	2398	2399
60000	24576		4140	2144	2145	2146	2147	2148	2149	2150	2151	4540	2400	2401	2402	2403	2404	2405	2406	2407
70000	28672		4150	2152	2153	2154	2155	2156	2157	2158	2159	4550	2408	2409	2410	2411	2412	2413	2414	2415
		4160	2160	2161	2162	2163	2164	2165	2166	2167	4560	2416	2417	2418	2419	2420	2421	2422	2423	
		4170	2168	2169	2170	2171	2172	2173	2174	2175	4570	2424	2425	2426	2427	2428	2429	2430	2431	
		4200	2176	2177	2178	2179	2180	2181	2182	2183	4600	2432	2433	2434	2435	2436	2437	2438	2439	
		4210	2184	2185	2186	2187	2188	2189	2190	2191	4610	2440	2441	2442	2443	2444	2445	2446	2447	
		4220	2192	2193	2194	2195	2196	2197	2198	2199	4620	2448	2449	2450	2451	2452	2453	2454	2455	
		4230	2200	2201	2202	2203	2204	2205	2206	2207	4630	2456	2457	2458	2459	2460	2461	2462	2463	
		4240	2208	2209	2210	2211	2212	2213	2214	2215	4640	2464	2465	2466	2467	2468	2469	2470	2471	
		4250	2216	2217	2218	2219	2220	2221	2222	2223	4650	2472	2473	2474	2475	2476	2477	2478	2479	
		4260	2224	2225	2226	2227	2228	2229	2230	2231	4660	2480	2481	2482	2483	2484	2485	2486	2487	
		4270	2232	2233	2234	2235	2236	2237	2238	2239	4670	2488	2489	2490	2491	2492	2493	2494	2495	
		4300	2240	2241	2242	2243	2244	2245	2246	2247	4700	2496	2497	2498	2499	2500	2501	2502	2503	
		4310	2248	2249	2250	2251	2252	2253	2254	2255	4710	2504	2505	2506	2507	2508	2509	2510	2511	
		4320	2256	2257	2258	2259	2260	2261	2262	2263	4720	2512	2513	2514	2515	2516	2517	2518	2519	
		4330	2264	2265	2266	2267	2268	2269	2270	2271	4730	2520	2521	2522	2523	2524	2525	2526	2527	
		4340	2272	2273	2274	2275	2276	2277	2278	2279	4740	2528	2529	2530	2531	2532	2533	2534	2535	
		4350	2280	2281	2282	2283	2284	2285	2286	2287	4750	2536	2537	2538	2539	2540	2541	2542	2543	
		4360	2288	2289	2290	2291	2292	2293	2294	2295	4760	2544	2545	2546	2547	2548	2549	2550	2551	
		4370	2296	2297	2298	2299	2300	2301	2302	2303	4770	2552	2553	2554	2555	2556	2557	2558	2559	
		Octal							Octal											
		0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7			
5000	2560	5000	2560	2561	2562	2563	2564	2565	2566	2567	5400	2816	2817	2818	2819	2820	2821	2822	2823	
to	to	5010	2568	2569	2570	2571	2572	2573	2574	2575	5410	2824	2825	2826	2827	2828	2829	2830	2831	
5777	3071	(Octal)	5020	2576	2577	2578	2579	2580	2581	2582	5420	2832	2833	2834	2835	2836	2837	2838	2839	
		5030	2584	2585	2586	2587	2588	2589	2590	2591	5430	2840	2841	2842	2843	2844	2845	2846	2847	
		5040	2592	2593	2594	2595	2596	2597	2598	2599	5440	2848	2849	2850	2851	2852	2853	2854	2855	
		5050	2600	2601	2602	2603	2604	2605	2606	2607	5450	2856	2857	2858	2859	2860	2861	2862	2863	
		5060	2608	2609	2610	2611	2612	2613	2614	2615	5460	2864	2865	2866	2867	2868	2869	2870	2871	
		5070	2616	2617	2618	2619	2620	2621	2622	2623	5470	2872	2873	2874	2875	2876	2877	2878	2879	
		5100	2624	2625	2626	2627	2628	2629	2630	2631	5500	2880	2881	2882	2883	2884	2885	2886	2887	
		5110	2632	2633	2634	2635	2636	2637	2638	2639	5510	2888	2889	2890	2891	2892	2893	2894	2895	
		5120	2640	2641	2642	2643	2644	2645	2646	2647	5520	2896	2897	2898	2899	2900	2901	2902	2903	
		5130	2648	2649	2650	2651	2652	2653	2654	2655	5530	2904	2905	2906	2907	2908	2909	2910	2911	
		5140	2656	2657	2658	2659	2660	2661	2662	2663	5540	2912	2913	2914	2915	2916	2917	2918	2919	
		5150	2664	2665	2666	2667	2668	2669	2670	2671	5550	2920	2921	2922	2923	2924	2925	2926	2927	
		5160	2672	2673	2674	2675	2676	2677	2678	2679	5560	2928	2929	2930	2931	2932	2933	2934	2935	
		5170	2680	2681	2682	2683	2684	2685	2686	2687	5570	2936	2937	2938	2939	2940	2941	2942	2943	
		5200	2688	2689	2690	2691	2692	2693	2694	2695	5600	2944	2945	2946	2947	2948	2949	2950	2951	
		5210	2696	2697	2698	2699	2700	2701	2702	2703	5610	2952	2953	2954	2955	2956	2957	2958	2959	
		5220	2704	2705	2706	2707	2708	2709	2710	2711	5620	2960	2961	2962	2963	2964	2965	2966	2967	
		5230	2712	2713	2714	2715	2716	2717	2718	2719	5630	2968	2969	2970	2971	2972	2973	2974	2975	
		5240	2720	2721	2722	2723	2724	2725	2726	2727	5640	2976	2977	2978	2979	2980	2981	2982	2983	
		5250	2728	2729	2730	2731	2732	2733	2734	2735	5650	2984	2985	2986	2987	2988	2989	2990	2991	
		5260	2736	2737	2738	2739	2740	2741	2742	2743	5660	2992	2993	2994	2995	2996	2997	2998	2999	
		5270	2744	2745	2746	2747	2748	2749	2750	2751	5670	3000	3001	3002	3003	3004	3005	3006	3007	
		5300	2752	2753	2754	2755	2756	2757	2758	2759	5700	3008	3009	3010	3011	3012	3013	3014	3015	
		5310	2760	2761	2762	2763	2764	2765	2766	2767	5710	3016	3017	3018	3019	3020	3021	3022	3023	
		5320	2768	2769	2770	2771	2772	2773	2774	2775	5720	3024	3025	3026	3027	3028	3029	3030	3031	
		5330	2776	2777</																

		0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7		
6000 3072 to 6777 3583 (Octal) (Decimal)		6000	3072	3073	3074	3075	3076	3077	3078	3079	6400	3328	3329	3330	3331	3332	3333	3334	3335
		6010	3080	3081	3082	3083	3084	3085	3086	3087	6410	3336	3337	3338	3339	3340	3341	3342	3343
		6020	3088	3089	3090	3091	3092	3093	3094	3095	6420	3344	3345	3346	3347	3348	3349	3350	3351
		6030	3096	3097	3098	3099	3100	3101	3102	3103	6430	3352	3353	3354	3355	3356	3357	3358	3359
		6040	3104	3105	3106	3107	3108	3109	3110	3111	6440	3360	3361	3362	3363	3364	3365	3366	3367
		6050	3112	3113	3114	3115	3116	3117	3118	3119	6450	3368	3369	3370	3371	3372	3373	3374	3375
		6060	3120	3121	3122	3123	3124	3125	3126	3127	6460	3376	3377	3378	3379	3380	3381	3382	3383
		6070	3128	3129	3130	3131	3132	3133	3134	3135	6470	3384	3385	3386	3387	3388	3389	3390	3391
Octal Decimal	10000 . 4096 20000 . 8192 30000 . 12288 40000 . 16384 50000 . 20480 60000 . 24576 70000 . 28672	6100	3136	3137	3138	3139	3140	3141	3142	3143	6500	3392	3393	3394	3395	3396	3397	3398	3399
		6110	3144	3145	3146	3147	3148	3149	3150	3151	6510	3400	3401	3402	3403	3404	3405	3406	3407
		6120	3152	3153	3154	3155	3156	3157	3158	3159	6520	3408	3409	3410	3411	3412	3413	3414	3415
		6130	3160	3161	3162	3163	3164	3165	3166	3167	6530	3416	3417	3418	3419	3420	3421	3422	3423
		6140	3168	3169	3170	3171	3172	3173	3174	3175	6540	3424	3425	3426	3427	3428	3429	3430	3431
		6150	3176	3177	3178	3179	3180	3181	3182	3183	6550	3432	3433	3434	3435	3436	3437	3438	3439
		6160	3184	3185	3186	3187	3188	3189	3190	3191	6560	3440	3441	3442	3443	3444	3445	3446	3447
		6170	3192	3193	3194	3195	3196	3197	3198	3199	6570	3448	3449	3450	3451	3452	3453	3454	3455
		6200	3200	3201	3202	3203	3204	3205	3206	3207	6600	3456	3457	3458	3459	3460	3461	3462	3463
		6210	3208	3209	3210	3211	3212	3213	3214	3215	6610	3464	3465	3466	3467	3468	3469	3470	3471
		6220	3216	3217	3218	3219	3220	3221	3222	3223	6620	3472	3473	3474	3475	3476	3477	3478	3479
		6230	3224	3225	3226	3227	3228	3229	3230	3231	6630	3480	3481	3482	3483	3484	3485	3486	3487
		6240	3232	3233	3234	3235	3236	3237	3238	3239	6640	3488	3489	3490	3491	3492	3493	3494	3495
		6250	3240	3241	3242	3243	3244	3245	3246	3247	6650	3496	3497	3498	3499	3500	3501	3502	3503
		6260	3248	3249	3250	3251	3252	3253	3254	3255	6660	3504	3505	3506	3507	3508	3509	3510	3511
		6270	3256	3257	3258	3259	3260	3261	3262	3263	6670	3512	3513	3514	3515	3516	3517	3518	3519
		6300	3264	3265	3266	3267	3268	3269	3270	3271	6700	3520	3521	3522	3523	3524	3525	3526	3527
		6310	3272	3273	3274	3275	3276	3277	3278	3279	6710	3528	3529	3530	3531	3532	3533	3534	3535
		6320	3280	3281	3282	3283	3284	3285	3286	3287	6720	3536	3537	3538	3539	3540	3541	3542	3543
		6330	3288	3289	3290	3291	3292	3293	3294	3295	6730	3544	3545	3546	3547	3548	3549	3550	3551
		6340	3296	3297	3298	3299	3300	3301	3302	3303	6740	3552	3553	3554	3555	3556	3557	3558	3559
		6350	3304	3305	3306	3307	3308	3309	3310	3311	6750	3560	3561	3562	3563	3564	3565	3566	3567
		6360	3312	3313	3314	3315	3316	3317	3318	3319	6760	3568	3569	3570	3571	3572	3573	3574	3575
		6370	3320	3321	3322	3323	3324	3325	3326	3327	6770	3576	3577	3578	3579	3580	3581	3582	3583
7000 3584 to 7777 4095 (Octal) (Decimal)		7000	3584	3585	3586	3587	3588	3589	3590	3591	7400	3840	3841	3842	3843	3844	3845	3846	3847
		7010	3592	3593	3594	3595	3596	3597	3598	3599	7410	3848	3849	3850	3851	3852	3853	3854	3855
		7020	3600	3601	3602	3603	3604	3605	3606	3607	7420	3856	3857	3858	3859	3860	3861	3862	3863
		7030	3608	3609	3610	3611	3612	3613	3614	3615	7430	3864	3865	3866	3867	3868	3869	3870	3871
		7040	3616	3617	3618	3619	3620	3621	3622	3623	7440	3872	3873	3874	3875	3876	3877	3878	3879
		7050	3624	3625	3626	3627	3628	3629	3630	3631	7450	3880	3881	3882	3883	3884	3885	3886	3887
		7060	3632	3633	3634	3635	3636	3637	3638	3639	7460	3888	3889	3890	3891	3892	3893	3894	3895
		7070	3640	3641	3642	3643	3644	3645	3646	3647	7470	3896	3897	3898	3899	3900	3901	3902	3903
		7100	3648	3649	3650	3651	3652	3653	3654	3655	7500	3904	3905	3906	3907	3908	3909	3910	3911
		7110	3656	3657	3658	3659	3660	3661	3662	3663	7510	3912	3913	3914	3915	3916	3917	3918	3919
		7120	3664	3665	3666	3667	3668	3669	3670	3671	7520	3920	3921	3922	3923	3924	3925	3926	3927
		7130	3672	3673	3674	3675	3676	3677	3678	3679	7530	3928	3929	3930	3931	3932	3933	3934	3935
		7140	3680	3681	3682	3683	3684	3685	3686	3687	7540	3936	3937	3938	3939	3940	3941	3942	3943
		7150	3688	3689	3690	3691	3692	3693	3694	3695	7550	3944	3945	3946	3947	3948	3949	3950	3951
		7160	3696	3697	3698	3699	3700	3701	3702	3703	7560	3952	3953	3954	3955	3956	3957	3958	3959
		7170	3704	3705	3706	3707	3708	3709	3710	3711	7570	3960	3961	3962	3963	3964	3965	3966	3967
		7200	3712	3713	3714	3715	3716	3717	3718	3719	7600	3968	3969	3970	3971	3972	3973	3974	3975
		7210	3720	3721	3722	3723	3724	3725	3726	3727	7610	3976	3977	3978	3979	3980	3981	3982	3983
		7220	3728	3729	3730	3731	3732	3733	3734	3735	7620	3984	3985	3986	3987	3988	3989	3990	3991
		7230	3736	3737	3738	3739	3740	3741	3742	3743	7630	3992	3993	3994	3995	3996	3997	3998	3999
		7240	3744	3745	3746	3747	3748	3749	3750	3751	7640	4000	4001	4002	4003	4004	4005	4006	4007
		7250	3752	3753	3754	3755	3756	3757	3758	3759	7650	4008	4009	4010	4011	4012	4013	4014	4015
		7260	3760	3761	3762	3763	3764	3765	3766	3767	7660	4016	4017	4018	4019	4020	4021	4022	4023
		7270	3768	3769	3770	3771	3772	3773	3774	3775	7670	4024	4025	4026	4027	4028	4029	4030	4031
		7300	3776	3777	3778	3779	3780	3781	3782	3783	7700	4032	4033	4034	4035	4036	4037	4038	4039
		7310	3784	3785	3786	3787	3788	3789	3790	3791	7710	4040	4041	4042	4043	4044	4045	4046	4047
		7320	3792	3793	3794	3795	3796	3797	3798	3799	7720	4048	4049	4050	4051	4052	4053	4054	4055
		7330	3800	3801	3802	3803	3804	3805	3806	3807	7730	4056	4057	4058	4059	4060	4061	4062	4063
		7340	3808	3809	3810	3811	3812	3813	3814	3815	7740	4064	4065	4066	4067	4068	4069	4070	4071
		7350	3816	3817	3818	3819	3820	3821	3822	3823	7750	4072	4073	4074	4075	4076	407		

APPENDIX F
POWERS OF TWO TABLE

POWERS OF TWO TABLE

2^n	n	2^{-n}
1	0	1.0
2	1	0.5
4	2	0.25
8	3	0.125
16	4	0.062 5
32	5	0.031 25
64	6	0.015 625
128	7	0.007 812 5
256	8	0.003 906 25
512	9	0.001 953 125
1 024	10	0.000 976 562 5
2 048	11	0.000 488 281 25
4 096	12	0.000 244 140 625
8 192	13	0.000 122 070 312 5
16 384	14	0.000 061 035 156 25
32 768	15	0.000 030 517 578 125
65 536	16	0.000 015 258 789 062 5
131 072	17	0.000 007 629 394 531 25
262 144	18	0.000 003 814 697 265 625
524 288	19	0.000 001 907 348 632 812 5
1 048 576	20	0.000 000 953 674 316 406 25
2 097 152	21	0.000 000 476 837 158 203 125
4 194 304	22	0.000 000 238 418 579 101 562 5
8 388 608	23	0.000 000 119 209 289 550 781 25
16 777 216	24	0.000 000 059 604 644 775 390 625
33 554 432	25	0.000 000 029 802 322 387 695 312 5
67 108 864	26	0.000 000 014 901 161 193 847 656 25
134 217 728	27	0.000 000 007 450 580 596 923 828 125
268 435 456	28	0.000 000 003 725 290 298 461 914 062 5
536 870 912	29	0.000 000 001 862 645 149 230 957 031 45
1 073 741 824	30	0.000 000 000 931 322 574 615 478 515 625
2 147 483 648	31	0.000 000 000 465 661 287 307 739 257 812 5
4 294 967 296	32	0.000 000 000 232 830 643 653 869 628 906 25
8 589 934 592	33	0.000 000 000 116 415 321 826 934 814 453 125
17 179 869 184	34	0.000 000 000 058 207 660 913 467 407 226 562 5
34 359 738 368	35	0.000 000 000 029 103 830 456 733 703 613 281 25
68 719 476 736	36	0.000 000 000 014 551 915 228 366 851 806 640 625
137 438 953 472	37	0.000 000 000 007 275 957 614 183 425 903 320 312 5
274 877 906 944	38	0.000 000 000 003 637 978 807 091 712 951 660 156 25
549 755 813 888	39	0.000 000 000 001 818 989 403 545 856 475 830 078 125
1 099 511 627 776	40	0.000 000 000 000 909 494 701 772 928 237 915 039 062 5

APPENDIX G

PROGRAM EXAMPLES

```

*          OUTPUT A FIELD OF DATA IF FLAG IS SET
*
*
*          9000      ORG      X'9000'
*          9000  OUTBUF  DS,200      *200 WORD OUTPUT BUFFER
*          8020  DATA    DF         128,32 *MSB=128,32 BIT FIELD
*          0000  FLAG    EQU        0   *BIT COLUMN ZERO
*
*          8010      ORG      BULKC
*          8010  8010  3660C000  AS,X'C000' *SELECT ARRAY #0 & #1
*          8011  8011  02008845  Y,FLAG  *RESPONSE = WORDS TO REAR
*          8012  8012  34A000C8  BL,200  *MAX BUFF SIZE COUNTER
*          8013  8013  32809001  DP,OUTBUF+1 *INDEX TO OUTBUF
*          8014  8014  28E18018  DONE   *EXIT IF NO FLAG SET
*          8015  8015  03F86640  LOOP   *SELECT & CLEAR 1ST Y
*          8016  8016  36000000  STEP   DATA *C REG. = VALUE OF SELECTED
*          8017  8017  27C4A0FD  LC
*
*          8018  8018  30050000  WORD VIA THE LINK POINTER REG.
*          8019  8019  2901801B  SR      C,(DP),3 *STORE COMMON REGISTER
*          801A  801A  28F18015  INTO MEM ADDRESS CONTAINED IN DP, THEN DP=DP+1, *
*          W 801B  801B  30819000  BL = BL-1
*          801C  801C  38012C0C  BZ,BL  DONE *EXIT IF OUTPUT 200 WORDS
*          801D  801D  38002000  BRS    LOOP *LOOP IF MORE Y'S SET
*                               SR      BL,OUTBUF *STORE COUNT IN 1ST
*                               SLOT. SEQ. PROC. MUST COMPUTE 200-COUNT FOR
*                               NUMBER OF ITEMS IN THE OUTPUT BUFFER.
*                               INT,0'300'  12 *INTERRUPT SEQ. PROC.
*                               WAIT   *HALT AP

```

```

* THIS ROUTINE WILL CLEAR TWO ARRAY MEMORIES
* TO ALL ZEROES.
*
8000      ORG          X'8000'
* NOTE THAT WHENEVER THE AP GOES FROM THE INACTIVE
* STATE TO THE ACTIVE STATE, THE INSTRUCTION AT
* LOCATION X'800001 IS EXECUTED.
8000 8000 28018010      B           PROGCLEAR
8010      ORG          BULKC
8010 8010 3660C000 PROGCLEAR LI,2    AS,X'000000' *SELECT ARRAY #0 & #1
8011 8011 33900000      LI          FP1,0   *START WITH BIT COLUMN 0
8012 8012 000007741     CLR         Y       *CLEAR ALL Y
8013 8013 3EFF8014     RPT,256   *EXECUTE NEXT INSTRUCTION 256
*TIMES
*
8014 8014 16300002      S           Y,FP1+  *STORE Y INTO RTT COLUMN
* OF MEMORY REFERENCED BY FP1, THEN FP1 = FP1+1
*
8015 8015 38002000      WAIT        HALT AP

```

* CHECK FOR MULTIPLE RESPONSE
*
*

8010 8010 000088A2
8011 8011 03F86640
8012 8012 28E18013

L X,Y *SAVE RESPONSE IN X
STEP *CLEAR FIRST RESPONDER
BNR ONLYONE * BRANCH IF NO Y'S SET

* MULTIPLE RESPONSE PROCESSING FOLLOWS
*
*

*

* SINGLE RESPONSE PROCESSING FOLLOWS
*
*

8013 8013 00008843 ONLYONE

L Y,X *RESTORE THE SINGLE RESPONDER

*

* EXAMPLE OF PAGING ALTERNATING PROGRAM
 * SEGMENTS INTO PAGE 1 AND PAGE 2
 *

```

    8000      ORG      X'8000' *BEGIN BY BOOTLOADING
    8000 8000 28018010   B       BULKC *PAGE 1 WITH THE INITIAL
                            8010
    8010 8010 38008006 BULKC   ORG       BULKC *PROGRAM SEGMENT
    8011 8011 28008010   MVSG,PAGE1     PG1PROG *THIS INSTRUCTION WILL
    8012 8012 380007F0
    8013 8013 38048015
    *
    * TEST THE PAGER UNITL IT IS NOT BUSY, THEN WILL
    * COMMAND THE PAGER TO MOVE A PROGAM SEGMENT
    * INTO PAGE 1 ACCORDING TO THE SPECIFICATIONS
    * GIVEN AT PG1PROG.
    *
    8014 8014 28010200   B           BEGIN *AP CONTROL WILL WAIT
    *                           HERE UNTIL THE PAGER HAS COMPLETED LOADING
    *                           PAGE1 ACCORDING TO THE ABOVE MOVE COMMAND.
    *
    8015 8015 C0060200 PG1PROG  STRTSG,PAGE1 *DEFINE A PAGE 1 PROGRAM
    *                           SEGMENT ASSEMBLED TO EXECUTE PROPERLY ONLY
    *                           IF LOADED BEGINNING AT THE FIRST LOCATION OF
    *                           OF PAGE 1
    8016 0200 38008006 BEGIN   MVSG,PAGE2     PG2PROG *BEGIN LOADING PAGE 2
    8017 0201 28000200
    8018 0202 38000BF0
    8019 0203 38048020
    *
    801A 0204 36000000   LI           WHILE PROGRAM EXECUTION OCCURS HERE IN PAGE 1
    *                           C,0 *CLEAR COMMON REGISTER
    *
    . .
    .
    801B 0205 28010400   B           INTOPAGE2 *IF THE PAGER HASN'T
    *                           FINISHED LOADING PAGE 2 BY THE TIME THIS BRANCH
    *                           IS ENCOUNTERED, THE AP WILL WAIT UNTIL PAGING
    *                           IS COMPLETE BEFORE BRANCHING.
    ENDSG
    *
    *
    8020 8020 C0060400 PG2PROG  STRTSG,PAGE2 *DEFINE A PAGE 2 PROGRAM
    *                           SEGMENT ASSEMBLED TO EXECUTE PROPERLY ONLY
    *                           IF LOADED BEGINNING AT THE FIRST LOCATION
    *                           OF PAGE 2
    8021 0400 38008006 INTOPAGE2 MVSG,PAGE1     NEXTPAGE1 *THIS INSTRUCTION
    8022 0401 28000400
    8023 0402 380007F0
    8024 0403 3804802B
    *
    * WILL TEST THE PAGER UNTIL IT IS NOT BUSY,
    * THEN WILL COMMAND THE PAGER TO MOVE A PROGRAM
    * SEGMENT INTO PAGE 1 ACCORDING TO THE
  
```

SPECIFICATIONS AT NEXTPAGE1.

8025 0404 308000018 LRR C,(BL,DP) *COMMON = BL & DP

8026 0405 28010200 B INTOPAGE1 *AP CONTROL WILL
* WAIT HERE UNTIL THE PAGER HAS COMPLETED LOADING
* PAGE1 ACCORDING TO THE ABOVE MOVE COMMAND.
* ENDSG

8027 8027 000000020
8028 8028 000000420
8029 8029 000000820
802A 802A 000000000

802B 802B C0040200 NEXTPAGE1 STRTSG,PAGE1 *DEFINE A PAGE 1 PROGRAM
* SEGMENT ASSEMBLED TO EXECUTE PROPERLY ONLY
* IF LOADED BEGINNING AT THE FIRST LOCATION OF
* PAGE 1
802C 0200 380008006 INTOPAGE1 MVSG,PAGE2 NEXTPAGE2 *THIS INSTRUCTION
802D 0201 280000200
802E 0202 38000BF0
802F 0203 380400000

* WILL TEST THE PAGER UNTIL IT IS NOT BUSY,
* THEN WILL COMMAND THE PAGER TO MOVE A PROGRAM
* SEGMENT INTO PAGE 2 ACCORDING TO THE
* SPECIFICATIONS GIVEN AT NEXTPAGE2,ETC.
*
*
*
8030 8030 000000020 ENDSG
8031 8031 000000420
8032 8032 000000820
8033 8033 000000000

*
*
*

***** INSPECT AND CHANGE AN ARRAY FIELD *****

*
* USAGE DIRECTIONS:
*

- * 1. STORE THE ARRAY NUMBER INTO ARRAYNUM
(CODING ASSUMES 2 ARRAY MEMORY MODULES)
- * 2. STORE THE WORD NUMBER INTO WORDNUM
- * 3. STORE THE MOST SIGNIFICANT BIT POSITION
INTO MSBPOSITN
- * 4. STORE THE NUMBER OF CONSECUTIVE BIT
POSITIONS INTO NUMBITS
- * 5. TO EXECUTE AN INSPECT FUNCTION, EXECUTE A
RETURN BRANCH (USING R0) TO THE BULK CORE
ADDRESS ASSIGNED TO INSPECT. THE RESULTING
CURRENT VALUE OF THE INSPECTION MAY BE READ
FROM RESULT.
- * 6. TO EXECUTE A CHANGE FUNCTION, STORE THE
CHANGE INTO RESULT, THEN EXECUTE A RETURN
BRANCH (USING R0) TO THE BULK CORE ADDRESS
ASSIGNED TO CHANGE.

*
* START

	ORG	BULK C
8010 8010 00000000	ARRAYNUM DC	0 *ARRAY NUMBER STORAGE
8011 8011 00000000	WORDNUM DC	0 *ARRAY NUMBER STORAGE
8012 8012 00000000	MSBPOSITN DC	0 *MSB OF FIELD STORAGE
8013 8013 00000000	NUMBITS DC	0 *FIELD LENGTH OR WIDTH
8014 8014 00000000	RESULT DC	0 *THE ANSWER

*
*
*
***** STARTING PLACE FOR AN INSPECT FUNCTION
*
*
*

8015 8015 36000000	INSPECT LI	C,0 *CLEAR COMMON REGISTER
8016 8016 33918010	LR,1	FP1,ARRAYNUM *FP1 = LEAST
	*	SIGNIFICANT 8 BITS OF ARRAYNUM, SHOULD BE A
	*	VALUE OF ZERO OR ONE.
8017 8017 33418011	LR	FP2,WORDNUM *THE LINK POINTER
	*	REGISTER (FP1,FP2) NOW REFERENCES THE ARRAY
	*	AND THE WORD WITHIN THE ARRAY.
8018 8018 35A18012	LR,2	FP3,MSBPOSITN *FIELD POINTER,
	*	LOAD FP3 WITH LEAST SIGNIFICANT 8 BIT POSITIONS
	*	OF MSBPOSITN.
8019 8019 35718013	LR,3	FL1,NUMBITS *FIELD WIDTH COUNTER
	*	LOAD FL1 WITH LEAST SIGNIFICANT 8 BIT POSITIONS
	*	OF NUMBITS
801A 801A 3660C000	LOOP LI,2	AS,X'1C000' *SELECT BOTH ARRAYS
801B 801B 00007741	CLR	Y *CLEAR ALL Y
801C 801C 2821801F	BZ,FP1	1STARRAY
801D 801D 36604000	LI,2	AS,X'14000' *SELECT 2ND ARRAY
801E 801E 28018020	B	\$+2
801F 801F 36608000	1STARRAY LI,2	AS,X'18000' *SELECT 1ST ARRAY
8020 8020 03808845	L	Y,FP3 * Y = BIT COLUMN(FP3)
	*	USING THE RESOLVER LOGIC, GENERATE A 256 BIT

* INPUT WITH A ONE IN THE BIT POSITION REFERENCED
 * BY FP2 AND 255 ZEROES ELSEWHERE. LOGICAL AND THIS
 8021 8021 03C82240 GEN,32 X'03C82240! *VALUE WITH Y,I.E.,
 * RESET ALL Y'S EXCEPT IN THE SELECTED WORD
 8022 8022 28E18024 BNR \$+2 *JUMP IF SELECTED BIT = 0
 8023 8023 00008841 SET Y *SET ALL Y IF BIT = 1
 8024 8024 001F8888 GEN,32 X'001F8888! *LEFT SHIFT ARG ONE
 8025 8025 22FFA0FB GEN,32 X'22FFA0FB! *BIT POSITION
 8026 8026 21C0BFFA GEN,32 X'21C0BFFA! *LOAD BIT #31 OF
 COMMON REG. WITH SELECTED BIT NUMBER VALUE IN Y
 W 8027 8027 01310001 INCR FP1,FL1 *FP3=FP3+1, & FL1=FL1+1
 8028 8028 2891801A BNZ,FL1 LOOP *LOOP ON REST OF FIELD WIDTH
 8029 8029 30018014 SR C,RESULT *STORE THE RESULT
 802A 802A 28080000 B (R0) *RETURN TO CALLING PROGRAM

*
 *
 **** STARTING PLACE FOR A CHANGE FUNCTION
 *
 *
 802B 802B 36018014 CHANGE LR C,RESULT *ARG = NEW VALUE
 802C 802C 3660C000 LI,2 AS,X'0000! *SELECT BOTH ARRAYS
 802D 802D 35718013 LR,3 FL1,NUMBITS *OBTAIN FIELD WIDTH
 802E 802E 01010001 DECR FL1 *REPEAT COUNT FOR LOOP
 802F 802F 33900020 LT FP1,32 *PREPARE TO COMPUTE THE
 8030 8030 3C008031 RPT *MSB POSITION ON THE FIELD IN
 8031 8031 01340001 DECR FP1 *THE COMMON REGISTER
 8032 8032 35718013 LR,3 FL1,NUMBITS *RELOAD FIELD WIDTH
 8033 8033 35A18012 LR,2 FP3,MSBPOSITN *(LOAD ARRAY FIELD
 POINTER. FP1 IS THE COMMON REG. FIELD POINTER.
 W 8034 8034 31818043 SR FP1,SAVEFP1 *TEMP SAVE POINTER
 8035 8035 33418011 LR FP2,WORDNUM *INIT LINK POINTER
 8036 8036 33818043 REPEAT LR FP1,SAVEFP1 *GET COM REG FIELD
 *
 8037 8037 0030B7A0
 *
 W 8038 8038 31818043
 8039 8039 33918010 LR,1 FP1,SAVEFP1 *TEMP SAVE PTR
 803A 803A 00007741 CLR FP1,ARRAYNUM *INIT LINK PTR
 803B 803B 03C88840 GEN,32 Y *CLEAR ALL Y
 *
 *
 *
 *
 803C 803C 08000002 L X'03C88840! *USING THE RESOLVER
 803D 803D 00008843 L LOGIC, GENERATE A 256 BIT INPUT WITH A ONE IN THE
 803E 803E 0BA00001 GEN,32 BIT POSITION REFERENCED BY FP2 AND 255 ZEROES
 803F 803F 13A00002 ELSEWHERE. LOAD THIS VALUE INTO Y,I.E., THE Y RS
 CONTAINS A ONE FOR ONLY THE REFERENCED WORD.
 *
 *
 *
 *
 8040 8040 01010001 L M,Y *SET UP FOR A MASKED WRITE
 8041 8041 28918036 L Y,X *Y = COMMON REG(FP1) VALUE
 8042 8042 28080000 SM Y,FP3+ *STORE THE COMMON REG
 8043 8043 00000000 SAVEFP1
 0000 END
 BIT VALUE REFERENCED BY FP1 INTO THE WORD OF
 MEMORY SPECIFIED BY FP2 AND THE BIT COLUMN IN
 THE SELECTED MEMORY WORD REFERENCED BY FP3,
 THEN INCREMENT FP3.
 DECR FL1 *DECREMENT FIELD WIDTH
 BNZ,FL1 REPEAT *STORE OTHER BITS
 B (R0) *RETURN TO MAIN PROGRAM
 DC
 END

INDEX

A

Addressing	2-7
APPLE	1-2
APPLE features	1-2
APPLE language structure	2-1
Argument field	2-1
Arithmetic	2-138
Array co-ordinates	3-10
Assembler directives	1-2, 2-9
Assigning slots	3-1, -5
Associative instructions	2-55
Associative memory device code	3-8
Associative Memory or Common Register field expression	2-7
Attach (SVC)	3-5

B

Branch instructions	2-17
Buffer format	3-7, -9, -13, -19
Buffer format for device -1 (Control memory)	3-7
Buffer format for device -2 (Associative memory)	3-9
Buffer format for device -3 (Registers)	3-13
BUFFER pseudo-op	3-3, -6, -8, -12, -17
Byte count	3-9, -18, -19,
Byte count update	3-20

INDEX

C

Character set	2-4
Command field	2-1
Command summary	A-i
Comment field	2-3
Comment statements	1-2
Common Register field	2-7
Constants	2-5
Control and test	2-154
Control Digit	2-7, C-2
Control Memory address	2-7, C-2
Control Memory device code	3-6

D

Decimal constants	2-5
Device Assignment Table (DAT)	3-2
Device codes	3-5
Devices	3-5
Done bit	3-10, -19

E

Echo Bit	3-20
End-of-medium bit (EOM)	3-9, -19
Error codes	B-i

INDEX

E

Error codes (SVC)	3-10, -19
Execution location counter	2-6, C-2
Exit (SVC)	3-24
Expressions	2-6

F

Field expression	2-7
Formatted ASCII	3-18
Formatted binary	3-18
Free device (SVC) for new task	3-23

H

Hexadecimal constants	2-5
Hexadecimal/decimal table	D-i

I

I setup	3-27
Immediate value	2-5, 2-39, C-4
Improper mode	3-9, -19
In line	1-2
Interrupt setup (SVC)	3-26, -27
Interrupt signal (SVC)	3-26, -27

INDEX

L

Label field	2-1
Language elements	2-4
Load Location counter	2-6, C-5
Loads	2-55
Location counter symbol (\$)	2-6
Location counters	2-6

M

Max size	3-9, -17
Mnemonic summary	A-i
Mode byte	3-20
Moves	2-125

O

Octal constants	2-5
Octal/decimal	E-i
One-to-many	1-2
One-to-one	1-2

P

Pager control (SVC)	3-28
Pager instructions	2-159
PI/O Control (SVC)	3-29
Powers of two table	F-i
Program Counter (location counter)	2-6, C-5
Program examples	G-i

INDEX

R

Read (SVC)	3-15
READ BUFFER Pseudo-op	3-17
Register device code	3-13
Register instructions	2-35
Required entries	2-3
Reset peripheral device (SVC)	3-22
Restart program (SVC)	3-21

S

Searches	2-108
Slot numbers	3-1
Source statements	2-1
STARAN S registers (device code -3)	3-13
Status byte	3-7, -9, -19
Stores	2-79
Subroutine call sequence	1-2
Summary of APPLE mnemonics and instruction formats	A-i
Supervisor calls (SVC)	3-1
Supervisor services	3-4
Symbol table	2-4
Symbols	2-4

T

Terms and symbols	C-i
-------------------	-----

INDEX

T

Timer (SVC) 3-25

U

Unformatted ASCII 3-18

Unformatted binary 3-18

W

Write (SVC) 3-16

WRITE BUFFER Pseudo-op 3-17

GOODYEAR AEROSPACE

