VAX-11/780
Microprogramming Tools
User's Guide

Order No. AA-H306B-TE

March 1982

Digital Equipment Corporation • Maynard, Massachusetts

First Printing, June 1979
Second Printing, March 1982

The information in this document is subject to change without notice and should not be construed as a commitment by Digital Equipment Corporation. Digital Equipment Corporation assumes no responsibility for any errors that may appear in this document.

The software described in this document is furnished under a license and may by used or copied only in accordance with the terms of such license.

No responsibility is assumed for the use or reliability of software on equipment that is not supplied by Digital Equipment Corporation or its affiliated companies.

Printed in U.S.A.

The postpaid READER'S COMMENTS form on the last page of this document requests the user's critical evaluation to assist in preparing future documentation.

The following are trademarks of Digital Equipment Corporation:

DEC	DECsystem-1Ø	PDT
DECUS	DECSYSTEM-2Ø	RSTS
DIGITAL	DECwriter	RSX
PDP	DIBOL	VMS
UNIBUS	EduSystem	VT
VAX	IAS	DIGITAL Logo
DECnet	MASSBUS	ZKDSR

TABLE OF CONTENTS

PREFACE

Manual Objectives

This manual describes the use of the tools available for the VAX 11/780 WCS user. The appendices contain the VAX 11/780 Definition Language, and a sample program expressed in this language.

Intended Audience

This manual is intended for assembly language programmers and hardware engineers. The reader is assumed to be familiar with microprogramming and the characteristics of the VAX CPU architecture.

Structure of this Document

This manual describes the process of assembling, loading, and executing a microprogram.

Associated Documents

Information on the MICRO2 assembler is given in the following document:

MICRO2 User's Guide/Reference Manual (AA-H531A-TE)

Information on the VAX 11/780 Data Path is given in the following document:

VAX 11/780 Data Path Description (AA-H307A-TE)

Information on the VAX 11/780 software is given in the following document:

VAX 11/780 Software Handbook

The VAX/VMS command language and environment are described in the following document:

VAX/VMS Command Language User's Guide (AA-D023A-TE)

The text editing capability is described in:

VAX-11 Text Editing Reference Manual (AA-D029A-TE)

Information on the VAX 11/780 Hardware is given in the following documents:

VAX 11/780 Hardware Handbook
VAX 11/780 Architecture Handbook

Conventions Used in this Document The conventions used in this document are the same as those used in the VAX/VMS Command Language User's Guide (AA-D023A-TE).

CHAPTER 1

INTRODUCTION

The VAX-11/780 Extended Writable Control Store consists of 2048 words occupying addresses 1800 through 1FFF, 1024 words occupying addresses 1C00 through 1FFF when G&H is present. Each word contains 96 bits plus three parity bits. User microprograms that enhance a machine for specific applications execute in the Extended WCS. The process of writing, loading, and executing microprograms is diagrammed below.

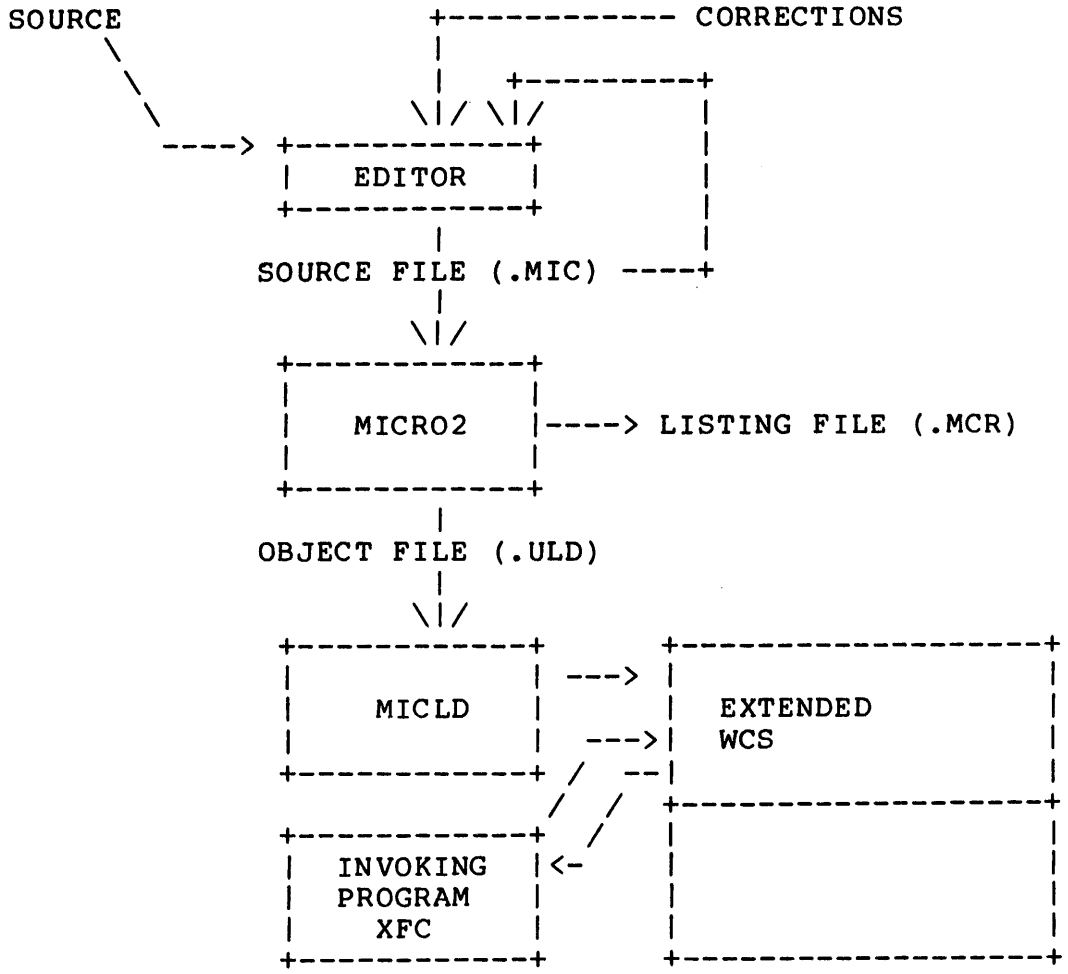


Figure 1-1

The first step in microprogramming the WCS is the creation of the microprogram. To do this, you write the microprogram in MICRO2 source language and create a source file (.MIC). Then, you assemble the microprogram to create a listing file (.MCR) and an object file (.ULD). MICRO2 detects as many errors as possible in the source microprogram. You correct the errors and reassemble until you are satisfied with the resulting microprogram.

When you are ready to test the microprogram, you load it into the WCS using the microprogram loader, MICLD. You can then transfer to the microprogram by executing an XFC instruction in a main memory program.

This manual describes assembling, loading, and executing a microprogram for the VAX 11/780 WCS. Some information on debugging a microprogram can be found in Appendixes C and D of the VAX 11/780 Data Path Description (AA-H307A-TE).

A macro language for microprogramming the VAX 11/780 is given in Appendix A. This language, called the VAX 11/780 Predefinition language, defines the fields of the microword and a set of macros for performing the logical functions associated with microprogramming the VAX 11/780. A VAX 11/780 microprogram written in this definition language is given in Appendix B.

CHAPTER 2

ASSEMBLING YOUR MICROPROGRAM

MICRO2 is a general purpose tool, written in BLISS, that executes under user control in the VAX/VMS environment. The MICRO2 language lets you express the actions of a microprogram symbolically.

The MICRO2 assembler translates a microprogram written in its source language to the bit representation that is loaded into the Extended WCS. In doing this, it performs syntactic checks on the program and issues messages if the source microprogram is not valid.

Further, MICRO2 allocates any microwords that you do not specifically allocate. You can allocate a microword absolutely, specify a constraint on its allocation (such as the two lowest order bits of the address must be zero), or leave the allocation to MICRO2.

The following sections provide a quick reference to the MICRO2 language and its use. A complete description of the MICRO2 assembler is given in a separate document:

MICRO2 User's Guide/Reference Manual (AA-H531A-TE)

The MICRO2 User's Guide/Reference Manual is a tutorial, which contains many examples of the use of the MICRO2 language.

2.1 PROGRAM STRUCTURE

The MICRO2 assembler is a line-oriented processor, which accepts a sequence of input lines written in MICRO2 source language and produces a listing file and an object module.

The input to MICRO2 is a source program. A MICRO2 source program can contain one or more memories. The bit-numbering direction and the program radix apply to the entire program.

2.1.1 The Bit Numbering

The .LTOR and .RTOL keywords define the way in which the bits of a microword are numbered, so that MICRO2 knows whether to count from the left end or the right end of the word to locate a bit position. The form of the bit-numbering keyword-line is the keyword itself, namely:

```
.LTOR
.RTOL
```

The .LTOR keyword directs MICRO2 to consider the bits of the microword numbered from left to right. The .RTOL keyword directs MICRO2 to consider the bits numbered from right to left.

If no bit-numbering keyword is given, then .LTOR is assumed.

MICRO2 uses the first .LTOR or .RTOL keyword it finds to establish the direction in which the bits are numbered and ignores any subsequent bit-numbering keywords in the program.

2.1.2 The Program Radix

The program radix is set by either the .OCTAL or .HEXADECIMAL keywords. The form of the .OCTAL and .HEXADECIMAL keyword-line is simply the keyword itself, as follows:

```
.OCTAL
.HEXADECIMAL
```

MICRO2 uses the first program radix keyword it find and ignores any subsequent program radix keywords in the program.

2.1.3 Memories

A program can include data for as many as 26 memories. Except for the bit-numbering convention and program radix, which can be specified only once in a program, all other constructs are considered to belong to a memory. Each memory has its own address-space, word-width, field- and macro-definitions, and microinstructions.

The beginning of a section of the memory is specified by a memory-indicator keyword. The memory-indicator keywords are as follows:

```
.xCODE
```

where x is any one of the letters in the alphabet

```
ex.     .UCODE
        .ACODE
```

The identification, definitions, and instructions following that memory-indicator and up to the end of the program or another memory-indicator are associated with the specified memory.

2.1.4 The Program Title

The .TITLE keyword-line supplies a title for MICRO2 to use as part of the heading of the output listing. MICRO2 reproduces the quoted string following the .TITLE keyword as part of the first line of the output listing. The .TITLE keyword-line has the following form:

```
.TITLE "title-string"
```

2.1.5 The Table of Contents

The .TOC keyword-line supplies a subtitle and adds an entry to the table of contents. MICRO2 reproduces the text given in quotes following the .TOC keyword as part of the second line of the page heading of the output listing. The .TOC keyword-line has the following form:

```
.TOC "text-string"
```

2.1.6 Listing Pagination

The .PAGE keyword-line indicates a new listing page and, optionally, provides a table of contents entry and a subtitle.

To simply indicate a new page for the output listing, include the .PAGE keyword without any text string, as follows:

```
.PAGE
```

To start a new page, add a subtitle, and make an entry in the table of contents, add a text string to the .PAGE keyword-line, as follows:

```
.PAGE "text-string"
```

2.1.7 Comments

Comments can be included anywhere in the program. A comment begins with a ";" character and ends at the end of the line.

2.2 FIELD DEFINITIONS

A field-definition consists of a name followed by the separator '/=' followed by the position of the bits within the word to be associated with the field name followed by a list of one or more qualifiers. The form is:

```
field-name /= < left-bit:right-bit > { , qualifier ... }
```

A single bit field can be expressed by including only the left-bit within the angle brackets. Qualifiers can be omitted.

2.2.1 Names

A field-name can be any valid MICRO2 name. MICRO2 allows a name to be made up of characters from the following set:

A B C ...Z	Upper case letters
a b c ... z	Lower case letters
0 1 2 ... 9	Numbers
!	Exclamation mark
#	Hash mark
&	Ampersand
(	Left parenthesis
)	Right parenthesis
<	Left angle bracket
>	Right angle bracket
*	Asterisk
+	plus sign
-	minus sign
.	period
?	question mark
_	Underscore
	Space and tab

2.2.2 Field Position

The left-bit and right-bit are decimal numbers that identify the beginning and end bits of the field in the microword. If you specified right-to-left bit numbering, then the left-bit must be greater than or equal to the right-bit. If you specified left-to-right bit numbering, then the left-bit must be less than or equal to the right-bit.

2.2.3 Qualifiers

Qualifiers are used to establish a default for a field, to identify the field as one that can contain a label, to designate a field to be used for a parity bit, and to associate the setting of a field with the condition of other fields within the microword.

2.2.3.1 The .DEFAULT Qualifier - The .DEFAULT qualifier specifies a value that MICRO2 can use for a field when the field is not explicity set. The form is:

```
.DEFAULT = expression
```

In forming a microword, MICRO2 begins with a word consisting of all zeroes, sets the fields explicitly set in the microinstruction, and then applies defaults. MICRO2 uses a default for a field if and only if no bit of the field is set explicitly.

In applying defaults, MICRO2 uses the order in which the fields are specified in the microprogram.

2.2.3.2 The .ADDRESS and .NEXTADDRESS Qualifiers - MICRO2 requires that the jump field be identified by either an .ADDRESS or .NEXTADDRESS qualifier. A field defined with the .ADDRESS or .NEXTADDRESS qualifier can be set to the value of any label in the program. The form is simply the keyword, namely:

```
.NEXTADDRESS
.ADDRESS
```

In addition to designating the associated field as a jump field, the .NEXTADDRESS qualifier specifies that the default for the field is the value of the address associated with the next microinstruction given in the program.

2.2.3.3 The .VALIDITY Qualifier - The .VALIDITY qualifier lets you make assertions about the conditions under which a field can be legally set. The form is:

```
.VALIDITY = expression
```

The .VALIDITY qualifier associates a validity expression with a field. If the validity expression is not satisfied when the field is set in a microword, MICRO2 produces a warning message.

2.2.4 Value Definitions

A value-definition associates a name with a particular value of a particular field.

Value-definitions follow a field definition. A value-definition conists of a value-name followed by the separator '=' followed by the value to be equated with that name. The value-definition can also have its own .VALIDITY expression. Thus, the form is:

```
value-name=value,.VALIDITY=exp
```

A value-name is any valid MICRO2 name, as defined in Section 2.2.1. The .VALIDITY expression is optional.

2.3 EXPRESSIONS

An expression in MICRO2 is enclosed in angle brackets. An expression can be any of the following:

A number
An expression name
A function call
A field value name
A field contents indicator
A predefined symbol

The following sections consider each of these cases in detail.

2.3.1 Numbers

MICRO2 recognizes integers or decimal numbers. An integer is interpreted according to the program radix. A number with a decimal point is always interpreted as a decimal number. The program radix is set by either the .OCTAL or the .HEXADECIMAL keyword. If a program radix is not given, then an octal radix is assumed.

2.3.2 Expression-Names

An expression-name is defined by the .SET keyword as follows:

```
.SET/expression-name = <expression>
```

2.3.3 Function Calls

MICRO2 provides functions for comparison, arithmetic, and Boolean operations. Also, MICRO2 provides functions to detect parity, shift, and select a case from a set of choices.

The functions are given in the following table:

Function	Value
Comparison	
.EQL[op1,op2,...]	1 if op1=op2=...
.NEQ[op1,op2,...]	1 if op1<>op2 and op2<>op3 and ...
.GTR[op1,op2,...]	1 if op1>op2>...
.GEQ[op1,op2,...]	1 if op1>=op2>=...
.LSS[op1,op2,...]	1 if op1<op2<...
.LEQ[op1,op2,...]	1 if op1<=op2<=...
Arithmetic	
.MAX[op1,op2,...]	Value of largest operand
.MIN[op1,op2,...]	Value of smallest operand
.SUM[op1,op2,...]	op1+op2+...
.PROD[op1,op2,...]	op1*op2*...
.DIFF[op1,op2]	op1-op2
.QUOT[op1,op2]	op1/op2 (truncated)
.MOD[op1,op2]	remainder of op1/op2
Boolean	
.NOT[op]	Boolean complement of op
.AND[op1,...]	Boolean 'and' of operands
.OR[op1,...]	Boolean 'or' of operands
.XOR[op1,...]	Boolean 'xor' or operands
.NAND[op1,...]	Boolean complement of the 'and'
.NOR[op1,...]	Boolean complement of the 'or'
.EQV[op,...]	Boolean complement of the 'xor'
Miscellaneous	
.PARITY[op1,op2,...]	If operands contain an even number of 1's, then 1 else 0
.SHIFT[op1,op2]	If op2 is positive, then shift op1 left op2 places else shift op1 right op2 places
.CASE[op1]OF[op2,...]	The (op1-th + 1) operand of the list. That is, if op1 is 0, the first op2 is used. Up to 32 choices can be given.
.SELECT[{op1,op2,}...]	The first op2 for which op1 is true

The operands of a function can be expressions.

2.3.4 Value-Names

Since a value-name is only defined for a specific field, it must be qualified by the field-name when used in an expression as follows:

```
field-name/value-name
```

2.3.5 Field Contents Indicators

The contents of a field can be designated in an expression by giving the field-name followed by a slash. For example, to find out if the current contents of field B contains the value 4, you write the following expression:

```
.EQL[<B/>,<4>]
```

2.3.6 Predefined Symbol Names

The following symbols are predefined in MICRO2, as follows:

Symbol	Meaning
. (period)	The address of the current microinstruction

2.4 MACROS

The macro capability of MICRO2 permits the definition of a representation for a microprogram at a higher level than the basic field-value pairs. Once the fields of your microword are defined, a set of macros that set groups of fields appropriately for certain operations can be specified. Macros cannot generate more than one microinstruction.

2.4.1 The Macro-Name

Macro-names are formed using the set of characters given in Section 2.2.1. In addition to these characters, MICRO2 recognizes square bracket pairs and commas in macro-names as indicators of the number and position of the macro parameters.

The number and position of parameters in a macro-name are an integral part of the name. (That is, the macro ABC[][] is not the same as ABC[,].)

2.5 THE MACRO-BODY

The macro-body consists of any combination of field-settings and macro-calls separated by commas. When a macro is used in a microinstruction, MICRO2 replaces the macro-name by the macro-body associated with that name.

2.5.1 Parameters

Square brackets and commas indicate parameters in the macro-name. The character "@" followed by a decimal integer in the macro-body indicates the position of the parameter in the macro-body. This character pair is called a parameter-designator.

The decimal integer in the parameter-designator refers to the position, numbering from left to right, of the parameter in the name.

2.6 MICROINSTRUCTIONS

The microinstructions describe the processing to be performed by the microprogram. These microinstructions are expressed in terms of the field- and macro-names defined.

For each microinstruction, MICRO2 translates names into the appropriate sequence of bits and creates the associated microword. The microinstruction contains the information MICRO2 needs to set the bits of the microword.

A microinstruction begins with an absolute address assignment, one or more labels, or both. Following this optional information, a sequence of field-settings and/or macro-calls is given separated by commas.

That is, the form of the microinstruction is:

```
address:
{ label: }
    ...
          { field-setting } ,...
          { macro-call    }
```

Both the address and label can be omitted.

2.6.1 Continuing A Microinstruction

If a microinstruction occupies more than one line, the separator character ',' must be as the last non-blank character of all lines except the last line. For purposes of this discussion, the end of the line is assumed to be either the ';' character, which begins a comment, or the actual end of line. Thus the last non-blank character of a line means the last non-blank before the ';' or end of line.

2.7 THE MICROWORD

MICRO2 creates a microword in the following way:

1. MICRO2 begins with a word of the specified length in which each bit has a value of zero and a status of unset.

2. MICRO2 then fills in all the fields that are explicitly set in the microinstruction.

3. Then, MICRO2 sets any fields that have an associated default and that contain only unset bits.

4. Then, MICRO2 evaluates any VALIDITY expressions.

5. Finally, MICRO2 performs any parity adjustment indicated.

2.8 THE ADDRESS SPACE

The .REGION keyword determines the address-space. The .REGION keyword is followed by one or more pairs of address limits, as follows:

```
.REGION/low-bound,high-bound...
```

Low-bound and high-bound are expressions whose values are interpreted according to the program radix. An address-space thus can consist of any number of address-ranges. An address-range is specified by the low-bound and high-bound.

Any number of .REGION keyword-lines can be given. MICRO2 allocates the microinstructions following a .REGION up to the next .REGION keyword (or the end of memory) in the specified address space.

If a .REGION keyword-line is not given at the beginning of a memory, MICRO2 assumes that the address space begins at 0 and ends at MAXPC, the highest available address for the given architecture.

2.9 SPECIFYING THE METHOD OF ALLOCATION

Within the specified address space, either sequential or random allocation (or some combination of both) can be used.

2.9.1 Sequential Allocation

In sequential mode, MICRO2 allocates a microinstruction by taking the address of the previous microinstruction and adding 1.

MICRO2 begins allocating with the first address in the address space defined by the .REGION keyword and continues incrementing until it reaches either an absolute address assignment or the end of an address-range.

When it reaches an absolute address, MICRO2 uses that address for the associated microinstruction and as the new base for incrementation.

When MICRO2 uses the last instruction in an address-range, it chooses the first address in the next address-range for the next microinstruction. After MICRO2 uses the last address in the last range, it uses the address 0000 and issues an error message for each word allocated following the last legal allocation.

2.9.2 Random Allocation

In random mode, constraints can be given to select a set of addresses based on the low order bit configuration. Constraints are described in detail in the next section.

MICRO2 first allocates all absolute assignments and constraints and then allocates the remaining microinstructions starting at the first unallocated address in the first address-range and continuing sequentially through the unallocated addresses of the address space.

2.9.2.1 Constraints - Many microprogrammable microprocessors perform conditional branching by ORing some logic function into the low order bit position of the next microinstruction address. MICRO2 provides a constraint capability for generating a set of addresses for conditional branching.

A constraint consists of an "=" character followed by a constraint string composed of a sequence of 0 and 1 characters.

A constraint specifies a set of addresses. In response to a constraint string, MICRO2 chooses a base address that satisfies the low order bit configuration specified by the constraint. The bits of an address are always ordered from right to left. So the low order bit is the right-most bit.

MICRO2 then assigns the next n microinstructions to the addresses formed by systematically increasing the base address counting only in those bits designated as 0's in the constraint string.

2.9.2.2 Indicating a bit that can be 0 or 1 - In addition to 0's and 1's, the character '*' can be used in a constraint string. This character informs MICRO2 that it can select an address that has either a 0 or a 1 in that position for the base address.

2.9.2.3 The size of the address set - The number of microinstructions in the set, n, is determined by the number of zeroes in the constraint string, as follows:

```
n=2**X
```

Where X is the number of 0's in the constraint string.

2.9.2.4 Constraints Within Constraints - If MICRO2 encounters a constraint string within the set of instructions it is allocating to the block of addresses associated with an outer constraint string, it skips to the next address satisfying the inner constraint and then proceeds according to the algorithm specified by the outer constraint. The purpose of the nested constraint is to skip over some addresses that would otherwise be allocated by the outermost constraint.

2.9.2.5 Terminating a Constraint - A null constraint within the scope of the constraint terminates the constraint. A null constraint is the "=" character. A constraint can also be terminated by an absolute address assignment; however, in this case, MICRO2 issues a warning message.

2.10 COMMUNICATION

In MICRO2, communication among memories in the same program and communication among separate programs can be accomplished.

2.10.1 Memory Communication

Each memory has its own definitions, identification, and address-space. However, if the same field-name is defined in more than one memory, then the value-names defined for that field in any memory are known in all other memories that define the field. This feature permits communication between memories.

2.10.2 Program Communication

Separate programs can be assembled and loaded in a control store by handling address space assignment and communication. If, for example, you wish to have n separate programs, you divide the control store into n+1 logical spaces, namely:

- Communication Space
- Space for Program 1
- Space for Program 2
  ...
- Space for program n

If the address of entry points are fixed, these separate programs can transfer to one another.

2.11 CONDITIONAL ASSEMBLY

The conditional assembly capability permits suppression of the assembly of parts of a program.

2.11.1 The Conditional Assembly Keywords

Three keywords are provided for conditional assembly as follows:

```
.IF/expression-name
.IFNOT/expression-name
.ENDIF
```

These keywords divide a program into blocks. The .IF and .IFNOT keywords begin a block. They include an expression-name that is associated with either a true (1) or false (0) value. These keywords have the following meaning.

Keyword	Meaning
.IF/expression-name	If expression-name is associated with a true value (1), assemble the following block; otherwise suppress its assembly.
.IFNOT/expression-name	If expression-name is associated with a false value (0), assemble the following block; otherwise, suppress its assembly.

In practice, a false value is any value that is not 1. For example, if the expression-name has the value 2, MICRO2 considers it to represent a false value.

2.11.2 Conditional Assembly Blocks

A conditional assembly block begins with either an .IF or .IFNOT and ends with either an .ENDIF for the same expression or another .IF or .IFNOT for the same expression.

2.12 SETTING AND CHANGING EXPRESSION-NAMES

Expression-names are defined and set with the .SET keyword as follows:

```
.SET/expression-name=expression
```

Once an expression-name is defined, .CHANGE keyword must be used to change its value.

```
.CHANGE/expression-name = expression
```

2.13 LIST CONTROLS

The list controls specify which portions of the output listing are to be produced.

MICRO2 determines whether or not to make a contribution to a file by looking at a counter. If the counter contains a positive number, MICRO2 contributes to the associated file. If the counter is a negative number, MICRO2 does not contribute.

The list controls are as follows:

Keyword	Meaning
.LIST	Increment the listing counter
.NOLIST	Decrement the listing counter
.CREF	Increment the cross reference counter
.NOCREF	Decrement the cross reference counter
.BIN	Increment the object counter
.NOBIN	Decrement the object counter
.EXPAND	List all fields explicitly inserted in an instruction after the last line of the instruction.
.NOEXPAND	Do not list fields.

At the beginning of an assembly, each counter has the value 0.

2.13.1 The List Control Counters

If a list control counter is positive, then MICRO2 creates the specified part of listing. If a list control is negative, MICRO2 suppresses the specified part of the listing.

The counter associated with the .LIST and .NOLIST control determines whether or not an output listing is produced. The counter associated with the .BIN and .NOBIN controls determines whether or not the object part (left field) of the listing is produced. The counter associated with the .CREF and .NOCREF controls whether or not names will be added to the cross reference map. The use of the MICRO2 assembler in the VAX environment is described in the following sections.

2.14 THE VAX 11/780 DEFINITION LANGUAGE

The VAX 11/780 Definition Language describes the VAX 11/780 architecture and provides a macro language for writing microprograms. A listing of this definition is given in Appendix A; the source for the definition language is available on a floppy in the VAX 11/780 WCS kit. This file (VAXDEF.MIC) is copied to SYS$LIBRARY when the user WCS tools are installed on a system.

You can express the actions of a microprogram in the macros given in the definition language. Then, when you assemble your program, you include the file VAXDEF.MIC as the first input file, as indicated in Appendix A.

2.15 USER INTERFACE

The MICRO2 assembler is called at command level as shown below:

Format

```
+-------------------------------------------------+
|                                                 |
|     MICRO2 input-file-spec                      |
|                                                 |
|     File Qualifiers                             |
|     ---------------                             |
|          /LIST[=file-spec]                      |
|          /NOLIST                                |
|          /ULD = [file-spec]                     |
|          /NOULD                                 |
+-------------------------------------------------+
```

Prompts

_File: input-file-spec

File-Parameters

Input-file-spec
Specifies the names of one or more files to be assembled. If you specify more than one input file, you can use the character '+' to separate file-specs. Input files must not have line numbers; such files are rejected by MICRO2.

Description
MICRO2 assembles the programs contained in the input-file-spec and produces a listing file and an object file.

File Qualifiers

/LIST [=file-spec]
Directs MICRO2 to produce a listing file. If you include a file-spec, MICRO2 uses that file-spec for the listing file. If you do not include a file-spec, MICRO2 uses the name of the input-file, or the name of the first input file in the case of multiple input files, with the default extension .MCR for the listing file.

/NOLIST
Directs MICRO2 to suppress the listing file.

/ULD [=file-spec]
Directs MICRO2 to produce an object-file. If you include a file-spec, MICRO2 uses the file-spec for the output file. If you do not include a file-spec, MICRO2 uses the name of the input, or the name of the first input file in the multiple input file case, with the default extension .ULD for the object file.

/NOULD
Directs MICRO2 to suppress the object file.

Examples

1. MICRO2 ALPHA

   MICRO2 assembles the program in the file ALPHA.MIC and produces a listing file ALPHA.MCR and the object file ALPHA.ULD.

2. MICRO2/LIST=BETA ALPHA

   MICRO2 assembles the program in the file ALPHA.MIC and produces the listing file BETA.MCR and the object file ALPHA.ULD.

3. MICRO2/NOULD ALPHA+GAMMA

   MICRO2 assembles the program formed by the concatenation of ALPHA.MIC and GAMMA.MIC and produces the listing file ALPHA.MCR.

4. MICRO2/LIST=BETA/NOULD ALPHA

   MICRO2 assembles the program in the file ALPHA.MIC and produces the listing file BETA.MCR. MICRO2 does not produce an object file because the qualifier /NOULD is given.

5. MICRO2 [SYSLIB]VAXDEF+MYPROG

   MICRO2 assembles the definition language in the file SYS$LIBRARY:VAXDEF.MIC and the microprogram in the file MYPROG.MIC and produces the listing file VAXDEF.MCR and the output file VAXDEF.ULD.

File Specifications

MICRO2 accepts any legal VAX filename. For the purpose of error reporting, MICRO2 abbreviates the filename to the first six characters and the extension to the first three characters.

CHAPTER 3

LOADING A MICROPROGRAM

MICLD is a BLISS program that runs under user control in the VAX/VMS environment. MICLD loads the object files (.ULD) produced by MICRO2 into the Extended Writable Control Store.

MICLD requires kernel (CMKRNL) and error logging (BUGCHK) privileges. It uses VAX privileged registers and instructions to load the microprogram into the WCS and to report the WCS load in the system error log. To use MICLD, therefore, you must have privileges to execute in kernel mode.

3.1 FUNCTIONS

In loading a microprogram, MICLD does the following:

- Verifies that the Extended WCS board is installed.
- Initializes each word of the Extended WCS to a special pattern.
- Loads the set of ULD formatted object modules that make up the microprogram into the Extended WCS.
- Optionally sets the entry vector (VAX address 10E0 hex) to jump to the place in the microprogram where execution begins.
- Records the fact that the WCS was loaded into the system error log.

The following sections consider each of these functions in detail.

3.1.1 Verifying the Installation of the Extended WCS Board

MICLD first checks that the Extended WCS Board is physically and operationally present in the system. If MICLD finds that the Extended WCS Board is not installed, it issues an error message and exits back to the operating system.

3.1.2 Initializing the Extending WCS

If MICLD finds that the Extended WCS is properly installed, it then initializes each word of the Extended WCS, starting at 18ØØ and continuing through 1FFF (1CØØ through 1FFF if G&H is present) to an initialization pattern.

You can specify the initialization pattern for MICLD to use by including the file qualifier /INITIAL, as described in Section 3.3. If you do not specify an initialization pattern, MICLD uses the default pattern given in Figure 3-1.

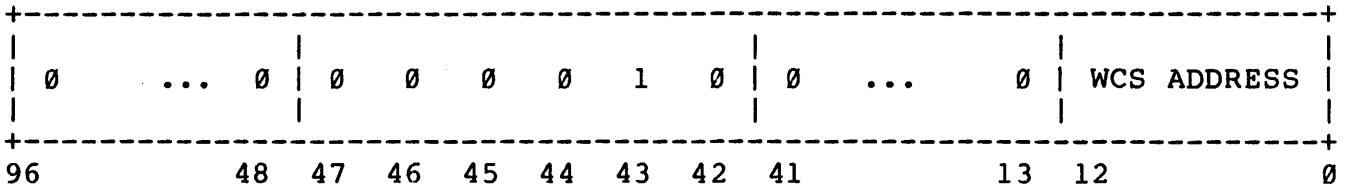


Figure 3-1. Default Initialization Pattern

Bits Ø through 12 of the default initialization pattern contain the address of the microword in the WCS being initialized. Bits <47:42> contain a two (2) to guarantee that no reads or writes will occur to the main memory should the microword inadvertently be executed. All other bits in the 96-bit wide microword are set to Ø.

After execution of MICLD, any microaddress not explicitly loaded contains the initialization pattern. If a microprogram mistakenly jumps to a word that contains the default initialization pattern, then the execution of that word causes a branch to itself with no-op function. The machine then loops at that instruction, continuing to branch to itself. Thus, if your program branches to an unexpected address, you get both protection and information. To recover control from this microcode loop, you enter the console "INIT" command while the console is in console command mode. If the console is not in console command mode, entering Control-P at the console will get it there.

If a microprogram gets into an instruction loop as described above, you must manually reboot the system and then reload the WCS.

3.1.3 Setting the Starting Address

The entry vector (VAX address 1ØEØ hex) must be set to the address at which the microprogram in the Extended WCS begins. MICLD sets the entry vector to an address if an /ENTRY qualifier is given in the command line. If an address is given with the /ENTRY qualifier, MICLD interprets that address as a hexadecimal number and uses that number to set the entry vector. If an address is not given, then MICLD uses the address of the first microinstruction in the ULD file to set the entry vector.

If an /ENTRY file qualifier is not given, the entry vector must be set by using the console program or the privileged instructions before the execution of the microprogram is attempted. The entry vector is discussed further in section 4.1.2.

3.1.4 Loading the Microprogram

MICLD loads the microwords specified in the ULD file. MICLD begins by creating an image of the loaded WCS. It initializes this image to the initialization pattern and then reads the ULD files, starting with the first word in the first file and continuing sequentially until the last word of the last file.

Each entry in the ULD file for a microword contains both the address of the microword and its contents. MICLD uses the address to determine the position within the image in which the contents of the word is to be stored.

When MICLD finishes reading the last word of the last file, it loads the created image sequentially into the control store.

MICLD permits over-writing. That is, it lets you load an address more than once. When MICLD loads a word into any address, it checks to see if the address has been loaded previously. If so, MICLD issues a warning message and then loads the new word into the given address, destroying the previous contents.

If you take advantage of the over-writing capability of MICLD, you must be careful about the order in which you specify files when you call MICLD. If you do not over-write the WCS, then you can specify your files in any order.

MICLD issues a message at the end of the loading process, indicating whether or not the loading was successful.

3.1.5 Logging the WCS Load

Each time MICLD runs successfully, it makes a note in the system error log that the WCS contents has been changed. These notes can help the system manager determine which user microcode is in the WCS, or relate system problems to particular pieces of user microcode.

3.2 VERIFYING THAT THE LOADING WAS SUCCESSFUL

It sometimes happens that, due to a hardware malfunction, the WCS is not properly loaded and MICLD is not able to detect that fact. The WCS is a write-only memory and MICLD, thus, cannot verify its contents by reading it back after loading and comparing its actual contents with its expected contents.

Erratic or unexpected performance of the executing microprogram can indicate that the WCS is not properly loaded. However, such behavior can also mean that the microprogram is not completely debugged. Under these circumstances, you can try to validate the loading process by one of the following methods.

3.2.1 Using The Sample Program

One way to try to validate the loading process is to load and execute a program whose behavior is known. The sample microprogram given in Appendix B can be used for this purpose. A command file that uses a test program to verify the installation of the tools and the loading process is included in the VAX 11/780 WCS kit.

To invoke this command file in the VMS environment, type:

```
@[SYSEXE]WCSTOLTST
```

This command file assembles the sample microprogram (BSERCH) given in Appendix B utilizing the VAX Definition Language given in Appendix A. It then loads the resulting object file into the extended WCS and executes the sample assembly language test program (BSTEST) given in Section B.4. BSTEST executes an XFC causing the loaded sample program to be executed.

After execution of the microprogram, control returns to BSTEST. If the microprogram executed properly, BSTEST prints the following message on the terminal:

```
"Successful Test Completion"
```

3.2.2 Sequencing With The Debugger

Another way to validate the loading process is to invoke the WCS debugger from the VAX console and single step the microprogram. You can then observe if the correct sequence of address is executed. The debugger is described in Appendix D of the VAX 11/780 Data Path Description.

Since the WCS is a write-only memory, the debugger is not able to read its contents. You must create a floppy disk image of the contents of the WCS. The debugger then gives the illusion of reading the WCS by reading this floppy disk image that contains the microwords loaded into the WCS. Under normal circumstances, this method of operation is effective. However, the debugger cannot be used to validate the loading process except, as described above, by sequencing through the microprogram.

3.3 USER INTERFACE

To load the Extended Writable Control Store, use the following command:

Format

```
+-----------------------------------------------------------+
|                                                           |
|     MICLD          input-file-spec,...                    |
|                                                           |
|                                                           |
|     File Qualifier                                        |
|     --------------                                        |
|                                                           |
|       /INITIAL=pattern                                    |
|                                                           |
|       /ENTRY[=hex-address]                                |
|                                                           |
+-----------------------------------------------------------+
```

Prompts

_File: input-file-spec,...

File Parameters

Input-file-spec,...
Specifies the names of one or more files to be loaded into the Extended WCS. If you specify more than one input file, you can use either the character '+' or the character ',' to separate file-specs.

Description

MICLD loads the files you give in the order specified. If you want more than one file to coexist in the WCS, then you separate the filenames with the '+' character.

In the VAX command language syntax, the ',' separator calls for the individual application of the program to each file. Thus, if you use the ',' separator, MICLD initializes the WCS and loads the first file, then initializes the WCS and loads the second file, and so on. The use of the ',' separator has little, if any, legitimate use in the loader command line.

File Qualifiers

/INITIAL=pattern
Specifies the pattern for MICLD to use to initialize the WCS. Pattern is a string of right-justified hexadecimal digits. Any missing digits are padded with zeroes. If you do not give this qualifier, the default pattern given in Section 3.1.2 is used.

/ENTRY[=hex-address]
Specifies the address at which the microprogram in the extended WCS begins. If you do not specify a hex-address, the loader assumes that the microprogram begins at the address of the first word in the first ULD. If you do not give this qualifier, you must set the starting address as described in Section 4.1.2.

Examples

1. MICLD ALPHA+BETA

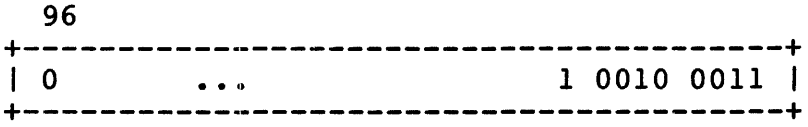
   MICLD initializes the WCS to the default pattern and loads ALPHA.ULD and then loads BETA.ULD.

2. MICLD ALPHA,BETA

   MICLD initializes the WCS to the default pattern and loads ALPHA.ULD. It then initializes the WCS again and loads BETA.ULD.

3. MICLD/INITIAL=123 ALPHA

   MICLD initializes the WCS to the pattern specified, namely:



   Then, it loads ALPHA.ULD.

4. MICLD/INITIAL=123/ENTRY=1400 ALPHA

   MICLD initializes the WCS to the pattern specified by 123 (as shown above), sets the entry vector to begin execution at address 1400 (hex), and loads the file ALPHA.ULD.

3.4 PROGRAM BEHAVIOR

After you give MICLD the list of files, it loads the files specified into the WCS.

If MICLD detects errors or special circumstances (such as over-loading), it issues a message. At the completion of the loading process, MICLD issues a final message indicating whether or not the loading process was successful.

3.4.1 VAX-11/780 WCS Architecture Description

The Extended WCS is 96 bits wide by 2K occupying VAX addresses 1800 (HEX) through 1FFF. The WCS is divided into three 32 bit X 2K pieces, termed banks. These banks are referenced as BANK 0, BANK 1, and BANK 2 (see Figure 3-2). MICLD loads a microprogram into the Extended WCS one bank at a time. That is, it breaks each microword into three 32-bit pieces and then loads the pieces consecutively into BANK 0, 1, and 2.

Two VAX 11/780 Processor Specific registers support the WCS, namely: the WCS Address Register (WCSA) and the WCS Data Register (WCSD).

The WCSA register consists of 32 bits. The first 16 bits are not used; the last 16 bits are divided into three fields. The WCS ADDRESS field, occupying bits 0 through 12, points to the current WCS address being loaded. The BANK SELECT field (CTR), occupying bits 14 and 13, contains a value of 1, or 2 representing the current bank being loaded. The PIN field, occupying bit 15, is set to 1 if any writes are done with inverted parity. MICLD sets the PIN field to 0. The WCSA register is identified in VAX as processor register number 44.

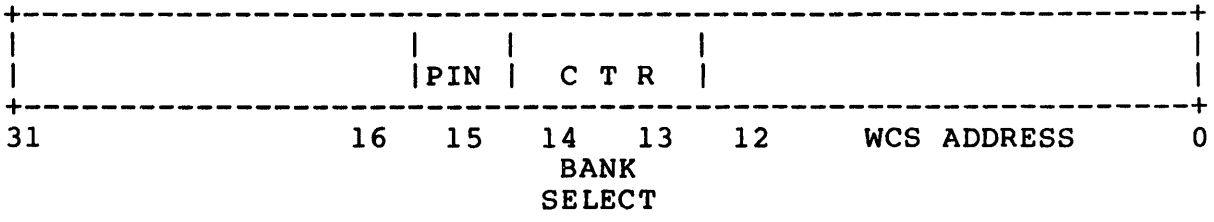


Figure 3-2 WCSA Register

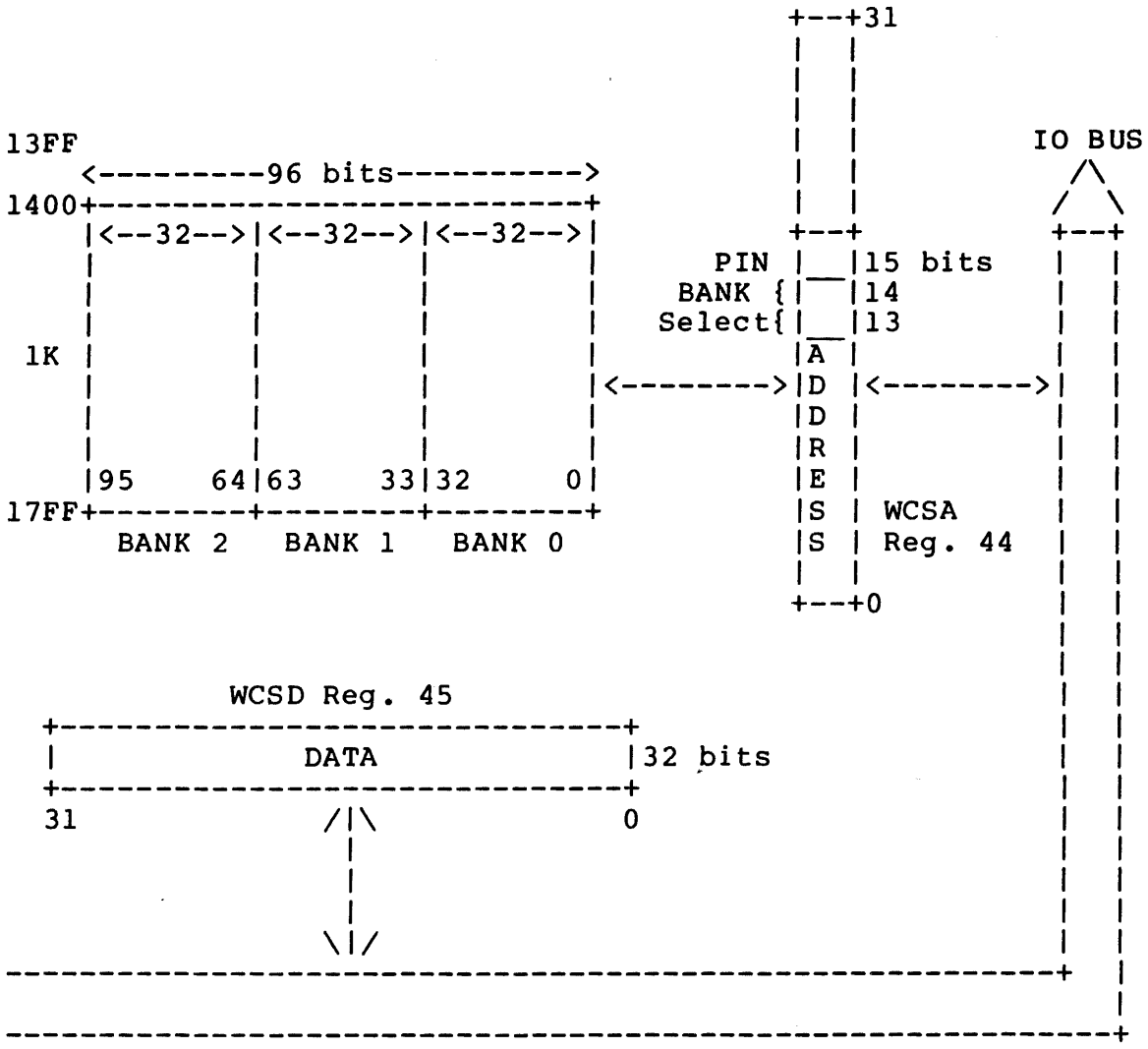


Figure 3-3

The WCSD register consists of 32 bits and contains the microcode or data to be loaded into the WCS. The WCSD register is identified in VAX as processor register number 45.

Information is written into and read from these two VAX processor registers using the Move to Processor Register (MTPR) and Move from Processor Register (MFPR) privileged instructions provided by the VAX-11 system.

To load the user WCS, MICLD must first initialize the WCSA register with the WCS address where the loader is to begin loading microcode. The Bank Select bits are set to 00. Several actions take place to load a complete microword into the WCS.

Step 1: MICLD loads the first 32 bits of the microword (destined for BANK0) into the WCSD register.

Step 2: VAX autonomously loads the data in the WCSD register into the User WCS at the location specified by the WCSA register and then auto increments the BANK SELECT field for BANK 1.

Step 3: MICLD loads the second 32 bits of the microword (destined for BANK 1) into the WCSD register.

Step 4: VAX repeats Step 2, placing data in BANK 1 at the same WCS location and auto increments the BANK SELECT field for BANK 2.

Step 5: MICLD loads the third 32 bits of the microword (destined for BANK 2) into the WCSD register, completing the WCS load of the microword.

Step 6: VAX repeats Step 2, placing the data into BANK 2, resets the BANK SELECT field for BANK 0, and increments the WCS ADDRESS field to point to the next WCS address.

MICLD is then ready to begin loading the next microword into the WCS.

3.4.2 VMS Operating System Support

Programs running under VMS execute in an assigned processor mode. Listed in ascending order of processor privilege and descending access capability, there are four processor modes for programs: 1) user, 2) supervisor, 3) executive, and 4) kernel. All programs begin at user mode and may change processor modes depending on the privileges initially assigned to it.

MICLD begins at user level. MICLD uses the Change Mode to Kernel command (CHMK) to elevate itself to kernel mode. The CHMK command is documented in the VAX 11/780 Architecture Handbook, Volume 1, Section 13-2.

Successful execution of the CHMK command and other processor mode commands depends on the program being run in an account equipped with the necessary privileges.

3.5 ERROR MESSAGES

MICLD issues error messages in the VAX standard error message format.

Namely:

%<program-name>-<severity-code>-<abbreviation>,<message>

Where:

program-name	is MICLD
severity-code	is I (information) E (error) F (fatal)
<abbreviation>	is a short identifier for the message
<message>	is the error message text

The abbreviations and messages produced by MICLD are as follows:

Abbreviation	Message
ABKEYW	ambiguous keyword
AMVERB	ambiguous verb
BADENTRY	/ENTRY value not a hex number
BADENTRY	/ENTRY value not in the user WCS address range
BADINIT	initial value conversion error
BADULD	data record conversion error
BADULD	invalid data record syntax
BADULD	missing equal sign
BADULD	missing right bracket
CHALOG	unable to log WCS content change
FNF	file not found <filename>
INVCMD	invalid command format
NOPARM	missing parameter
NOPRIV	no kernel mode privileges
NOPRIV	unable to write to error log
NOWCS	WCS memory not installed
TERMEOF	end of file on terminal

WCSBND	address out of bounds at <address>(hex)
WCSLOAD	WCSA error <address> should be <address>(hex)
WCSCHANGE <filename>	WCS content changed by <userid> using file
WCSMLO	memory location overwritten at <address>(hex)

CHAPTER 4

EXECUTING A MICROPROGRAM

To execute a microprogram in the extended WCS, the following actions are necessary:

1. An XFC instruction in a main memory program must be executed.

2. The field consisting of Bits 1 and 0 of Vector 14 of the System Control Block must be set to 2.

3. The entry vectory (10E0 hex) must be set to jump to the first instruction to be executed in the microprogram in the Extended WCS.

The following sections describe these actions in detail.

4.1 EXTENDED FUNCTION CALL

A microprogram is invoked by the XFC instruction. The Extended Function Call (XFC) instruction provides a controlled mechanism for software to request services of non-standard microcode in the extended Writeable Control Store (WCS). The request is controlled by the system control block. All opcodes reserved to the extended WCS start with FC (hex), which is the XFC instruction, using the format:

FC

The XFC instruction has no parameters.

You can pass parameters either as normal operands or in fixed registers. For example, if you have more than one extended function resident in the WCS, you can use the bytes following the opcode to specify which of several extended functions are requested.

Execution of the XFC instruction generates what is called an exception.

4.1.1 Exceptions

The notification of an event relevant primarily to the currently executing process, which invokes software in the context of the current process, is termed an exception.

Exceptions are handled by, or trapped to, operating system software. Further, all exceptions either wait for the instruction that caused them to complete before happening or they restore the processor to the state it was in prior to executing the instruction that caused the execution.

An exception caused by the execution of an XFC instruction (classified as a fault) occurs during an instruction, leaving the registers and memory in a consistent state such that elimination of the fault condition and restarting the instruction will give correct results. The XFC instruction causes faults called the opcode reserved for customer and customer reserved exceptions. The value of the PC that is saved on the stack points to the instruction faulting.

Exceptions are usually reflected to the originating mode as a signal. In general, the signal is interpreted via a vector in the system control block. Separate vectors are defined for each class of exception and interrupting device controller.

4.1.2 Setting The System Control Block Vector

When an XFC is executed, VMS handles the fault by trapping to vector 14 (hex) of the system control block (SCB). VMS examines the low order two bits of the vector and if it finds the value 2, then it traps to the entry vector (10E0 hex).

The two low order bits of Vector 14 of the SCB must be set to 2 to execute a microprogram in the Extended WCS. The console data deposit command can be used to set vector 14 relative to the base SCBB.

4.1.3 Patching The Entry Vector

After vector 14 is accessed, the system traps to address 10E0 (hex), which is resident in the VAX microcode area and is called the entry vector. The entry vector must be loaded with a JUMP microinstruction to the desired entry point in the extended WCS. This extended WCS entry point is usually a control routine or exception handler; however, it can simple be the first instruction in the microcode function to be performed.

The entry vector is set during the loading process by MICLD if an /ENTRY file qualifier is given in the MICLD command line. The entry vector can also be set at the VAX console using the console program.

Note, that when the fault occurs, the system traps (in the end) to the user WCS. The user defines an exception handler in microcode to service the "extended customer opcode" fault.

The WCS contains only one application, there is no need for a handler to resolve what function should be performed in the WCS. However, if the WCS contain several microroutines, an exception handler must resolve event by accessing additional data. The microprogrammer must have a good understanding of the VAX micro machine data path to develop this exception handler.

More information on the System Control Block and the handling of exceptions can be found in the VAX 11/780 Hardware Handbook.

APPENDIX A

VAX 11/780 FIELD AND MACRO DEFINITIONS

The VAX 11/780 Definition Language identifies the fields of the microword and provides a macro language to aid you in writing microprograms. The sample microprogram in Appendix B is written in this definition language.

When assembling a program written in this definition language, you include the definition language source by concatenating your program file with the definition file VAXDEF.MIC in the MICRO2 command line as follows:

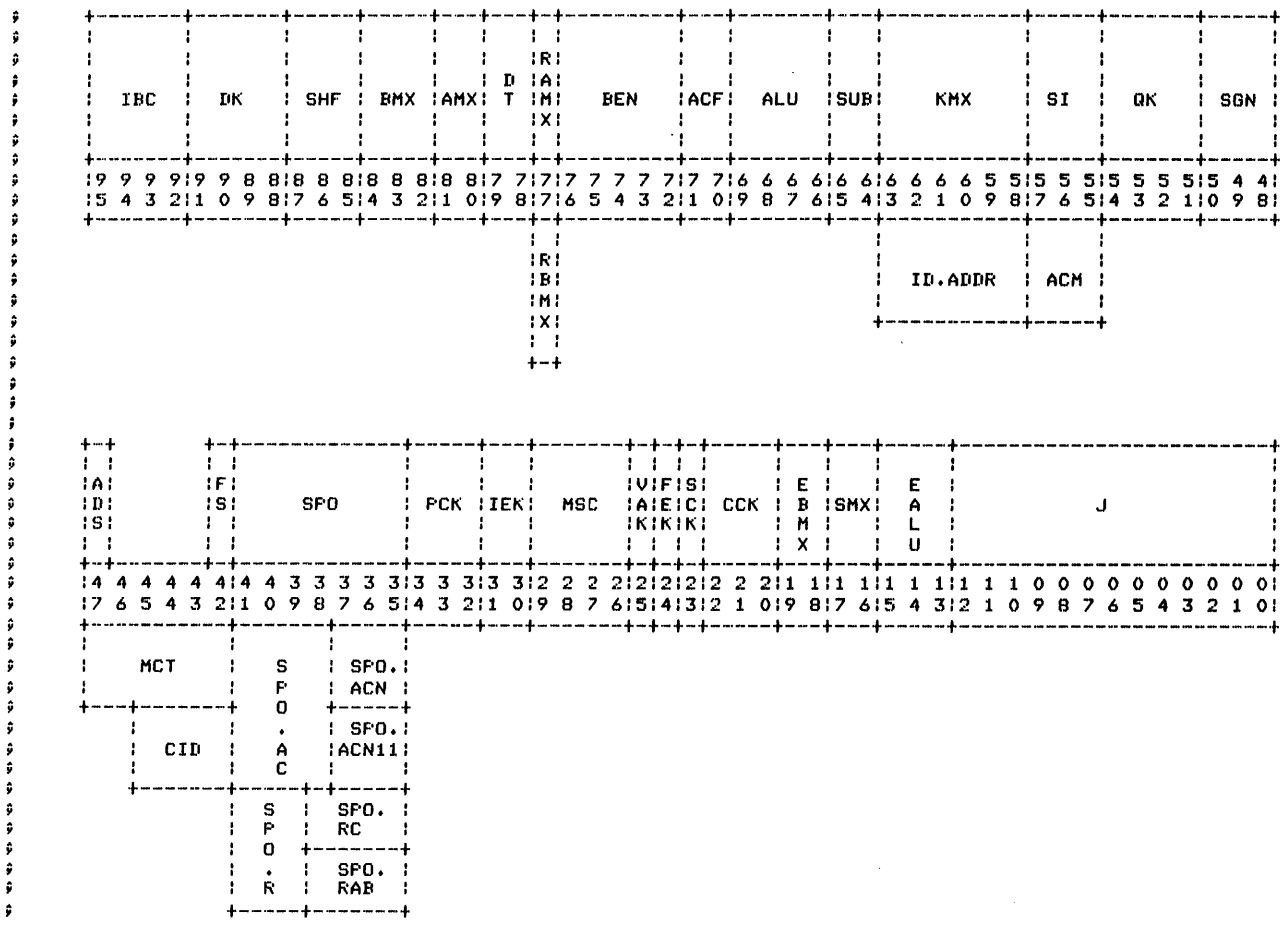
```
$ MICRO2 [SYSLIB]VAXDEF+MYPROG
```

In the above example, MYPROG.MIC is the name of the file that contains the source MICRO2 microprogram.

The definition language file is available on a floppy in the VAX 11/780 WCS kit. It is copied to SYS$LIBRARY:VAXDEF.MIC when the kit is installed on a system.

```
.TOC    "Machine definition    : Control word chart"

.NOBIN                  ;DON'T LEAVE LISTING SPACE FOR BINARY
.RTOL                   ;BITS NUMBERED FROM 0 ON THE RIGHT OF A FIELD
.HEXADECIMAL            ;RADIX 16
.NOCREF                 ;SUPPRESS CREF ON UNINTERESTING FIELDS
.SET/NATIVE=1           ;Validity check for CCK micro orders
```



```
.TOC    "Machine definition     : ACF, ACM, ADS, ALU, AMX"

ACF/=<71:70>,,DEFAULT=0         !ACCELERATOR CONTROL
        NOP=0
        SYNC=1
        TRAP=2
        CONTROL=3               !ACCELLERATOR-DEPENDENT CONTROL FUNCTION

ACM/=<57:55>                    !ACCELERATOR MISCELLANEOUS CONTROL
        PWR.UP=0
        ABORT=1                 !RETURN ACCEL TO MONITORING IRD
        POLY.DONE=6

ADS/=<47:47>                    !ADDRESS SELECT
        VA=0
        IBA=1

ALU/=<69:66>,,DEFAULT=0F        !ALU CONTROL FUNCTIONS
        A-B=00
        A-B.RLOG=01
        A-B-1=02
        INST.DEP=03                     !INSTRUCTION DEPENDENT
        A+B+1=04
        A+B=05
        A+B.RLOG=06
        ORNOT=07                        !A .OR. .NOT. B
        XOR=08                          !A .XOR. B
        ANDNOT=09                       !A .AND. .NOT. B
        NOTA=0A                         !.NOT. A
        A+B+PSL.C=0B
        OR=0C                           !A .OR. B
        AND=0D                          !A .AND. B
        B=0E
        A=0F

AMX/=<81:80>                    !AMX TO ALU
        LA=0
        RAMX=1
        RAMX.SXT=2              !RAMX SIGN EXTENDED ACCORDING TO DT
        RAMX.0XT=3              !RAMX ZERO EXTENDED. 0XT(L)=0
```

```
.TOC    "Machine definition     : BEN, BMX"

BEN/=<76:72>,,DEFAULT=0         ;BRANCH ENABLE
        NOP=0                   ;NO BRANCH
        Z=1                     ; ALU Z
        ROR=2                   ;LA<1>, PSL<C>, LA<0>
        C31=3                   ; ALU C31, 0
        IRC.ROM=4               ;OUTPUT OF EXTENDED IRC-ROM
        IB.0=5                  ;IB 0 READY ?
        ACCEL=6                 ;CODE FROM ACCELERATOR
        DATA.TYPE=8             ;(VAX MODE) *, ASRC+VSRC, ASRC+Q+D
                                ;   0--NORMAL, 1--QUAD OR DOUBLE
                                ;   2--FIELD,  3--ADDRESS
        END.DP1=8               ;(-11 MODE) *, 0 CLASS, J CLASS+DM27
        IR2-1=9                 ;(VAX MODE) *, IR<2:1>
        PC.MODES=9              ;(-11 MODE) *, SM47+SM57+DM47+DM57, DST R=PC
        REI=0A                  ;(VAX MODE) MODE.LSS.ASTLVL, *, *
        SRC.PC=0A               ;(-11 MODE) SRC R=PC
        IB.TEST=0B              ; 0--TB MISS, 1--ERROR
                                ; 2--STALL, 3--DATA OK
        MUL=0C                  ;SC.NE.0, D<1:0>
        SIGNS=0D                ;Q<31>, D.NE.0, D<31>
        INTERRUPT=0E            ;AC LOW, INTERNAL INTERRUPT, INT REQ
        DECIMAL=0F              ;0, D BYTE 0 VALID DIGIT, D2-0 NEG SIGN
        UTRAP=10                ;MICROTRAP DISPATCH VECTOR
        LAST.REF=11             ;-FPD, NESTED ERROR, LOW TWO BITS:
                                ; 0--READ INTERLOCK, 1--READ, READ CHK
                                ; 2--WRITE, 3--READ, WRITE CHK
        EALU=12                 ;EALU N, EALU Z, SC.NEQ.0, SS
        SC=14                   ;SC<9:8>.NE.0, SC.GT.0, SC<9:5>.NE.0
        ALU1-0=15               ;RLOG EMPTY, ALU<1:0>=0, ALU<1>, ALU<0>
                                ; (ALU BITS FROM PREVIOUS STATE)
        STATE7-4=16             ;STATE <7:4>
        STATE3-0=17             ;STATE <3:0>
        D.BYTES=18              ;BYTES 3, 2, 1, 0 OF D.NE.0
        D3-0=19                 ;D<3:0>
        PSL.CC=1A               ;N,Z,V,C OF PSL
        ALU=1B                  ;ALU N, ALU Z, IR<0>, ALU C31
        PSL.MODE=1C             ;-VA<31:30>, -CONSOLE, IS+CM, KERNEL
        TB.TEST=1D              ;PTE VALID, ALIGNED, QUAD, +
                                ; 0--TRANSLATION OK, 1--WR CHK AND M=0
                                ; 2--ACCESS VIOLATION, 3--TB MISS

BMX/=<84:82>                    ;BMX TO ALU
        MASK=0                  ;A 0 IN THE BIT SELECTED BY SC<4:0>
        PC.OR.LB=1              ;LB UNLESS R=PC, THEN PC
        PACKED.FL=2             ;PACKED FLOATING
        LB=3
        LC=4
        PC=5
        KMX=6
        RBMX=7                  ;D OR Q
```

```
.TOC    "Machine definition     : CCK, CID, DK, DT"

CCK/=<22:20>,.DEFAULT=0 ;CONDITION CODES
                                        ;Note : * = RESERVED OPERATION, "ALU SIGN" AND "AMX SIGN" ARE SIZE DEPENDENT
                                        ;-----------------------------------+-----------------------------------------------+
                                        ;          NATIVE MODE PSL          |          COMPATIBILITY MODE PSL               |
                                        ;-----------+---------------+---+---+-----------+---------------+---+-----------+
                                        ;     N     |       Z       | V | C |     N     |       Z       | V |     C     |
                                        ;-----------+---------------+---+---+-----------+---------------+---+-----------+
        NOP=0                           ;     N     |       Z       | 0 | C |     N     |       Z       | 0 |     C     |
        LOAD.UBCC=1                     ;     N     |       Z       | 0 | C |     N     |       Z       | 0 |     C     |
        SET.V=2             ,.VALIDITY=<V1>;     N     |       Z       | 1 | C |     *     |       *       | * |     *     |
        N_AMX.Z_TST.VC_VC=3             ; AMX SIGN  | Z.and.(ALU.eq.0) | 0 | C | AMX SIGN  | Z.and.(ALU.eq.0) | 0 |     C     |
        ROR=4               ,.VALIDITY=<V0>;     *     |       *       | * | * | ALU SIGN  |     ALU.eq.0  | 0 |  AMX<0>   |
        NZ_ALU.VC_0=5                   ; ALU SIGN  |     ALU.eq.0  | 0 | 0 | ALU SIGN  |     ALU.eq.0  | 0 |     0     |
        NZ_ALU.VC_VC=6      ,.VALIDITY=<V1>; ALU SIGN  |     ALU.eq.0  | 0 | C |           |               |   |           |
        C_AMX0=6            ,.VALIDITY=<V0>;           |               |   |   |     N     |       Z       | 0 |  AMX<0>   |
        INST.DEP=7                      ;                       Instruction dependent                                   |
                                        ;-----------+---------------+---+---+-----------+---------------+---+-----------+

CID/=<45:42>                    ;CONSOLE & ID BUS CONTROL IF FS/1
        NOP=1                   ;DEFAULT, ALLOW AUTO IB READ
        ACK=5                   ;SET CONSOLE ACKNOWLEGE FLAG
        CONT=7                  ;CLEAR CONSOLE MODE
        READ.SC=9               ;READ ID BUS REG SELECTED BY SC
        READ.KMX=0B             ;READ ID BUS REG SELECTED BY UKMX
        WRITE.SC=0D             ;WRITE REG SELECTED BY SC
        WRITE.KMX=0F            ;WRITE REG SELECTED BY UKMX

DK/=<91:88>,.DEFAULT=0
        NOP=0                   ;DEFAULT, HOLD
        LEFT2=1                 ;DOUBLE SHIFT LEFT
        RIGHT2=2                ;DOUBLE SHIFT RIGHT
        DIV=4                   ;IF NOT ALU CRY, SHIFT LEFT
                                ; ELSE LOAD FROM SHF
        LEFT=5                  ;SHIFT LEFT
        RIGHT=6                 ;SHIFT RIGHT
        SHF=8                   ;LOAD SHF MUX, INTEGER FORMAT
        SHF.FL=9                ;LOAD SHF MUX, UNPACKED FLOATING FORMAT
        ACCEL=0A                ;LOAD ACCELERATOR DATA FROM DF BUS
        BYTE.SWAP=0B            ;REFLECT BYTES AROUND BIT 16
        Q=0C                    ;LOAD Q THRU DAL
        DAL.SC=0D               ;LOAD DAL[SC]
        DAL.SV=0E               ;LOAD DAL[SHF VAL]
        CLR=0F                  ;LOAD ZEROS

DT/=<79:78>,.DEFAULT=0          ;DATA TYPE
                                ;CONTROLS AMX SIGN/ZERO EXTENDER, SHF AMOUNT,
                                ;CONDITION CODE SETTING, AND MEMORY REFERENCES
        LONG=0                  ;DEFAULT
        WORD=1
        BYTE=2
        INST.DEP=3              ;INSTRUCTION DEPENDENT -
                                ;ANY OF ABOVE, OR QUAD/DOUBLE
```

```
.TOC    "Machine definition     : EALU, EBMX, FEK, FS, IEK, IBC"

EALU/=<15:13>                   ;EXPONENT ALU
        A=0
        OR=1
        ANDNOT=2
        B=3
        A+B=4
        A-B=5
        A+1=6
        NABS.A-B=7              ;-ABS(A-B)

EBMX/=<19:18>                   ;EBMX TO EALU
        FE=0                    ;DEFAULT
        MX=1
        AMX.EXP=2
        SHF.VAL=3               ;SHIFT VALUE

FEK/=<24:24>,.DEFAULT=0         ;FE REGISTER CONTROL
        NOP=0                   ;DEFAULT, HOLD
        LOAD=1

FS/=<42:42>                     ;FUNCTION SELECT FOR 43-46
        MCT=0                   ;ENABLE MEMORY CONTROL
        CID=1                   ;ENABLE ID BUS AND CONSOLE CONTROL

IEK/=<31:30>                    ;INTERRUPT AND EXCEPTION ACKNOWLEDGE
        NOP=0
        ISTR=1                  ;STROBE INTERRUPT REQUESTS
        IACK=2                  ;INTERRUPT ACKNOWLEDGE
        EACK=3                  ;EXCEPTION ACKNOWLEDGE

IBC/=<95:92>,.DEFAULT=0         ;IBUF CONTROL FUNCTIONS
        NOP=0                   ;DEFAULT
        STOP=1
        FLUSH=2                 ;FLUSH IB AND LOAD IBA
        START=3
        CLR.0.1=4               ;CLEAR BYTES 0,1 (-11 OPCODE)
        CLR.2.3=5               ;CLEAR BYTES 2,3 (-11 ISTREAM DATA)
        BDEST=7                 ;TRANSFER BRANCH DISPLACEMENT
        CLR.0=0C                ;CLEAR BYTE 0 (VAX OPCODE)
        CLR.1=0D                ;CLEAR BYTE 1 (VAX SPECIFIER)
        CLR.0-3=0E              ;CLEAR BYTES 0-3 (-11 OP & DATA)
        CLR.1-5.COND=0F         ;CLEAR BYTES 1-5 CONDITIONALLY
                                ; IF THERE IS NO SPECIFIER EVALUATION,
                                ; CLEAR NOTHING.  IF A SELF-CONTAINED
                                ; SPECIFIER, CLEAR IT.  IF IMMEDIATE,
                                ; ABSOLUTE, OR DISPLACEMENT, CLEAR THE
                                ; ISTREAM LITERAL.
```

```
.TOC    "Machine definition     : ID.ADDR, J"

ID.ADDR/=<63:58>                #ID BUS ADDRESS
        IBUF=0                  #RD     #SPECIFIER/LITERAL DATA FROM IB
        DAY.TIME=1              #RD+WR  #CURRENT TIME OF DAY...
                                #  MUST READ UNTIL STOPS CHANGING
        SYS.ID=3                #RD     #SYSTEM IDENTIFICATION
        RXCS=4                  #RD+WR  #CONSOLE RECIEVE CONTROL/STATUS
        RXDB=5                  #RD     #CONSOLE RECIEVE DATA BUFFER
                                #  (TO-ID REGISTER)
        TXCS=6                  #RD+WR  #CONSOLE TRANSMIT CONTROL/STATUS
        TXDB=7                  #WR     #CONSOLE TRANSMIT DATA BUFFER
                                #  (FROM-ID REGISTER)
;       DQ=8                            #DATA PATH D/Q REGISTERS (MAINT ONLY)
        NXT.PER=9               #WR     #INTERVAL CLOCK NEXT PERIOD REGISTER
        CLK.CS=0A               #RD+WR  #INTERVAL CLOCK CONTROL/STATUS
        INTERVAL=0B             #RD     #CURRENT INTERVAL COUNT
        CES=0C                  #RD+WR  #CPU ERROR/STATUS
        VECTOR=0D               #RD+WR  #EXCEPTION & INTERRUPT CONTROL
        SIR=0E                  #RD+WR  #SOFTWARE INTERRUPT REGISTER
        PSL=0F                  #RD+WR  #PROCESSOR STATUS LONGWORD
        TBUF=10                         #TRANSLATION BUFFER DATA
        TBER0=12                        #TB ERROR/STATUS 0
        TBER1=13                        #TB ERROR/STATUS 1
        ACC.0=14                        #ACCELERATOR REGISTER #0
        ACC.1=15                        #ACCELERATOR REGISTER #1
        ACC.2=16                        #ACCELERATOR REGISTER #2
        ACC.CS=17                       #ACCELERATOR CONTROL/STATUS
        SILO=18                         #NEXT ITEM FROM SBI HISTORY
        SBI.ERR=19                      #SBI ERROR REGISTER
        TIME.ADDR=1A                    #SBI TIMEOUT ADDRESS
        FAULT=1B                        #FAULT/STATUS
        COMP=1C                         #SBI SILO COMPARATOR
        MAINT=1D                        #SBI MAINTENANCE
        PARITY=1E                       #CACHE PARITY
        USTACK=20                       #MICROSTACK
        UBREAK=21                       #MICRO BREAK
        WCS.ADDR=22
        WCS.DATA=23             #WR     #WRITING WCS COUNTS ADDRESS
```

```
;ID BUS ADDRESSES CONTINUED.  ADDRESSES 24-3F ARE RAM LOCATIONS

        P0BR=24                         ;PROCESS SPACE 0 BASE REGISTER
        P1BR=25                         ;PROCESS SPACE 1 BASE REGISTER
        SBR=26                          ;SYSTEM SPACE BASE REGISTER
        KSP=28                          ;KERNEL STACK POINTER
        ESP=29                          ;EXEC STACK POINTER
        SSP=2A                          ;SUPERVISOR STACK POINTER
        USP=2B                          ;USER STACK POINTER
        ISP=2C                          ;INTERRUPT STACK POINTER
        FPDA=2D
        D.SV=2E
        Q.SV=2F
        T0=30                           ;GENERAL TEMPS
        T1=31
        T2=32
        T3=33
        T4=34
        T5=35
        T6=36
        T7=37
        T8=38
        T9=39
        PCBB=3A                         ;PROCESS CONTROL BLOCK BASE
        SCBB=3B                         ;SYSTEM CONTROL BLOCK BASE
        P0LR=3C                         ;PROCESS 0 LENGTH REGISTER
        P1LR=3D                         ;PROCESS 1 LENGTH REGISTER
        SLR=3E                          ;SYSTEM LENGTH REGISTER

.CREF

J/=<12:0>,.NEXTADDRESS                  ;NEXT MICRO WORD ADDRESS

.NOCREF
```

```
.TOC    "Machine definition     : KMX"

KMX/=<63:58>            ;CONSTANTS OR # FROM FK
        .8=0            ;#8 FROM FK
        .1=1            ;#1 FROM FK
        .2=2            ;#2 FROM FK
        .3=3            ;#3 FROM FK
        .4=4            ;#4 FROM FK
        SP1.CON=5       ;SPECIFIER 1 CONSTANT
        SP2.CON=6       ;SECIFIER 2 CONSTANT (-11 MODE)
        ZERO=6          ; OR ZEROS (VAX MODE)
        SC=7            ;SC[9:0] FROM FK

;8 - 3F: CONSTANTS (1 CYCLE SETUP IF ALU IN ARITH MODE)
                        ;DECIMAL VALUE OF CONSTANT
        .14=8           ;20
        .A0=9           ;160
        .34=0A          ;52
        .28=0B          ;40
        .40=0C          ;64
        .50=0D          ;80
        .7FF0=0E        ;******* .3000=0E If system rev is less than 6 *******
        .EF=0F          ;239
        .80=10          ;128
        .8000=11        ;-32768
        .FF=12          ;255
        .FF00=13        ;-256
        .1E=14          ;30
        .3F=15          ;63
        .7F=16          ;127
        .7=17           ;7
        .F=18           ;15
        .10=19          ;16
        .FFE8=1A        ;-24
        .FFF0=1B        ;-16
        .FFF8=1C        ;-8
        .20=1D          ;32
        .30=1E          ;48
        .18=1F          ;24
        .3FF=20         ;1023
        .C=21           ;12
        .D=22           ;13
        .1F=23          ;31
        .1F00=24        ;7936
        .B0=25          ;176
        .E003=26        ;
        .7C=27          ;124
        .FFE0=28        ;-32
        .60=29          ;96
;       SPARE=2A
        .DFCF=2B        ;?
        .4000=2C        ;******* .FFEF=2C If system rev is less than 6 *******
        .FFF1=2D        ;-15
        .19=2E          ;25
        .FFF9=2F        ;-7
```

```
; KMX DEFINITION CONTINUED

        .FFFF=30        ;-1
        .88=31          ;1
        .3030=32        ;?
        .F0=33          ;240
        .C0=34          ;192
        .6=35           ;6
        .9=36           ;9
        .FFF6=37        ;-10
        .FFF5=38        ;-11
        .1A=39          ;26
        .24=3A          ;36
        .1B=3B          ;27
        .FFFC=3C        ;-4
        .A=3D           ;10
        .7E=3E          ;126
;       SPARE=3F
```

```
.TOC    "Machine definition    : MCT, MSC"

MCT/=<47:42>,.DEFAULT=3E                ;MEMORY CONTROL
        TEST.RCHK=00                    ;TEST TBUF WITH READ CHECK
        MEM.NOP=02                      ;NEITHER CPU NOR IB GETS MEM CYCLE
        TEST.WCHK=04                    ;TEST TBUF WITH WRITE CHECK
        WRITE.V.NOCHK=0A                ;WRITE, INHIBIT TRAPS
        WRITE.V.WCHK=0C                 ;WRITE, NORMAL VARIETY
        LOCKWRITE.V.XCHK=0E             ;INTERLOCK WRITE, VIRTUAL ADDRESS
        READ.V.RCHK=10                  ;READ, NORMAL VARIETY
        READ.V.NOCHK=12                 ;READ, INHIBIT TRAPS
        READ.V.WCHK=14                  ;READ FOR MODIFY
        READ.V.IBCHK=16                 ;READ, CHECK CONTROLLED BY IBUFFER
        READ.V.NEWPC=18                 ;BEGIN NEW INSTRUCTION STREAM
                                        ; DATA GOES TO INSTRUCTION BUFFER
        LOCKREAD.V.NOCHK=1A             ;INTERLOCK READ, INHIBIT CHECK
        LOCKREAD.V.WCHK=1C              ;INTERLOCK READ, NORMAL VARIETY
        SBI.HOLD=20                     ;STOP ALL SBI ACTIVITY
        SBI.HOLD+UNJAM=22               ;RESET SBI
        INVALIDATE=24                   ;CLEAR CACHE ENTRIES
        VALIDATE=26                     ;MICRODIAGNOSTIC FORCE VALID
        EXTWRITE.P=28                   ;EXTENDED WRITE TO CLEAR MOS ERRORS
        WRITE.P=2A                      ;WRITE, PHYSICAL
        LOCKWRITE.P=2E                  ;INTERLOCK WRITE, PHYSICAL
        READ.P=32                       ;READ, PHYSICAL
        READ.INT.SUM=36                 ;INTERRUPT SUMMARY READ
        LOCKREAD.P=3A                   ;INTERLOCK READ, PHYSICAL
        ALLOW.IB.READ=3E                ;GIVE IB A CYCLE IF IT WANTS ONE

MSC/=<29:26>,.DEFAULT=0
        NOP=0                           ;DEFAULT
        CHK.CHM=01                      ;CREATE NEW PSL FOR CHM
        CHK.FLT.OPR=02                  ;UTRAP IF ALU<15>=1, ALU<14:7>=0
        CHK.ODD.ADDR=03
        IRD=04                          ;THIS STATE IS INSTRUCTION DECODE
        LOAD.STATE=05
        LOAD.ACC.CC=06                  ;TAKE CONDITION CODES FROM ACCELERATOR
        READ.RLOG=07                    ;(AND POP RLOG STACK)
        CLR.FPD=08                      ;CLEAR PSL<FPD> BIT
        SET.FPD=09                      ;SET SAME
        CLR.NEST.ERR=0A                 ;CLR NESTED ERROR FLAG IN CPU STATUS
        SET.NEST.ERR=0B                 ;SET SAME
        SECOND.REF=0C                   ;OF UNALIGNED DATA REFERENCE
        RETRY.NO.TRAP=0D                ;APPLY SAVED CONTEXT, INHIBIT TRAPS
        RETRY.TRAP=0E                   ;APPLY SAVED CONTEXT TO THIS REF
        INH.CM.ADDR=0F                  ;ALLOW USE OF FULL 32-BIT ADDRESS
```

```
.TOC    "Machine definition     : PCK, QK, RAMX, RBMX"

PCK/=<34:32>,.DEFAULT=0         !ADDRESS COUNT CONTROL
        NOP=0                   !DEFAULT
        PC_VA=1
        PC_IBA=2
        VA+4=3                  !VA_VA+4
        PC+1=4                  !PC_PC+1
        PC+2=5                  !PC_PC+2
        PC+4=6                  !PC_PC+4
        PC+N=7                  !PC_PC+N,  N IS DETERMINED BY INSTR BUFFER

QK/=<54:51>,.DEFAULT=0
        NOP=0                   !DEFAULT, HOLD
        LEFT2=1                 !DOUBLE SHIFT LEFT 2
        RIGHT2=2                !DOUBLE SHIFT RIGHT 2
        LEFT=5
        RIGHT=6
        SHF=8                   !LOAD SHF, INTEGER FORMAT
        SHF.FL=9                !LOAD SHF, UNPACKED FLOATING FORMAT
        DEC.CON=0A              !DECIMAL CONSTANT = 6'S IN EACH NIBBLE
                                !FOR WHICH ALU CRY OUT IS FALSE
        ACCEL=0B                !LOAD ACCELERATOR DATA FROM DF BUS
        D=0C
        ID=0E                   !LOAD ID BUS
        CLR=0F                  !LOAD ZERO

RAMX/=<77:77>,.DEFAULT=0        !DATA PATH MIXER TO AMX
        D=0                     !DEFAULT
        Q=1

RBMX/=<77:77>                   !DATA PATH MIXER TO BMX.  SAME BIT AS RAMX
        Q=0
        D=1
```

```
.TOC    "Machine definition    : SCK, SGN, SHF, SI, SMX"

SCK/=<23:23>,.DEFAULT=0         !SC REGISTER CONTROL
        NOP=0                   !DEFAULT, HOLD
        LOAD=1                  !LOAD SMX<09:00>

SGN/=<50:48>,.DEFAULT=0         !SIGN CONTROLS
        NOP=0                   !DEFAULT
        LOAD.SS=1               !SS_ALU<15>
        SS.FROM.SD=2            !SS_SD
        NOT.SD=3                !SD_NOT SD
        SD.FROM.SS=4            !SD_SS
        SS.XOR.ALU=5            !SD_ALU<15>, SS_SS.XOR.ALU<15>
        ADD.SUB=6               !SD_ALU<15>, SS_SS.XOR.ALU<15>.XOR.IR<1>
        CLR.SD+SS=7             !CLEAR BOTH

SHF/=<87:85>,.DEFAULT=0         !ALU SHIFTER CONTROLS
        ALU=0                   !DEFAULT, SHF_ALU
        LEFT=1                  !SHF_ALU(L1), INSERT SI CNTL
        RIGHT=2                 !SHF_ALU(R1), INSERT SI CNTL
        ALU.DT=3                !SHF_ALU(DT: L0,L1,L2,L3), INSERT 0
        RIGHT2=4                !SHF_ALU(R2), INSERT SI CNTL
        LEFT3=5                 !SHF_ALU(L3)

SI/=<57:55>,.DEFAULT=3          !SHIFT INPUT CONTROLS
                                !       SHF     D       Q
                                !       ---     -       -
        DIVD=0                  !       PSL<N>  Q31     ALU C31
        ASHR=1                  !       ALU 31  Q0      Q31
        ASHL=2                  !       0       0       D31
        ZERO=3                  !       0       0       0
!       SPARE=4
        DIV=5                   !       Q31     Q31     ALU C31
        MUL+=6                  !       0       ALU 0,1 0
        MUL-=7                  !       1       ALU 0,1 1

SMX/=<17:16>                    !MIXER TO SC
        EALU=0                  !EALU <9:0>
        FE=1                    !FE<9:0>
        ALU=2                   !ALU<09:00>
        ALU.EXP=3               !ALU<14:07>
```

```
.TOC    "Machine definition    : SPO, SPO.AC, SPO.ACN, SPO.ACN11, SPO.R"

SPO/=<41:35>,.DEFAULT=0                 ;SCRATCH PAD OPCODE, 7 BITS
        NOP=0                           ;DEFAULT
        LOAD.LC.SC=6                    ;LOAD LC, ADR=SC[03:00]
        WRITE.RC.SC=7                   ;WRITE RC, ADR=SC[03:00]

SPO.AC/=<41:38>                         ;4 FUNCTION BITS OF SPO FIELD
        LOAD.LAB=1                      ;LOAD LA, LB FROM R(ACN)
        LOAD.LA=2                       ;LOAD LA_RN, HOLD LB
        WRITE.RAB=3                     ;WRITE RA, RB (ACN)

SPO.ACN/=<37:35>                ;AC NUMBER IN SPO FIELD
                                ;VAX MODE       RA              RB
        SP1.SP1=0               ;0              SP1 R           SP1 R
        SP2.SP2=1               ;1              SP2 R           SP2 R
        SP2.SP1=2               ;2              SP2 R           SP1 R
        PRN=3                   ;3              PRN             PRN
        PRN+1=4                 ;4              PRN+1           PRN+1
        SC=5                    ;5              SC<03:00>       SC<03:00>
        SP1+1=6                 ;6      SP1 R+1         SP1 R+1

SPO.ACN11/=<37:35>              ;AC NUMBER IN SPO FIELD -- 11 MODE
                                ;-11 MODE          RA              RB
        SRC.SRC=0               ;   0           SRC R           SRC R
        DST.DST=1               ;   1           DST R           DST R
        DST.SRC=2               ;   2           DST R           SRC R
;       SRC.SRC=3               ;   3           SRC R           SRC R
        SRC.OR.1=4              ;   4           SRC R .OR. 1    SRC R .OR. 1
        SC=5                    ;   5           SC<03:00>       SC<03:00>

SPO.R/=<41:39>                  ;SCRATCH PAD FUNCS WITH LOW 4 BITS OF SP AS ADR
        LOAD.LC=2               ;LOAD LC, ADR=SPO.RN
        WRITE.RC=3              ;WRITE RC
        LOAD.LAB=4              ;LOAD LA, LB
        WRITE.RAB=5             ;WRITE RA, RB
        LOAD.LAB1.WRITE.RC=6    ;LOAD LA, LB[R1], AND WRITE RC[RN]
        LOAD.LC.WRITE.RAB1=7    ;LOAD LC[RN], AND WRITE RA, RB[R1]
```

```
.TOC    "Machine definition     : SPO.RAB, SPO.RC, SUB, VAK"

SPO.RAB/=<38:35>                #RA/RB LOCATIONS
        R0=0
        R1=1
        R2=2
        R3=3
        R4=4
        R5=5
        R6=6
        R7=7
        AP=0C                   #R12 = ARGUMENT LIST POINTER
        FP=0D                   #R13 = STACK FRAME POINTER
        SP=0E                   #R14 = STACK POINTER
        R15=0F                  #R15 = PC, TO SOFTWARE, SCRATCH TO UCODE

SPO.RC/=<38:35>                 #RC LOCATIONS
        T0=0
        T1=1
        T2=2
        T3=3
        T4=4
        T5=5
        T6=6
        T7=7
        LC.SV=8                 #MEM MGMT SAVES LC HERE
        VA.SV=9
        PTE.VA=0A
        PTE.PA=0B
        PC.SV=0C
        SC.SV=0D
        VA.REF=0E
        MBIT.VA=0F
        PTE.MASK=0F

SUB/=<65:64>,.DEFAULT=0         #SUBROUTINE CONTROL
        NOP=0                   #DEFAULT
        CALL=1                  #PUSH UPC OF THIS MICROINSTRUCTION
                                # ONTO USTACK
        RET=2                   #"OR" TOP OF USTACK TO UPC
                                # AND POP USTACK
        SPEC=3                  #REPLACE LOW 8 BITS OF NEXT
                                # UPC WITH SPECIFIER DECODE FROM
                                # INSTRUCTION BUFFER

VAK/=<25:25>,.DEFAULT=0
        NOP=0                   #DEFAULT
        LOAD=1                  #LOAD VA
```

```
.TOC    "Machine definition     : Validity checks"

        .SET/V0=<.NOT[<NATIVE>]>
        .SET/V1=<NATIVE>

.CREF                           ;RE-ENABLE CROSS REFERENCE
```

```
.TOC    "Macro definition      ; Register transfer macros"

ALU_-1                          "AMX/RAMX.OXT,DT/LONG,ALU/NOTA"
ALU_0(A)                        "AMX/RAMX.OXT,DT/LONG,ALU/A"
ALU_0+D                         "AMX/RAMX.OXT,DT/LONG,RBMX/D,BMX/RBMX,ALU/A+B"
ALU_0+D+1                       "AMX/RAMX.OXT,DT/LONG,RBMX/D,BMX/RBMX,ALU/A+B+1"
ALU_0+K[]                       "KMX/@1,BMX/KMX,AMX/RAMX.OXT,DT/LONG,ALU/A+B"
ALU_0+K[]+1                     "KMX/@1,BMX/KMX,AMX/RAMX.OXT,DT/LONG,ALU/A+B+1"
ALU_0+LB+1                      "AMX/RAMX.OXT,DT/LONG,BMX/LB,ALU/A+B+1"
ALU_0+LC                        "AMX/RAMX.OXT,DT/LONG,BMX/LC,ALU/A+B"
ALU_0+LC+1                      "AMX/RAMX.OXT,DT/LONG,BMX/LC,ALU/A+B+1"
ALU_0+MASK+1                    "AMX/RAMX.OXT,DT/LONG,BMX/MASK,ALU/A+B+1"
ALU_0+Q                         "AMX/RAMX.OXT,DT/LONG,RBMX/Q,BMX/RBMX,ALU/A+B"
ALU_0+Q+1                       "AMX/RAMX.OXT,DT/LONG,RBMX/Q,BMX/RBMX,ALU/A+B+1"
ALU_0-D                         "AMX/RAMX.OXT,DT/LONG,RBMX/D,BMX/RBMX,ALU/A-B"
ALU_0-D-1                       "AMX/RAMX.OXT,DT/LONG,RBMX/D,BMX/RBMX,ALU/A-B-1"
ALU_0-K[]                       "AMX/RAMX.OXT,DT/LONG,KMX/@1,BMX/KMX,ALU/A-B"
ALU_0-K[]-1                     "KMX/@1,BMX/KMX,AMX/RAMX.OXT,DT/LONG,ALU/A-B-1"
ALU_0-LB                        "AMX/RAMX.OXT,DT/LONG,BMX/LB,ALU/A-B"
ALU_0-LC                        "AMX/RAMX.OXT,DT/LONG,BMX/LC,ALU/A-B"
ALU_0-LC-1                      "AMX/RAMX.OXT,DT/LONG,BMX/LC,ALU/A-B-1"
ALU_0-Q                         "AMX/RAMX.OXT,DT/LONG,RBMX/Q,BMX/RBMX,ALU/A-B"
ALU_0-Q-1                       "AMX/RAMX.OXT,DT/LONG,RBMX/Q,BMX/RBMX,ALU/A-B-1"
ALU_0[]D                        "ALU/@1,AMX/RAMX.OXT,LONG,BMX/RBMX,RBMX/D"
ALU_0[]LC                       "ALU/@1,AMX/RAMX.OXT,LONG,BMX/LC"
ALU_D                           "RAMX/D,AMX/RAMX,ALU/A"
ALU_D(B)                        "RBMX/D,BMX/RBMX,ALU/B"
ALU_D+K[]                       "RAMX/D,AMX/RAMX,KMX/@1,BMX/KMX,ALU/A+B"
ALU_D+K[]+1                     "RAMX/D,AMX/RAMX,KMX/@1,BMX/KMX,ALU/A+B+1"
ALU_D+K[].RLOG                  "AMX/RAMX,RAMX/D,KMX/@1,BMX/KMX,ALU/A+B.RLOG"
ALU_D+LB                        "RAMX/D,AMX/RAMX,BMX/LB,ALU/A+B"
ALU_D+LC                        "RAMX/D,AMX/RAMX,BMX/LC,ALU/A+B"
ALU_D+LC+1                      "RAMX/D,AMX/RAMX,BMX/LC,ALU/A+B+1"
ALU_D+LC+PSL.C                  "RAMX/D,AMX/RAMX,BMX/LC,ALU/A+B+PSL.C"
ALU_D+Q                         "RAMX/D,AMX/RAMX,RBMX/Q,BMX/RBMX,ALU/A+B"
ALU_D+Q+1                       "RAMX/D,AMX/RAMX,RBMX/Q,BMX/RBMX,ALU/A+B+1"
ALU_D+Q+PSL.C                   "ALU/A+B+PSL.C,AMX/RAMX,BMX/RBMX,RBMX/Q,RAMX/D"
ALU_D+RLOG                      "ALU/A+B,AMX/RAMX,RAMX/D,BMX/0,MSC/READ.RLOG"
ALU_D-K[]                       "RAMX/D,AMX/RAMX,KMX/@1,BMX/KMX,ALU/A-B"
ALU_D-K[]-1                     "RAMX/D,AMX/RAMX,KMX/@1,BMX/KMX,ALU/A-B-1"
ALU_D-LB                        "RAMX/D,AMX/RAMX,BMX/LB,ALU/A-B"
ALU_D-LB.RLOG                   "RAMX/D,AMX/RAMX,BMX/LB,ALU/A-B.RLOG"
ALU_D-LC                        "RAMX/D,AMX/RAMX,BMX/LC,ALU/A-B"
ALU_D-LC-1                      "RAMX/D,AMX/RAMX,BMX/LC,ALU/A-B-1"
ALU_D-Q                         "RAMX/D,AMX/RAMX,RBMX/Q,BMX/RBMX,ALU/A-B"
ALU_D-Q-1                       "RAMX/D,AMX/RAMX,RBMX/Q,BMX/RBMX,ALU/A-B-1"
ALU_D.OXT[]                     "RAMX/D,AMX/RAMX.OXT,DT/@1,ALU/A"
ALU_D.OXT[]+K[]                 "RAMX/D,AMX/RAMX.OXT,DT/@1,KMX/@2,BMX/KMX,ALU/A+B"
ALU_D.OXT[]+LC                  "ALU/A+B,AMX/RAMX.OXT,DT/@1,RAMX/D,BMX/LC"
ALU_D.OXT[]+Q                   "ALU/A+B,AMX/RAMX.OXT,DT/@1,RAMX/D,BMX/RBMX,RBMX/Q"
ALU_D.OXT[]-K[]                 "RAMX/D,AMX/RAMX.OXT,DT/@1,KMX/@2,BMX/KMX,ALU/A-B"
ALU_D.OXT[]-Q                   "RAMX/D,AMX/RAMX.OXT,DT/@1,RBMX/Q,BMX/RBMX,ALU/A-B"
ALU_D.OXT[].AND.K[]             "RAMX/D,AMX/RAMX.OXT,DT/@1,KMX/@2,BMX/KMX,ALU/AND"
ALU_D.OXT[].ANDNOT.K[]          "ALU/ANDNOT,AMX/RAMX.OXT,DT/@1,RAMX/D,BMX/KMX,KMX/@2"
ALU_D.OXT[].OR.Q                "RAMX/D,AMX/RAMX.OXT,DT/@1,BMX/RBMX,ALU/OR"
ALU_D.AND.K[]                   "RAMX/D,AMX/RAMX,KMX/@1,BMX/KMX,ALU/AND"
ALU_D.AND.MASK                  "RAMX/D,AMX/RAMX,BMX/MASK,ALU/AND"
ALU_D.ANDNOT.K[]                "RAMX/D,AMX/RAMX,KMX/@1,BMX/KMX,ALU/ANDNOT"
ALU_D.ANDNOT.MASK               "RAMX/D,AMX/RAMX,BMX/MASK,ALU/ANDNOT"
ALU_D.ANDNOT.Q                  "RAMX/D,AMX/RAMX,RBMX/Q,BMX/RBMX,ALU/ANDNOT"
ALU_D.OR.K[]                    "RAMX/D,AMX/RAMX,KMX/@1,BMX/KMX,ALU/OR"
ALU_D.OR.LC                     "RAMX/D,AMX/RAMX,BMX/LC,ALU/OR"
ALU_D.OR.Q                      "RAMX/D,AMX/RAMX,RBMX/Q,BMX/RBMX,ALU/OR"
```

```
ALU_D.OR.RC[]                   "RAMX/D,AMX/RAMX,SPO.R/LOAD.LC,SPO.RC/@1,BMX/LC,ALU/OR"
ALU_D.ORNOT.MASK                "RAMX/D,AMX/RAMX,BMX/MASK,ALU/ORNOT"
ALU_D.SXT[]                     "RAMX/D,AMX/RAMX.SXT,DT/@1,ALU/A"
ALU_D.SXT[]+K[]                 "RAMX/D,AMX/RAMX.SXT,DT/@1,KMX/@2,BMX/KMX,ALU/A+B"
ALU_D.SXT[]+Q                   "RAMX/D,AMX/RAMX.SXT,DT/@1,BMX/RBMX,ALU/A+B"
ALU_D.SXT[].ANDNOT.K[]          "RAMX/D,AMX/RAMX.SXT,DT/@1,ALU/ANDNOT,BMX/KMX,KMX/@2"
ALU_D.SXT[].AND.K[]             "RAMX/D,AMX/RAMX.SXT,DT/@1,KMX/@2,BMX/KMX,ALU/AND"
ALU_D.XOR.K[]                   "RAMX/D,AMX/RAMX,KMX/@1,BMX/KMX,ALU/XOR"
ALU_D.XOR.LC                    "RAMX/D,AMX/RAMX,BMX/LC,ALU/XOR"
ALU_D.XOR.Q                     "RAMX/D,AMX/RAMX,RBMX/Q,BMX/RBMX,ALU/XOR"
ALU_D.XOR.RC[]                  "RAMX/D,AMX/RAMX,SPO.R/LOAD.LC,SPO.RC/@1,BMX/LC,ALU/XOR"
ALU_D.XOR.R[]                   "RAMX/D,AMX/RAMX,SPO.R/LOAD.LAB,SPO.RAB/@1,BMX/LB,ALU/XOR"
ALU_D[]K[]                      "RAMX/D,AMX/RAMX,KMX/@2,BMX/KMX,ALU/@1"
ALU_D[]LB                       "ALU/@1,AMX/RAMX,RAMX/D,BMX/LB"
ALU_D[]LC                       "RAMX/D,AMX/RAMX,BMX/LC,ALU/@1"
ALU_D[]Q                        "RAMX/D,AMX/RAMX,RBMX/Q,BMX/RBMX,ALU/@1"
ALU_K[]                         "KMX/@1,BMX/KMX,ALU/B"
ALU_LA                          "AMX/LA,ALU/A"
ALU_LA+K[]                      "AMX/LA,KMX/@1,BMX/KMX,ALU/A+B"
ALU_LA+K[]+1                    "ALU/A+B+1,AMX/LA,BMX/KMX,KMX/@1"
ALU_LA+K[].RLOG                 "AMX/LA,KMX/@1,BMX/KMX,ALU/A+B.RLOG"
ALU_LA+LB                       "AMX/LA,BMX/LB,ALU/A+B"
ALU_LA+LC                       "ALU/A+B,AMX/LA,BMX/LC"
ALU_LA+LC+1                     "ALU/A+B+1,AMX/LA,BMX/LC"
ALU_LA+LC+PSL.C                 "ALU/A+B+PSL.C,AMX/LA,BMX/LC"
ALU_LA+Q                        "ALU/A+B,AMX/LA,BMX/RBMX,RBMX/Q"
ALU_LA-D                        "AMX/LA,RBMX/D,BMX/RBMX,ALU/A-B"
ALU_LA-D-1                      "AMX/LA,RBMX/D,BMX/RBMX,ALU/A-B-1"
ALU_LA-K[]                      "AMX/LA,KMX/@1,BMX/KMX,ALU/A-B"
ALU_LA-K[]-1                    "AMX/LA,KMX/@1,BMX/KMX,ALU/A-B-1"
ALU_LA-K[].RLOG                 "AMX/LA,KMX/@1,BMX/KMX,ALU/A-B.RLOG"
ALU_LA-LC                       "ALU/A-B,AMX/LA,BMX/LC"
ALU_LA-Q                        "ALU/A-B,AMX/LA,BMX/RBMX,RBMX/Q"
ALU_LA-Q-1                      "ALU/A-B-1,AMX/LA,BMX/RBMX,RBMX/Q"
ALU_LA.AND.K[]                  "AMX/LA,KMX/@1,BMX/KMX,ALU/AND"
ALU_LA.AND.LC                   "ALU/AND,AMX/LA,BMX/LC"
ALU_LA.ANDNOT.K[]               "AMX/LA,KMX/@1,BMX/KMX,ALU/ANDNOT"
ALU_LA.ANDNOT.MASK              "AMX/LA,BMX/MASK,ALU/ANDNOT"
ALU_LA.OR.K[]                   "ALU/OR,AMX/LA,BMX/KMX,KMX/@1"
ALU_LA.XOR.LC                   "AMX/LA,BMX/LC,ALU/XOR"
ALU_LA[]D                       "AMX/LA,RBMX/D,BMX/RBMX,ALU/@1"
ALU_LA[]LB                      "AMX/LA,BMX/LB,ALU/@1"
ALU_LA[]Q                       "AMX/LA,RBMX/Q,BMX/RBMX,ALU/@1"
ALU_LB                          "BMX/LB,ALU/B"
ALU_LC                          "BMX/LC,ALU/B"
ALU_NOT.D                       "ALU/NOTA,AMX/RAMX,RAMX/D"
ALU_NOT.K[]                     "BMX/KMX,KMX/@1,ALU/ORNOT,AMX/RAMX.OXT,DT/LONG"
ALU_NOT.RC[]                    "SPO.R/LOAD.LC,SPO.RC/@1,BMX/LC,AMX/RAMX.OXT,DT/LONG,ALU/ORNOT"
ALU_PACK.FP                     "BMX/PACKED.FL,ALU/B"
ALU_PC                          "BMX/PC,ALU/B"
ALU_Q                           "RAMX/Q,AMX/RAMX,ALU/A"
ALU_Q(B)                        "RBMX/Q,BMX/RBMX,ALU/B"
ALU_Q+K[]                       "RAMX/Q,AMX/RAMX,KMX/@1,BMX/KMX,ALU/A+B"
ALU_Q+K[]+1                     "ALU/A+B+1,AMX/RAMX,RAMX/Q,BMX/KMX,KMX/@1"
ALU_Q+LB                        "RAMX/Q,AMX/RAMX,BMX/LB,ALU/A+B"
ALU_Q+LB+1                      "RAMX/Q,AMX/RAMX,BMX/LB,ALU/A+B+1"
ALU_Q+LC                        "RAMX/Q,AMX/RAMX,BMX/LC,ALU/A+B"
ALU_Q+LC+1                      "ALU/A+B+1,AMX/RAMX,RAMX/Q,BMX/LC"
ALU_Q+LC+PSL.C                  "ALU/A+B+PSL.C,AMX/RAMX,RAMX/Q,BMX/LC"
ALU_Q+MASK                      "ALU/A+B,AMX/RAMX,RAMX/Q,BMX/MASK"
ALU_Q-D                         "RAMX/Q,AMX/RAMX,RBMX/D,BMX/RBMX,ALU/A-B"
ALU_Q-D-1                       "ALU/A-B-1,AMX/RAMX,RAMX/Q,BMX/RBMX,RBMX/D"
ALU_Q-K[]                       "RAMX/Q,AMX/RAMX,KMX/@1,BMX/KMX,ALU/A-B"
```

```
ALU_Q-LB                        "RAMX/Q,AMX/RAMX,BMX/LB,ALU/A-B"
ALU_Q-LC                        "RAMX/Q,AMX/RAMX,BMX/LC,ALU/A-B"
ALU_Q-MASK-1                    "ALU/A-B-1,AMX/RAMX,RAMX/Q,BMX/MASK"
ALU_Q.OXT[]                     "RAMX/Q,AMX/RAMX.OXT,DT/@1,ALU/A"
ALU_Q.OXT[]+D                   "ALU/A+B,AMX/RAMX.OXT,DT/@1,BMX/RBMX,RBMX/D,RAMX/Q"
ALU_Q.OXT[]+D+1                 "ALU/A+B+1,AMX/RAMX.OXT,DT/@1,BMX/RBMX,RAMX/Q,RBMX/D"
ALU_Q.OXT[]+K[]                 "ALU/A+B,AMX/RAMX.OXT,DT/@1,RAMX/Q,BMX/KMX,KMX/@2"
ALU_Q.OXT[]-D                   "ALU/A-B,RAMX/Q,AMX/RAMX.OXT,DT/@1,BMX/RBMX"
ALU_Q.OXT[]-K[]                 "ALU/A-B,AMX/RAMX.OXT,DT/@1,RAMX/Q,BMX/KMX,KMX/@2"
ALU_Q.OXT[].ANDNOT.K[]          "ALU/ANDNOT,AMX/RAMX.OXT,DT/@1,RAMX/Q,BMX/KMX,KMX/@2"
ALU_Q.OXT[].OR.K[]              "ALU/OR,AMX/RAMX.OXT,DT/@1,RAMX/Q,BMX/KMX,KMX/@2"
ALU_Q.OXT[].OR.D                "ALU/OR,AMX/RAMX.OXT,DT/@1,RAMX/Q,BMX/RBMX,RBMX/D"
ALU_Q.AND.D                     "AMX/RAMX,RAMX/Q,BMX/RBMX,RBMX/D,ALU/AND"
ALU_Q.AND.K[]                   "RAMX/Q,AMX/RAMX,KMX/@1,BMX/KMX,ALU/AND"
ALU_Q.ANDNOT.K[]                "RAMX/Q,AMX/RAMX,KMX/@1,BMX/KMX,ALU/ANDNOT"
ALU_Q.ANDNOT.MASK               "RAMX/Q,AMX/RAMX,BMX/MASK,ALU/ANDNOT"
ALU_Q.ANDNOT.R[]                "ALU/ANDNOT,AMX/RAMX,RAMX/Q,BMX/LB,SPO.R/LOAD.LAB,SPO.RAB/@1"
ALU_Q.OR.K[]                    "RAMX/Q,AMX/RAMX,KMX/@1,BMX/KMX,ALU/OR"
ALU_Q.OR.LC                     "RAMX/Q,AMX/RAMX,BMX/LC,ALU/OR"
ALU_Q.ORNOT.K[]                 "ALU/ORNOT,AMX/RAMX,RAMX/Q,BMX/KMX,KMX/@1"
ALU_Q.SXT[]                     "ALU/A,AMX/RAMX.SXT,DT/@1,RAMX/Q"
ALU_Q.SXT[]+K[]                 "RAMX/Q,AMX/RAMX.SXT,DT/@1,KMX/@2,BMX/KMX,ALU/A+B"
ALU_Q.SXT[]+LB                  "RAMX/Q,AMX/RAMX.SXT,DT/@1,BMX/LB,ALU/A+B"
ALU_Q.SXT[]+LB+1                "RAMX/Q,AMX/RAMX.SXT,DT/@1,BMX/LB,ALU/A+B+1"
ALU_Q.SXT[]+PC                  "RAMX/Q,AMX/RAMX.SXT,DT/@1,BMX/PC,ALU/A+B"
ALU_Q.SXT[].ANDNOT.K[]          "ALU/ANDNOT,AMX/RAMX.SXT,RAMX/Q,BMX/KMX,KMX/@2,DT/@1"
ALU_Q.XOR.D                     "RAMX/Q,AMX/RAMX,BMX/RBMX,RBMX/D,ALU/XOR"
ALU_Q.XOR.K[]                   "RAMX/Q,AMX/RAMX,KMX/@1,BMX/KMX,ALU/XOR"
ALU_Q.XOR.LC                    "RAMX/Q,AMX/RAMX,BMX/LC,ALU/XOR"
ALU_Q.XOR.RC[]                  "RAMX/Q,AMX/RAMX,SPO.R/LOAD.LC,SPO.RC/@1,BMX/LC,ALU/XOR"
ALU_Q[]D                        "RAMX/Q,AMX/RAMX,RBMX/D,BMX/RBMX,ALU/@1"
ALU_R(DST)                      "SPO.AC/LOAD.LAB,SPO.ACN11/DST.DST,AMX/LA,ALU/A"
ALU_R(SC).ANDNOT.K[]            "SPO.AC/LOAD.LAB,SPO.ACN/SC,AMX/LA,KMX/@1,BMX/KMX,ALU/ANDNOT"
ALU_R(SP1)+K[].RLOG             "SPO.AC/LOAD.LAB,SPO.ACN/SP1.SP1,AMX/LA,KMX/@1,BMX/KMX,ALU/A+B.RLOG"
ALU_RC(SC)                      "SPO/LOAD.LC.SC,BMX/LC,ALU/B"
ALU_RC[]                        "SPO.R/LOAD.LC,SPO.RC/@1,BMX/LC,ALU/B"
ALU_RLOG                        "BMX/0,ALU/B,MSC/READ.RLOG"
ALU_R[]                         "SPO.R/LOAD.LAB,SPO.RAB/@1,AMX/LA,ALU/A"
ALU_R[]-K[]                     "SPO.R/LOAD.LAB,SPO.RAB/@1,AMX/LA,KMX/@2,BMX/KMX,ALU/A-B"
ALU_R[].AND.K[]                 "SPO.R/LOAD.LAB,SPO.RAB/@1,AMX/LA,KMX/@2,BMX/KMX,ALU/AND"
ALU_R[].AND.LC                  "SPO.R/LOAD.LAB,SPO.RAB/@1,AMX/LA,BMX/LC,ALU/AND"
ALU_R[].ANDNOT.K[]              "SPO.R/LOAD.LAB,SPO.RAB/@1,AMX/LA,KMX/@2,BMX/KMX,ALU/ANDNOT"
ALU_R[].ANDNOT.MASK             "SPO.R/LOAD.LAB,SPO.RAB/@1,AMX/LA,BMX/MASK,ALU/ANDNOT"
ALU_R[].OR.K[]                  "SPO.R/LOAD.LAB,SPO.RAB/@1,AMX/LA,KMX/@2,BMX/KMX,ALU/OR"
ALU_R[].ORNOT.K[]               "ALU/ORNOT,AMX/LA,BMX/KMX,SPO.R/LOAD.LAB,SPO.RAB/@1,KMX/@2"
ALU_R[].XOR.K[]                 "SPO.R/LOAD.LAB,SPO.RAB/@1,AMX/LA,KMX/@2,BMX/KMX,ALU/XOR"
ALU_R[].XOR.Q                   "SPO.R/LOAD.LAB,SPO.RAB/@1,AMX/LA,RBMX/Q,BMX/RBMX,ALU/XOR"

CACHE.P_D[]                     "VAK/NOP,MCT/WRITE.P,DT/@1,DK/NOP"
CACHE[]_D                       "VAK/NOP,MCT/WRITE.V.WCHK,MSC/@1,DK/NOP"
CACHE_D(QUAD)                   "MCT/EXTWRITE.P,LONG,VAK/NOP,DK/NOP"
CACHE_D.INST.DEP                "VAK/NOP,MCT/WRITE.V.WCHK,DT/INST.DEP,DK/NOP"
CACHE_D[]                       "VAK/NOP,MCT/WRITE.V.WCHK,DT/@1,DK/NOP"
CACHE_D[].LK                    "VAK/NOP,MCT/LOCKWRITE.V.XCHK,DT/@1,DK/NOP"
CACHE_D[].NOCHK                 "VAK/NOP,MCT/WRITE.V.NOCHK,DT/@1,DK/NOP"

D&Q_D+Q                         "RAMX/D,AMX/RAMX,RBMX/Q,BMX/RBMX,ALU/A+B,SHF/ALU,DK/SHF,QK/SHF"
D&RC[]_PC                       "BMX/PC,ALU/B,SHF/ALU,DK/SHF,SPO.R/WRITE.RC,SPO.RC/@1"
D&VA_ALU                        "VAK/LOAD,SHF/ALU,DK/SHF"
D&VA_D+LC                       "RAMX/D,AMX/RAMX,BMX/LC,ALU/A+B,VAK/LOAD,SHF/ALU,DK/SHF"
D&VA_D+Q                        "D_D+Q,VAK/LOAD"
D&VA_D-K[]                      "RAMX/D,AMX/RAMX,KMX/@1,BMX/KMX,ALU/A-B,VAK/LOAD,SHF/ALU,DK/SHF"
D&VA_LA                         "AMX/LA,ALU/A,VAK/LOAD,SHF/ALU,DK/SHF"
```

```
D$VA_LB                         "BMX/LB,ALU/B,VAK/LOAD,SHF/ALU,DK/SHF"
D$VA_Q                          "RAMX/Q,AMX/RAMX,ALU/A,VAK/LOAD,DK/Q"
D$VA_Q+LB.PC                    "RAMX/Q,AMX/RAMX,BMX/PC.OR.LB,ALU/A+B,VAK/LOAD,SHF/ALU,DK/SHF"

D[]_CACHE                       "VAK/NOP,MCT/READ.V.RCHK,DT/@1,DK/NOP"
D[]_CACHE.IBCHK                 "VAK/NOP,MCT/READ.V.IBCHK,DT/@1,DK/NOP"
D[]_CACHE.LK                    "VAK/NOP,MCT/LOCKREAD.V.WCHK,DT/@1,DK/NOP"
D[]_CACHE.NOCHK                 "VAK/NOP,MCT/READ.V.NOCHK,DT/@1,DK/NOP"
D[]_CACHE.P                     "VAK/NOP,MCT/READ.P,DT/@1,DK/NOP"
D[]_CACHE.WCHK                  "VAK/NOP,MCT/READ.V.WCHK,DT/@1,DK/NOP"

D_0                             "DK/CLR"
D_0+K[]+1                       "AMX/RAMX.OXT,DT/LONG,KMX/@1,BMX/KMX,ALU/A+B+1,SHF/ALU,DK/SHF"
D_0+LC+1                        "AMX/RAMX.OXT,DT/LONG,BMX/LC,ALU/A+B+1,SHF/ALU,DK/SHF"
D_0-D                           "AMX/RAMX.OXT,DT/LONG,RBMX/D,BMX/RBMX,ALU/A-B,SHF/ALU,DK/SHF"
D_0-K[]                         "AMX/RAMX.OXT,DT/LONG,KMX/@1,BMX/KMX,ALU/A-B,SHF/ALU,DK/SHF"
D_0-Q                           "AMX/RAMX.OXT,DT/LONG,RBMX/Q,BMX/RBMX,ALU/A-B,SHF/ALU,DK/SHF"
D_0-Q-1                         "ALU_0-Q-1,D_ALU"
D_ACCEL&SYNC                    "DK/ACCEL,ACF/SYNC"
D_ALU                           "SHF/ALU,DK/SHF"
D_ALU(FRAC)                     "SHF/ALU,DK/SHF.FL"
D_ALU.LEFT                      "SHF/LEFT,DK/SHF"
D_ALU.LEFT2                     "SHF/ALU.DT,DT/LONG,DK/SHF"
D_ALU.LEFT3                     "SHF/LEFT3,DK/SHF"
D_ALU.RIGHT                     "SHF/RIGHT,DK/SHF"
D_ALU.RIGHT2                    "SHF/RIGHT2,DK/SHF"
D_BLANK                         "D_K[.20]"
D_CACHE.INST.DEP                "VAK/NOP,MCT/READ.V.IBCHK,DT/INST.DEP,DK/NOP"
D_CACHE.LK[]                    "VAK/NOP,MCT/LOCKREAD.V.WCHK,MSC/@1,DK/NOP"
D_CACHE.WCHK[]                  "VAK/NOP,MCT/READ.V.WCHK,MSC/@1,DK/NOP"
D_CACHE[]                       "VAK/NOP,MCT/READ.V.RCHK,MSC/@1,DK/NOP"
D_D(FRAC)                       "RAMX/D,AMX/RAMX,ALU/A,SHF/ALU,DK/SHF.FL"
D_D+K[]                         "RAMX/D,AMX/RAMX,KMX/@1,BMX/KMX,ALU/A+B,SHF/ALU,DK/SHF"
D_D+K[]+1                       "RAMX/D,AMX/RAMX,KMX/@1,BMX/KMX,ALU/A+B+1,SHF/ALU,DK/SHF"
D_D+LB                          "RAMX/D,AMX/RAMX,BMX/LB,ALU/A+B,SHF/ALU,DK/SHF"
D_D+LC                          "RAMX/D,AMX/RAMX,BMX/LC,ALU/A+B,SHF/ALU,DK/SHF"
D_D+LC+PSL.C                    "RAMX/D,AMX/RAMX,BMX/LC,ALU/A+B+PSL.C,SHF/ALU,DK/SHF"
D_D+Q                           "RAMX/D,AMX/RAMX,RBMX/Q,BMX/RBMX,ALU/A+B,SHF/ALU,DK/SHF"
D_D+Q+1                         "RAMX/D,AMX/RAMX,RBMX/Q,BMX/RBMX,ALU/A+B+1,SHF/ALU,DK/SHF"
D_D-K[]                         "RAMX/D,AMX/RAMX,KMX/@1,BMX/KMX,ALU/A-B,SHF/ALU,DK/SHF"
D_D-LC                          "RAMX/D,AMX/RAMX,BMX/LC,ALU/A-B,SHF/ALU,DK/SHF"
D_D-Q                           "RAMX/D,AMX/RAMX,RBMX/Q,BMX/RBMX,ALU/A-B,SHF/ALU,DK/SHF"
D_D-Q-1                         "RAMX/D,AMX/RAMX,RBMX/Q,BMX/RBMX,ALU/A-B-1,SHF/ALU,DK/SHF"
D_D.OXT[]                       "RAMX/D,AMX/RAMX.OXT,DT/@1,ALU/A,SHF/ALU,DK/SHF"
D_D.OXT[]+K[]                   "RAMX/D,AMX/RAMX.OXT,DT/@1,KMX/@2,BMX/KMX,ALU/A+B,SHF/ALU,DK/SHF"
D_D.OXT[]+Q                     "ALU/A+B,AMX/RAMX.OXT,DT/@1,BMX/RBMX,RBMX/Q,D_ALU"
D_D.OXT[]+Q+1                   "RAMX/D,AMX/RAMX.OXT,DT/@1,BMX/RBMX,ALU/A+B+1,D_ALU"
D_D.OXT[].ANDNOT.K[]            "RAMX/D,AMX/RAMX.OXT,DT/@1,KMX/@2,BMX/KMX,ALU/ANDNOT,SHF/ALU,DK/SHF"
D_D.OXT[].OR.Q                  "RAMX/D,AMX/RAMX.OXT,DT/@1,RBMX/Q,BMX/RBMX,ALU/OR,SHF/ALU,DK/SHF"
D_D.OXT[].XOR.Q                 "DK/SHF,ALU/XOR,SHF/ALU,AMX/RAMX.OXT,RAMX/D,DT/@1,RBMX/Q,BMX/RBMX"
D_D.OXT[].XOR.RC[]              "RAMX/D,AMX/RAMX.OXT,DT/@1,SPO.R/LOAD.LC,SPO.RC/@2,BMX/LC,ALU/XOR,SHF/ALU,DK/SHF"
D_D.AND.K[]                     "RAMX/D,AMX/RAMX,KMX/@1,BMX/KMX,ALU/AND,SHF/ALU,DK/SHF"
D_D.AND.K[].LEFT2               "RAMX/D,AMX/RAMX,KMX/@1,BMX/KMX,ALU/AND,SHF/ALU.DT,DT/LONG,DK/SHF"
D_D.AND.K[].RIGHT               "RAMX/D,AMX/RAMX,KMX/@1,BMX/KMX,ALU/AND,SHF/RIGHT,DK/SHF"
D_D.AND.LC                      "RAMX/D,AMX/RAMX,BMX/LC,ALU/AND,SHF/ALU,DK/SHF"
D_D.AND.MASK                    "RAMX/D,AMX/RAMX,BMX/MASK,ALU/AND,SHF/ALU,DK/SHF"
D_D.AND.Q                       "RAMX/D,AMX/RAMX,RBMX/Q,BMX/RBMX,ALU/AND,SHF/ALU,DK/SHF"
D_D.AND.RC[]                    "RAMX/D,AMX/RAMX,SPO.R/LOAD.LC,SPO.RC/@1,BMX/LC,ALU/AND,SHF/ALU,DK/SHF"
D_D.ANDNOT.K[]                  "RAMX/D,AMX/RAMX,KMX/@1,BMX/KMX,ALU/ANDNOT,SHF/ALU,DK/SHF"
D_D.ANDNOT.LC                   "RAMX/D,AMX/RAMX,BMX/LC,ALU/ANDNOT,SHF/ALU,DK/SHF"
D_D.ANDNOT.PSWZ                 "DK/SHF,ALU/ANDNOT,AMX/RAMX,RAMX/D,BMX/KMX,KMX/.4,SHF/ALU"
D_D.ANDNOT.Q                    "RAMX/D,AMX/RAMX,RBMX/Q,BMX/RBMX,ALU/ANDNOT,SHF/ALU,DK/SHF"
D_D.ANDNOT.RC[]                 "RAMX/D,AMX/RAMX,SPO.R/LOAD.LC,SPO.RC/@1,BMX/LC,ALU/ANDNOT,SHF/ALU,DK/SHF"
```

```
D_D.LEFT                    "DK/LEFT"
D_D.LEFT2                   "DK/LEFT2"
D_D.OR.ASCII                "D_D.OR.K[.30]"
D_D.OR.K[]                  "RAMX/D,AMX/RAMX,KMX/@1,BMX/KMX,ALU/OR,SHF/ALU,DK/SHF"
D_D.OR.PSWC                 "DK/SHF,ALU/OR,AMX/RAMX,RAMX/D,BMX/KMX,KMX/.1,SHF/ALU"
D_D.OR.PSWV                 "DK/SHF,ALU/OR,AMX/RAMX,RAMX/D,BMX/KMX,KMX/.2,SHF/ALU"
D_D.OR.Q                    "RAMX/D,AMX/RAMX,RBMX/Q,BMX/RBMX,ALU/OR,SHF/ALU,DK/SHF"
D_D.OR.RC[]                 "RAMX/D,AMX/RAMX,SPO.R/LOAD.LC,SPO.RC/@1,BMX/LC,ALU/OR,SHF/ALU,DK/SHF"
D_D.OR.R[]                  "ALU/OR,AMX/RAMX,RAMX/D,BMX/LB,SPO.R/LOAD.LAB,SPO.RAB/@1,DK/SHF"
D_D.ORNOT.MASK              "RAMX/D,AMX/RAMX,BMX/MASK,ALU/ORNOT,SHF/ALU,DK/SHF"
D_D.RIGHT                   "DK/RIGHT"
D_D.RIGHT(B)                "RBMX/D,BMX/RBMX,ALU/B,SHF/RIGHT,DK/SHF"
D_D.RIGHT2                  "DK/RIGHT2"
D_D.SWAP                    "DK/BYTE.SWAP"
D_D.SXT[]                   "RAMX/D,AMX/RAMX.SXT,DT/@1,ALU/A,SHF/ALU,DK/SHF"
D_D.SXT[].RIGHT             "RAMX/D,AMX/RAMX.SXT,DT/@1,ALU/A,SHF/RIGHT,DK/SHF"
D_D.XOR.K[]                 "RAMX/D,AMX/RAMX,KMX/@1,BMX/KMX,ALU/XOR,SHF/ALU,DK/SHF"
D_D.XOR.LC                  "RAMX/D,AMX/RAMX,BMX/LC,ALU/XOR,SHF/ALU,DK/SHF"
D_D.XOR.Q                   "RAMX/D,AMX/RAMX,RBMX/Q,BMX/RBMX,ALU/XOR,SHF/ALU,DK/SHF"
D_DAL.NORM                  "DK/DAL.SV"
D_DAL.SC                    "DK/DAL.SC"
D_D[]K[]                    "RAMX/D,AMX/RAMX,KMX/@2,BMX/KMX,ALU/@1,SHF/ALU,DK/SHF"
D_D[]MASK                   "RAMX/D,AMX/RAMX,BMX/MASK,ALU/@1,SHF/ALU,DK/SHF"
D_D[]Q                      "RAMX/D,AMX/RAMX,RBMX/Q,BMX/RBMX,ALU/@1,SHF/ALU,DK/SHF"
D_INT.SUM                   "MCT/READ.INT.SUM,DK/NOP"
D_K[]                       "KMX/@1,BMX/KMX,ALU/B,SHF/ALU,DK/SHF"
D_K[].RIGHT                 "KMX/@1,BMX/KMX,ALU/B,SHF/RIGHT,DK/SHF"
D_K[].RIGHT2                "KMX/@1,BMX/KMX,ALU/B,SHF/RIGHT2,DK/SHF"
D_LA                        "AMX/LA,ALU/A,SHF/ALU,DK/SHF"
D_LA(FRAC)                  "AMX/LA,ALU/A,SHF/ALU,DK/SHF.FL"
D_LA+D+PSL.C                "AMX/LA,RBMX/D,BMX/RBMX,ALU/A+B+PSL.C,SHF/ALU,DK/SHF"
D_LA-D                      "DK/SHF,ALU/A-B,AMX/LA,BMX/RBMX,RBMX/D,SHF/ALU"
D_LA-K[]                    "AMX/LA,KMX/@1,BMX/KMX,ALU/A-B,SHF/ALU,DK/SHF"
D_LA.AND.K[]                "AMX/LA,KMX/@1,BMX/KMX,ALU/AND,SHF/ALU,DK/SHF"
D_LA.RIGHT                  "AMX/LA,ALU/A,SHF/RIGHT,DK/SHF"
D_LB                        "BMX/LB,ALU/B,SHF/ALU,DK/SHF"
D_LB.PC                     "BMX/PC.OR.LB,ALU/B,SHF/ALU,DK/SHF"
D_LC                        "BMX/LC,ALU/B,SHF/ALU,DK/SHF"
D_LC(FRAC)                  "BMX/LC,ALU/B,SHF/ALU,DK/SHF.FL"
D_NOT.D                     "RAMX/D,AMX/RAMX,ALU/NOTA,SHF/ALU,DK/SHF"
D_NOT.K[]                   "KMX/@1,BMX/KMX,AMX/RAMX.OXT,DT/LONG,ALU/ORNOT,SHF/ALU,DK/SHF"
D_NOT.MASK                  "BMX/MASK,AMX/RAMX.OXT,DT/LONG,ALU/ORNOT,SHF/ALU,DK/SHF"
D_NOT.Q                     "RAMX/Q,AMX/RAMX,ALU/NOTA,SHF/ALU,DK/SHF"
D_NOT.R[]                   "LA_RA[@1],AMX/LA,ALU/NOTA,D_ALU"
D_PACK.FP                   "BMX/PACKED.FL,ALU/B,SHF/ALU,DK/SHF"
D_PACK.FP.LEFT              "BMX/PACKED.FL,ALU/B,SHF/LEFT,DK/SHF"
D_PC                        "BMX/PC,ALU/B,SHF/ALU,DK/SHF"
D_PC.LEFT                   "BMX/PC,ALU/B,SHF/LEFT,DK/SHF"
D_Q                         "DK/Q"
D_Q(FRAC)                   "RAMX/Q,AMX/RAMX,ALU/A,SHF/ALU,DK/SHF.FL"
D_Q+D                       "RAMX/Q,AMX/RAMX,RBMX/D,BMX/RBMX,ALU/A+B,SHF/ALU,DK/SHF"
D_Q+K[]                     "RAMX/Q,AMX/RAMX,KMX/@1,BMX/KMX,ALU/A+B,SHF/ALU,DK/SHF"
D_Q+LB                      "RAMX/Q,AMX/RAMX,BMX/LB,ALU/A+B,SHF/ALU,DK/SHF"
D_Q+PC                      "RAMX/Q,AMX/RAMX,BMX/PC,ALU/A+B,SHF/ALU,DK/SHF"
D_Q-D                       "RAMX/Q,AMX/RAMX,RBMX/D,BMX/RBMX,ALU/A-B,SHF/ALU,DK/SHF"
D_Q-D-1                     "RAMX/Q,AMX/RAMX,RBMX/D,BMX/RBMX,ALU/A-B-1,SHF/ALU,DK/SHF"
D_Q-K[]                     "RAMX/Q,AMX/RAMX,KMX/@1,BMX/KMX,ALU/A-B,SHF/ALU,DK/SHF"
D_Q-K[]-1                   "RAMX/Q,AMX/RAMX,KMX/@1,BMX/KMX,ALU/A-B-1,SHF/ALU,DK/SHF"
D_Q-PCSV                    "RAMX/Q,AMX/RAMX,BMX/O,MSC/READ.RLOG,ALU/A-B,SHF/ALU,DK/SHF"
D_Q.OXT[]                   "RAMX/Q,AMX/RAMX.OXT,DT/@1,ALU/A,SHF/ALU,DK/SHF"
D_Q.AND.K[]                 "RAMX/Q,AMX/RAMX,KMX/@1,BMX/KMX,ALU/AND,SHF/ALU,DK/SHF"
D_Q.AND.LC                  "RAMX/Q,AMX/RAMX,BMX/LC,ALU/AND,SHF/ALU,DK/SHF"
D_Q.AND.MASK                "RAMX/Q,AMX/RAMX,BMX/MASK,ALU/AND,SHF/ALU,DK/SHF"
```

```
D_Q.AND.RC[]                    "RAMX/Q,AMX/RAMX,SPO.R/LOAD.LC,SPO.RC/@1,BMX/LC,ALU/AND,SHF/ALU,DK/SHF"
D_Q.ANDNOT.D                    "RAMX/Q,AMX/RAMX,RBMX/D,BMX/RBMX,ALU/ANDNOT,SHF/ALU,DK/SHF"
D_Q.ANDNOT.K[]                  "RAMX/Q,AMX/RAMX,KMX/@1,BMX/KMX,ALU/ANDNOT,SHF/ALU,DK/SHF"
D_Q.ANDNOT.MASK                 "RAMX/Q,AMX/RAMX,BMX/MASK,ALU/ANDNOT,SHF/ALU,DK/SHF"
D_Q.ANDNOT.PSWC                 "DK/SHF,ALU/ANDNOT,AMX/RAMX,RAMX/Q,BMX/KMX,KMX/.1,SHF/ALU"
D_Q.ANDNOT.PSWN                 "DK/SHF,ALU/ANDNOT,AMX/RAMX,RAMX/Q,BMX/KMX,KMX/.8,SHF/ALU"
D_Q.ANDNOT.PSWZ                 "DK/SHF,ALU/ANDNOT,AMX/RAMX,RAMX/Q,BMX/KMX,KMX/.4,SHF/ALU"
D_Q.LEFT                        "RAMX/Q,AMX/RAMX,ALU/A,SHF/LEFT,DK/SHF"
D_Q.OR.K[]                      "RAMX/Q,AMX/RAMX,KMX/@1,BMX/KMX,ALU/OR,SHF/ALU,DK/SHF"
D_Q.OR.PSWC                     "DK/SHF,ALU/OR,AMX/RAMX,RAMX/Q,BMX/KMX,KMX/.1,SHF/ALU"
D_Q.OR.RC[]                     "RAMX/Q,AMX/RAMX,SPO.R/LOAD.LC,SPO.RC/@1,BMX/LC,ALU/OR,SHF/ALU,DK/SHF"
D_Q.ORNOT.MASK                  "RAMX/Q,AMX/RAMX,BMX/MASK,ALU/ORNOT,SHF/ALU,DK/SHF"
D_Q.RIGHT                       "RAMX/Q,AMX/RAMX,ALU/A,SHF/RIGHT,DK/SHF"
D_Q.RIGHT2                      "RAMX/Q,AMX/RAMX,ALU/A,SHF/RIGHT2,DK/SHF"
D_Q.SXT[]                       "RAMX/Q,AMX/RAMX.SXT,DT/@1,ALU/A,SHF/ALU,DK/SHF"
D_Q.XOR.RC[]                    "RAMX/Q,AMX/RAMX,SPO.R/LOAD.LC,SPO.RC/@1,BMX/LC,ALU/XOR,SHF/ALU,DK/SHF"
D_Q[]D                          "RAMX/Q,AMX/RAMX,RBMX/D,BMX/RBMX,ALU/@1,SHF/ALU,DK/SHF"
D_Q[]K[]                        "ALU/@1,SHF/ALU,DK/SHF,BMX/KMX,KMX/@2,AMX/RAMX,RAMX/Q"
D_Q[]MASK                       "RAMX/Q,AMX/RAMX,BMX/MASK,ALU/@1,SHF/ALU,DK/SHF"
D_R(PRN+1)                      "SPO.AC/LOAD.LAB,SPO.ACN/PRN+1,AMX/LA,ALU/A,SHF/ALU,DK/SHF"
D_R(SC)                         "SPO.AC/LOAD.LAB,SPO.ACN/SC,AMX/LA,ALU/A,SHF/ALU,DK/SHF"
D_R(SP1+1)                      "SPO.AC/LOAD.LAB,SPO.ACN/SP1+1,AMX/LA,ALU/A,SHF/ALU,DK/SHF"
D_RC(SC)                        "SPO/LOAD.LC.SC,BMX/LC,ALU/B,SHF/ALU,DK/SHF"
D_RC[]                          "SPO.R/LOAD.LC,SPO.RC/@1,BMX/LC,ALU/B,SHF/ALU,DK/SHF"
D_RLOG                          "BMX/0,MSC/READ.RLOG,ALU/B,SHF/ALU,DK/SHF"
D_RLOG.RIGHT                    "BMX/0,MSC/READ.RLOG,ALU/B,SHF/RIGHT,DK/SHF"
D_R[]                           "SPO.R/LOAD.LAB,SPO.RAB/@1,AMX/LA,ALU/A,SHF/ALU,DK/SHF"
D_R[](FRAC)                     "SPO.R/LOAD.LAB,SPO.RAB/@1,AMX/LA,ALU/A,SHF/ALU,DK/SHF.FL"
D_R[].AND.K[]                   "SPO.R/LOAD.LAB,SPO.RAB/@1,AMX/LA,KMX/@2,BMX/KMX,ALU/AND,SHF/ALU,DK/SHF"
D_R[].OR.K[]                    "SPO.R/LOAD.LAB,SPO.RAB/@1,AMX/LA,KMX/@2,BMX/KMX,ALU/OR,SHF/ALU,DK/SHF"
D_R[].ORNOT.K[]                 "LAB_R[@1],AMX/LA,BMX/KMX,KMX/@2,ALU/ORNOT,D_ALU"

EALU_D(EXP)                     "RAMX/D,AMX/RAMX,EBMX/AMX.EXP,EALU/B"
EALU_FE                         "EBMX/FE,EALU/B"
EALU_K[]                        "KMX/@1,EBMX/KMX,EALU/B"
EALU_R[](EXP)                   "SPO.R/LOAD.LAB,SPO.RAB/@1,AMX/LA,EBMX/AMX.EXP,EALU/B"
EALU_SC                         "EALU/A"
EALU_SC+FE                      "EBMX/FE,EALU/A+B"
EALU_SC+K[]                     "KMX/@1,EBMX/KMX,EALU/A+B"
EALU_SC-FE                      "EBMX/FE,EALU/A-B"
EALU_SC-K[]                     "KMX/@1,EBMX/KMX,EALU/A-B"
EALU_SC.ANDNOT.K[]              "KMX/@1,EBMX/KMX,EALU/ANDNOT"
EALU_STATE                      "EALU/A,MSC/LOAD.STATE"

FE&SC_K[]                       "KMX/@1,EBMX/KMX,EALU/B,FEK/LOAD,SMX/EALU,SCK/LOAD"
FE_0(A)                         "AMX/RAMX.OXT,DT/LONG,EBMX/AMX.EXP,EALU/B,FEK/LOAD"
FE_D(EXP)                       "RAMX/D,AMX/RAMX,EBMX/AMX.EXP,EALU/B,FEK/LOAD"
FE_EALU                         "FEK/LOAD"
FE_K[]                          "KMX/@1,EBMX/KMX,EALU/B,FEK/LOAD"
FE_LA(EXP)                      "AMX/LA,EBMX/AMX.EXP,EALU/B,FEK/LOAD"
FE_NABS(SC-FE)                  "EALU/NABS.A-B,EBMX/FE,FEK/LOAD"
FE_NABS(SC-LA(EXP))             "AMX/LA,EBMX/AMX.EXP,EALU/NABS.A-B,FEK/LOAD"
FE_Q(EXP)                       "RAMX/Q,AMX/RAMX,EBMX/AMX.EXP,EALU/B,FEK/LOAD"
FE_R[](EXP)                     "SPO.R/LOAD.LAB,SPO.RAB/@1,AMX/LA,EBMX/AMX.EXP,EALU/B,FEK/LOAD"
FE_SC                           "EALU/A,FEK/LOAD"
FE_SC+1                         "EALU/A+1,FEK/LOAD"
FE_SC+FE                        "EBMX/FE,EALU/A+B,FEK/LOAD"
FE_SC+K[]                       "KMX/@1,EBMX/KMX,EALU/A+B,FEK/LOAD"
FE_SC+LA(EXP)                   "AMX/LA,EBMX/AMX.EXP,EALU/A+B,FEK/LOAD"
FE_SC-FE                        "EBMX/FE,EALU/A-B,FEK/LOAD"
FE_SC-K[]                       "KMX/@1,EBMX/KMX,EALU/A-B,FEK/LOAD"
FE_SC-LA(EXP)                   "AMX/LA,EBMX/AMX.EXP,EALU/A-B,FEK/LOAD"
FE_SC-SHF.VAL                   "EBMX/SHF.VAL,EALU/A-B,FEK/LOAD"
```

```
FE_SC.ANDNOT.FE                 "EBMX/FE,EALU/ANDNOT,FEK/LOAD"
FE_SC.ANDNOT.K[]                "KMX/@1,EBMX/KMX,EALU/ANDNOT,FEK/LOAD"
FE_SC.OR.K[]                    "EALU/OR,EBMX/KMX,KMX/@1,FEK/LOAD"
FE_SHF.VAL                      "EBMX/SHF.VAL,EALU/B,FEK/LOAD"
FE_STATE                        "MSC/LOAD.STATE,EALU/A,FEK/LOAD"

ID(SC)_D                        "CID/WRITE.SC"
ID[]_D                          "CID/WRITE.KMX,ID.ADDR/@1"
ID_D&NO.SYNC                    "CID/WRITE.KMX,ADS/IBA,KMX/SP1.CON"
ID_D.SYNC                       "CID/WRITE.KMX,ADS/IBA,KMX/SP1.CON,ACF/SYNC"

K[]                             "KMX/@1"

LAB_R(DST)                      "SPO.AC/LOAD.LAB,SPO.ACN11/DST.DST"
LAB_R(PRN)                      "SPO.AC/LOAD.LAB,SPO.ACN/PRN"
LAB_R(PRN+1)                    "SPO.AC/LOAD.LAB,SPO.ACN/PRN+1"
LAB_R(SC)                       "SPO.AC/LOAD.LAB,SPO.ACN/SC"
LAB_R(SP1)                      "SPO.AC/LOAD.LAB,SPO.ACN/SP1.SP1"
LAB_R(SP1+1)                    "SPO.AC/LOAD.LAB,SPO.ACN/SP1+1"
LAB_R1&RC[]_0                   "ALU_0(A),LAB_R1&RC[@1]_ALU"
LAB_R1&RC[]_0+LC+1              "ALU/A+B+1,AMX/RAMX.OXT,DT/LONG,BMX/LC,SPO.R/LOAD.LAB1.WRITE.RC,SPO.RC/@1,SHF/ALU"
LAB_R1&RC[]_0-D                 "SPO.R/LOAD.LAB1.WRITE.RC,SPO.RC/@1,ALU/A-B,AMX/RAMX.OXT,DT/LONG,BMX/RBMX,RBMX/D,SHF/ALU"
LAB_R1&RC[]_ALU                 "SPO.R/LOAD.LAB1.WRITE.RC,SPO.RC/@1,SHF/ALU"
LAB_R1&RC[]_ALU.RIGHT2          "SPO.R/LOAD.LAB1.WRITE.RC,SPO.RC/@1,SHF/RIGHT2"
LAB_R1&RC[]_D+LC                "ALU_D+LC,LAB_R1&RC[@1]_ALU"
LAB_R1&RC[]_D.OXT[]+K[]         "ALU_D.OXT[@2]+K[@3],LAB_R1&RC[@1]_ALU"
LAB_R1&RC[]_Q-K[]               "ALU_Q-K[@2],LAB_R1&RC[@1]_ALU"
LAB_RC[]                        "SPO.R/LOAD.LAB,SPO.RAB/@1"

LA_R(DST)&LB_R(SRC)             "SPO.AC/LOAD.LAB,SPO.ACN11/DST.SRC"
LA_R(SP2)&LB_R(SP1)             "SPO.AC/LOAD.LAB,SPO.ACN/SP2.SP1"
LA_RA[]                         "SPO.AC/LOAD.LA,SPO.RAB/@1"
LC_RC(SC)                       "SPO/LOAD.LC.SC"
LC_RC[]                         "SPO.R/LOAD.LC,SPO.RC/@1"
LC_RC[]&R1_(LA+LB).LEFT         "AMX/LA,BMX/LB,ALU/A+B,SHF/LEFT,SPO.R/LOAD.LC.WRITE.RAB1,SPO.RC/@1"
LC_RC[]&R1_(LA+LB+PSL.C).LEFT   "AMX/LA,BMX/LB,ALU/A+B+PSL.C,SHF/LEFT,SPO.R/LOAD.LC.WRITE.RAB1,SPO.RC/@1"
LC_RC[]&R1_(LA+LB.RLOG).LEFT    "AMX/LA,BMX/LB,ALU/A+B.RLOG,SHF/LEFT,SPO.R/LOAD.LC.WRITE.RAB1,SPO.RC/@1"
LC_RC[]&R1_(LA-LB).LEFT         "AMX/LA,BMX/LB,ALU/A-B,SHF/LEFT,SPO.R/LOAD.LC.WRITE.RAB1,SPO.RC/@1"
LC_RC[]&R1_(LA-LB.RLOG).LEFT    "AMX/LA,BMX/LB,ALU/A-B.RLOG,SHF/LEFT,SPO.R/LOAD.LC.WRITE.RAB1,SPO.RC/@1"
LC_RC[]&R1_ALU                  "SPO.R/LOAD.LC.WRITE.RAB1,SPO.RC/@1,SHF/ALU"
LC_RC[]&R1_D                    "ALU_D,LC_RC[@1]&R1_ALU"
LC_RC[]&R1_LA+K[]               "SPO.R/LOAD.LC.WRITE.RAB1,SPO.RC/@1,SHF/ALU,ALU/A+B,AMX/LA,BMX/KMX,KMX/@2"
LC_RC[]&R1_LA-K[]               "ALU_LA-K[@2],LC_RC[@1]&R1_ALU"
LC_RC[]&R1_LB                   "ALU_LB,LC_RC[@1]&R1_ALU"
LC_RC[]&R1_Q                    "SPO.R/LOAD.LC.WRITE.RAB1,SPO.RC/@1,SHF/ALU,ALU/A,AMX/RAMX,RAMX/Q"

N&Z_ALU                         "CCK/NZ_ALU.VC_VC"
N&Z_ALU.V&C_0                   "CCK/NZ_ALU.VC_0"
N_AMX.Z_TST                     "CCK/N_AMX.Z_TST.VC_VC"

PC&VA_ALU                       "VAK/LOAD,PCK/PC_VA"
PC&VA_D                         "RAMX/D,AMX/RAMX,ALU/A,VAK/LOAD,PCK/PC_VA"
PC&VA_D+K[]                     "RAMX/D,AMX/RAMX,KMX/@1,BMX/KMX,ALU/A+B,VAK/LOAD,PCK/PC_VA"
PC&VA_D-K[]                     "RAMX/D,AMX/RAMX,KMX/@1,BMX/KMX,ALU/A-B,VAK/LOAD,PCK/PC_VA"
PC&VA_D-PC                      "RAMX/D,AMX/RAMX,BMX/PC,ALU/A-B,VAK/LOAD,PCK/PC_VA"
PC&VA_D.OXT[]                   "RAMX/D,AMX/RAMX.OXT,DT/@1,ALU/A,VAK/LOAD,PCK/PC_VA"
PC&VA_D.OXT[]+PC                "RAMX/D,AMX/RAMX.OXT,DT/@1,BMX/PC,ALU/A+B,VAK/LOAD,PCK/PC_VA"
PC&VA_D.SXT[]+PC                "RAMX/D,AMX/RAMX.SXT,DT/@1,BMX/PC,ALU/A+B,VAK/LOAD,PCK/PC_VA"
PC&VA_K[]                       "KMX/@1,BMX/KMX,ALU/B,VAK/LOAD,PCK/PC_VA"
PC&VA_PC                        "BMX/PC,ALU/B,VAK/LOAD,PCK/PC_VA"
PC&VA_Q                         "RAMX/Q,AMX/RAMX,ALU/A,VAK/LOAD,PCK/PC_VA"
PC&VA_Q+PC                      "RAMX/Q,AMX/RAMX,BMX/PC,ALU/A+B,VAK/LOAD,PCK/PC_VA"
PC&VA_Q-D                       "RAMX/Q,AMX/RAMX,RBMX/D,BMX/RBMX,ALU/A-B,VAK/LOAD,PCK/PC_VA"
```

```
PC&VA_Q-K[]                     "RAMX/Q,AMX/RAMX,KMX/@1,BMX/KMX,ALU/A-B,VAK/LOAD,PCK/PC_VA"
PC&VA_Q.SXT[]+PC                "RAMX/Q,AMX/RAMX.SXT,DT/@1,BMX/PC,ALU/A+B,VAK/LOAD,PCK/PC_VA"
PC&VA_RC[]                      "SPO.R/LOAD.LC,SPO.RC/@1,BMX/LC,ALU/B,VAK/LOAD,PCK/PC_VA"
PC&VA_R[].ANDNOT.K[]            "SPO.R/LOAD.LAE,SPO.RAB/@1,AMX/LA,KMX/@2,BMX/KMX,ALU/ANDNOT,VAK/LOAD,PCK/PC_VA"

PC_PC+1                         "PCK/PC+1"
PC_PC+2                         "PCK/PC+2"
PC_PC+4                         "PCK/PC+4"
PC_PC+N                         "PCK/PC+N"
PC_Q+PC                         "ALU/A+B,VAK/LOAD,PCK/PC_VA,BMX/PC,AMX/RAMX,RAMX/Q"
PC_VA                           "PCK/PC_VA"
PC_VIBA                         "PCK/PC_IBA"
PSL<C>_AMX0                     "CCK/C_AMX0"

Q&VA_ALU                        "VAK/LOAD,SHF/ALU,QK/SHF"
Q&VA_D                          "RAMX/D,AMX/RAMX,ALU/A,VAK/LOAD,SHF/ALU,QK/SHF"
Q&VA_D+LC                       "RAMX/D,AMX/RAMX,BMX/LC,ALU/A+B,VAK/LOAD,SHF/ALU,QK/SHF"
Q&VA_LA                         "AMX/LA,ALU/A,VAK/LOAD,SHF/ALU,QK/SHF"
Q&VA_Q+LB.PC                    "RAMX/Q,AMX/RAMX,BMX/PC.OR.LB,ALU/A+B,VAK/LOAD,SHF/ALU,QK/SHF"

QD_(Q+LB)D.RIGHT2               "ALU_Q+LB,Q_ALU.RIGHT2,D_D.RIGHT2"
QD_(Q+LC)D.RIGHT2               "ALU_Q+LC,Q_ALU.RIGHT2,D_D.RIGHT2"
QD_(Q-LB)D.RIGHT2               "ALU_Q-LB,Q_ALU.RIGHT2,D_D.RIGHT2"
QD_(Q-LC)D.RIGHT2               "ALU_Q-LC,Q_ALU.RIGHT2,D_D.RIGHT2"
QD_QD.RIGHT2                    "ALU_Q,Q_ALU.RIGHT2,D_D.RIGHT2"

Q_(LA+Q).RIGHT                  "AMX/LA,RBMX/Q,BMX/RBMX,ALU/A+B,SHF/RIGHT,QK/SHF"
Q_(Q+LB).RIGHT                  "RAMX/Q,AMX/RAMX,BMX/LB,ALU/A+B,SHF/RIGHT,QK/SHF"
Q_0                             "QK/CLR"
Q_0+LC+1                        "ALU/A+B+1,AMX/RAMX.OXT,DT/LONG,SHF/ALU,QK/SHF,BMX/LC"
Q_0+MASK+1                      "AMX/RAMX.OXT,DT/LONG,BMX/MASK,ALU/A+B+1,SHF/ALU,QK/SHF"
Q_0+PC.RLOG                     "AMX/RAMX.OXT,DT/LONG,BMX/PC,ALU/A+B.RLOG,SHF/ALU,QK/SHF"
Q_0-D                           "AMX/RAMX.OXT,DT/LONG,RBMX/D,BMX/RBMX,ALU/A-B,SHF/ALU,QK/SHF"
Q_0-K[]                         "AMX/RAMX.OXT,DT/LONG,KMX/@1,BMX/KMX,ALU/A-B,SHF/ALU,QK/SHF"
Q_0-LC                          "AMX/RAMX.OXT,DT/LONG,BMX/LC,ALU/A-B,SHF/ALU,QK/SHF"
Q_0-Q                           "AMX/RAMX.OXT,DT/LONG,RBMX/Q,BMX/RBMX,ALU/A-B,SHF/ALU,QK/SHF"
Q_ACCEL&SYNC                    "QK/ACCEL,ACF/SYNC"
Q_ALU                           "SHF/ALU,QK/SHF"
Q_ALU(FRAC)                     "SHF/ALU,QK/SHF.FL"
Q_ALU.LEFT                      "SHF/LEFT,QK/SHF"
Q_ALU.LEFT2                     "SHF/ALU.DT,DT/LONG,QK/SHF"
Q_ALU.LEFT3                     "QK/SHF,SHF/LEFT3"
Q_ALU.RIGHT                     "SHF/RIGHT,QK/SHF"
Q_ALU.RIGHT2                    "SHF/RIGHT2,QK/SHF"
Q_D                             "QK/D"
Q_D(FRAC)(B)                    "RBMX/D,BMX/RBMX,ALU/B,SHF/ALU,QK/SHF.FL"
Q_D+K[]                         "RAMX/D,AMX/RAMX,KMX/@1,BMX/KMX,ALU/A+B,SHF/ALU,QK/SHF"
Q_D+K[]+1                       "RAMX/D,AMX/RAMX,KMX/@1,BMX/KMX,ALU/A+B+1,SHF/ALU,QK/SHF"
Q_D+K[].LEFT                    "RAMX/D,AMX/RAMX,KMX/@1,BMX/KMX,ALU/A+B,SHF/LEFT,QK/SHF"
Q_D+LC                          "RAMX/D,AMX/RAMX,BMX/LC,ALU/A+B,SHF/ALU,QK/SHF"
Q_D-K[]                         "RAMX/D,AMX/RAMX,KMX/@1,BMX/KMX,ALU/A-B,SHF/ALU,QK/SHF"
Q_D-LC                          "RAMX/D,AMX/RAMX,BMX/LC,ALU/A-B,SHF/ALU,QK/SHF"
Q_D-Q                           "RAMX/D,AMX/RAMX,RBMX/Q,BMX/RBMX,ALU/A-B,SHF/ALU,QK/SHF"
Q_D.OXT[]                       "RAMX/D,AMX/RAMX.OXT,DT/@1,ALU/A,SHF/ALU,QK/SHF"
Q_D.OXT[]+K[].LEFT              "RAMX/D,AMX/RAMX.OXT,DT/@1,KMX/@2,BMX/KMX,ALU/A+B,SHF/LEFT,QK/SHF"
Q_D.OXT[].OR.PACK.FP            "RAMX/D,AMX/RAMX.OXT,DT/@1,BMX/PACKED.FL,ALU/OR,QK/SHF"
Q_D.AND.K[]                     "RAMX/D,AMX/RAMX,KMX/@1,BMX/KMX,ALU/AND,SHF/ALU,QK/SHF"
Q_D.AND.K[].RIGHT               "RAMX/D,AMX/RAMX,KMX/@1,BMX/KMX,ALU/AND,SHF/RIGHT,QK/SHF"
Q_D.AND.K[].RIGHT2              "RAMX/D,AMX/RAMX,KMX/@1,BMX/KMX,ALU/AND,SHF/RIGHT2,QK/SHF"
Q_D.AND.RC[]                    "RAMX/D,AMX/RAMX,SPO.R/LOAD.LC,SPO.RC/@1,BMX/LC,ALU/AND,SHF/ALU,QK/SHF"
Q_D.ANDNOT.RC[]                 "RAMX/D,AMX/RAMX,SPO.R/LOAD.LC,SPO.RC/@1,BMX/LC,ALU/ANDNOT,SHF/ALU,QK/SHF"
Q_D.LEFT3                       "RAMX/D,AMX/RAMX,ALU/A,SHF/LEFT3,QK/SHF"
Q_D.OR.K[]                      "RAMX/D,AMX/RAMX,KMX/@1,BMX/KMX,ALU/OR,SHF/ALU,QK/SHF"
```

```
Q_D.OR.RC[]                     "RAMX/D,AMX/RAMX,SPO.R/LOAD.LC,SPO.RC/@1,BMX/LC,ALU/OR,SHF/ALU,QK/SHF"
Q_D.RIGHT                       "RAMX/D,AMX/RAMX,ALU/A,SHF/RIGHT,QK/SHF"
Q_D.RIGHT2                      "RAMX/D,AMX/RAMX,ALU/A,SHF/RIGHT2,QK/SHF"
Q_D.SXT[]                       "RAMX/D,AMX/RAMX.SXT,DT/@1,ALU/A,SHF/ALU,QK/SHF"
Q_D.XOR.Q                       "QK/SHF,ALU/XOR,AMX/RAMX,RAMX/D,BMX/RBMX,RBMX/Q,SHF/ALU"
Q_DEC.CON                       "QK/DEC.CON"
Q_IB.BDEST                      "IBC/BDEST,QK/ID,MCT/ALLOW.IB.READ"
Q_IB.DATA                       "QK/ID,MCT/ALLOW.IB.READ"
Q_ID(SC)                        "CID/READ.SC,QK/ID"
Q_ID[]                          "CID/READ.KMX,ID.ADDR/@1,QK/ID"
Q_K[]                           "KMX/@1,BMX/KMX,ALU/B,SHF/ALU,QK/SHF"
Q_K[]+1                         "AMX/RAMX.OXT,DT/LONG,KMX/@1,BMX/KMX,ALU/A+B+1,SHF/ALU,QK/SHF"
Q_K[].CTX                       "KMX/@1,BMX/KMX,ALU/B,SHF/ALU.DT,DT/INST.DEP,QK/SHF"
Q_K[].RIGHT                     "KMX/@1,BMX/KMX,ALU/B,SHF/RIGHT,QK/SHF"
Q_K[].RIGHT2                    "KMX/@1,BMX/KMX,ALU/B,SHF/RIGHT2,QK/SHF"
Q_LA                            "AMX/LA,ALU/A,SHF/ALU,QK/SHF"
Q_LA+K[]                        "AMX/LA,KMX/@1,BMX/KMX,ALU/A+B,SHF/ALU,QK/SHF"
Q_LA+Q                          "AMX/LA,RBMX/Q,BMX/RBMX,ALU/A+B,SHF/ALU,QK/SHF"
Q_LA-K[]                        "AMX/LA,KMX/@1,BMX/KMX,ALU/A-B,SHF/ALU,QK/SHF"
Q_LA.AND.K[]                    "AMX/LA,KMX/@1,BMX/KMX,ALU/AND,SHF/ALU,QK/SHF"
Q_LA.ANDNOT.RC[]                "AMX/LA,SPO.R/LOAD.LC,SPO.RC/@1,BMX/LC,ALU/ANDNOT,SHF/ALU,QK/SHF"
Q_LB                            "BMX/LB,ALU/B,SHF/ALU,QK/SHF"
Q_LC                            "BMX/LC,ALU/B,SHF/ALU,QK/SHF"
Q_NOT.Q                         "RAMX/Q,AMX/RAMX,ALU/NOTA,SHF/ALU,QK/SHF"
Q_NOT.R[]                       "LA_RA[@1],AMX/LA,ALU/NOTA,Q_ALU"
Q_PACK.FP                       "BMX/PACKED.FL,ALU/B,SHF/ALU,QK/SHF"
Q_PC                            "BMX/PC,ALU/B,SHF/ALU,QK/SHF"
Q_Q(FRAC)                       "RAMX/Q,AMX/RAMX,ALU/A,SHF/ALU,QK/SHF.FL"
Q_Q(FRAC)(B)                    "RBMX/Q,BMX/RBMX,ALU/B,SHF/ALU,QK/SHF.FL"
Q_Q+D                           "RAMX/Q,AMX/RAMX,RBMX/D,BMX/RBMX,ALU/A+B,SHF/ALU,QK/SHF"
Q_Q+K[]                         "RAMX/Q,AMX/RAMX,KMX/@1,BMX/KMX,ALU/A+B,SHF/ALU,QK/SHF"
Q_Q+K[]+1                       "RAMX/Q,AMX/RAMX,KMX/@1,BMX/KMX,ALU/A+B+1,SHF/ALU,QK/SHF"
Q_Q+LC                          "RAMX/Q,AMX/RAMX,BMX/LC,ALU/A+B,SHF/ALU,QK/SHF"
Q_Q+PC                          "RAMX/Q,AMX/RAMX,BMX/PC,ALU/A+B,SHF/ALU,QK/SHF"
Q_Q-D                           "RAMX/Q,AMX/RAMX,RBMX/D,BMX/RBMX,ALU/A-B,SHF/ALU,QK/SHF"
Q_Q-D-1                         "RAMX/Q,AMX/RAMX,RBMX/D,BMX/RBMX,ALU/A-B-1,SHF/ALU,QK/SHF"
Q_Q-K[]                         "RAMX/Q,AMX/RAMX,KMX/@1,BMX/KMX,ALU/A-B,SHF/ALU,QK/SHF"
Q_Q-K[]-1                       "RAMX/Q,AMX/RAMX,KMX/@1,BMX/KMX,ALU/A-B-1,SHF/ALU,QK/SHF"
Q_Q-LC                          "RAMX/Q,AMX/RAMX,BMX/LC,ALU/A-B,SHF/ALU,QK/SHF"
Q_Q-LC-1                        "RAMX/Q,AMX/RAMX,BMX/LC,ALU/A-B-1,SHF/ALU,QK/SHF"
Q_Q-MASK-1                      "RAMX/Q,AMX/RAMX,BMX/MASK,ALU/A-B-1,SHF/ALU,QK/SHF"
Q_Q.OXT[]-K[]                   "RAMX/Q,AMX/RAMX.OXT,DT/@1,KMX/@2,BMX/KMX,ALU/A-B,SHF/ALU,QK/SHF"
Q_Q.OXT[].LEFT                  "RAMX/Q,AMX/RAMX.OXT,DT/@1,ALU/A,SHF/LEFT,QK/SHF"
Q_Q.OXT[].OR.D                  "RAMX/Q,AMX/RAMX.OXT,DT/@1,RBMX/D,BMX/RBMX,ALU/OR,SHF/ALU,QK/SHF"
Q_Q.AND.K[]                     "RAMX/Q,AMX/RAMX,KMX/@1,BMX/KMX,ALU/AND,SHF/ALU,QK/SHF"
Q_Q.AND.K[].RIGHT2              "RAMX/Q,AMX/RAMX,KMX/@1,BMX/KMX,ALU/AND,SHF/RIGHT2,QK/SHF"
Q_Q.AND.K[].RIGHT               "RAMX/Q,AMX/RAMX,KMX/@1,BMX/KMX,ALU/AND,SHF/RIGHT,QK/SHF"
Q_Q.AND.R[]                     "RAMX/Q,AMX/RAMX,SPO.R/LOAD.LAB,SPO.RAB/@1,BMX/LB,ALU/AND,SHF/ALU,QK/SHF"
Q_Q.AND.RC[]                    "RAMX/Q,AMX/RAMX,SPO.R/LOAD.LC,SPO.RC/@1,BMX/LC,ALU/AND,SHF/ALU,QK/SHF"
Q_Q.ANDNOT.D                    "RAMX/Q,AMX/RAMX,RBMX/D,BMX/RBMX,ALU/ANDNOT,SHF/ALU,QK/SHF"
Q_Q.ANDNOT.K[]                  "RAMX/Q,AMX/RAMX,KMX/@1,BMX/KMX,ALU/ANDNOT,SHF/ALU,QK/SHF"
Q_Q.ANDNOT.RC[]                 "RAMX/Q,AMX/RAMX,SPO.R/LOAD.LC,SPO.RC/@1,BMX/LC,ALU/ANDNOT,SHF/ALU,QK/SHF"
Q_Q.LEFT                        "QK/LEFT"
Q_Q.LEFT2                       "QK/LEFT2"
Q_Q.OR.K[]                      "RAMX/Q,AMX/RAMX,KMX/@1,BMX/KMX,ALU/OR,SHF/ALU,QK/SHF"
Q_Q.ORNOT.MASK                  "RAMX/Q,AMX/RAMX,BMX/MASK,ALU/ORNOT,SHF/ALU,QK/SHF"
Q_Q.RIGHT                       "QK/RIGHT"
Q_Q.RIGHT2                      "QK/RIGHT2"
Q_Q.SXT[]                       "RAMX/Q,AMX/RAMX.SXT,DT/@1,ALU/A,SHF/ALU,QK/SHF"
Q_Q.XOR.K[]                     "RAMX/Q,AMX/RAMX,KMX/@1,BMX/KMX,ALU/XOR,SHF/ALU,QK/SHF"
Q_R(PRN).ANDNOT.Q               "SPO.AC/LOAD.LAB,SPO.ACN/PRN,AMX/LA,RBMX/Q,BMX/RBMX,ALU/ANDNOT,SHF/ALU,QK/SHF"
Q_R(PRN+1)                      "SPO.AC/LOAD.LAB,SPO.ACN/PRN+1,AMX/LA,ALU/A,SHF/ALU,QK/SHF"
Q_R(PRN+1).AND.Q                "SPO.AC/LOAD.LAB,SPO.ACN/PRN+1,AMX/LA,RBMX/Q,BMX/RBMX,ALU/AND,SHF/ALU,QK/SHF"
```

```
Q_R(SC)                     "ALU/A,SHF/ALU,AMX/LA,SPO.AC/LOAD.LAB,SPO.ACN/SC,QK/SHF"
Q_R(SRC!1).AND.K[]          "SPO.AC/LOAD.LAB,SPO.ACN11/SRC.OR.1,AMX/LA,KMX/@1,BMX/KMX,ALU/AND,SHF/ALU,QK/SHF"
Q_RC(SC)                    "ALU/B,SHF/ALU,BMX/LC,SPO/LOAD.LC.SC,QK/SHF"
Q_RC[]                      "SPO.R/LOAD.LC,SPO.RC/@1,BMX/LC,ALU/B,SHF/ALU,QK/SHF"
Q_RC[](FRAC)                "SPO.R/LOAD.LC,SPO.RC/@1,BMX/LC,ALU/B,SHF/ALU,QK/SHF.FL"
Q_R[]                       "SPO.R/LOAD.LAB,SPO.RAB/@1,AMX/LA,ALU/A,SHF/ALU,QK/SHF"
Q_R[](FRAC)                 "SPO.R/LOAD.LAB,SPO.RAB/@1,AMX/LA,ALU/A,SHF/ALU,QK/SHF.FL"
Q_R[].AND.K[]               "SPO.R/LOAD.LAB,SPO.RAB/@1,AMX/LA,KMX/@2,BMX/KMX,ALU/AND,SHF/ALU,QK/SHF"
Q_R[].AND.K[].RIGHT         "SPO.R/LOAD.LAB,SPO.RAB/@1,AMX/LA,ALU/AND,BMX/KMX,KMX/@2,SHF/RIGHT,QK/SHF"
Q_R[].ANDNOT.K[]            "SPO.R/LOAD.LAB,SPO.RAB/@1,AMX/LA,KMX/@2,BMX/KMX,ALU/ANDNOT,SHF/ALU,QK/SHF"
Q_R[].OR.K[]                "ALU/OR,AMX/LA,SPO.R/LOAD.LAB,SPO.RAB/@1,BMX/KMX,KMX/@2,QK/SHF"
Q_SC                        "ALU/B,BMX/KMX,KMX/SC,SHF/ALU,QK/SHF"
Q_SHF                       "QK/SHF"

R(DST)_ALU                  "SHF/ALU,SPO.AC/WRITE.RAB,SPO.ACN11/DST.DST"
R(DST)_D                    "RAMX/D,AMX/RAMX,ALU/A,SHF/ALU,SPO.AC/WRITE.RAB,SPO.ACN11/DST.DST"
R(DST)_D.SXT[].RIGHT        "RAMX/D,AMX/RAMX.SXT,DT/@1,ALU/A,SHF/RIGHT,SPO.AC/WRITE.RAB,SPO.ACN11/DST.DST"

R(PRN)_0+D.RLOG             "ALU/A+B.RLOG,BMX/RBMX,RBMX/D,AMX/RAMX.OXT,DT/LONG,R(PRN)_ALU"
R(PRN)_ALU                  "SHF/ALU,SPO.AC/WRITE.RAB,SPO.ACN/PRN"
R(PRN)_D                    "RAMX/D,AMX/RAMX,ALU/A,SHF/ALU,SPO.AC/WRITE.RAB,SPO.ACN/PRN"
R(PRN)_D+K[].RLOG           "RAMX/D,AMX/RAMX,KMX/@1,BMX/KMX,ALU/A+B.RLOG,DT/LONG,R(PRN)_ALU"
R(PRN)_D-K[].RLOG           "RAMX/D,AMX/RAMX,KMX/@1,BMX/KMX,ALU/A-B.RLOG,DT/LONG,R(PRN)_ALU"
R(PRN)_D.OR.Q               "RAMX/D,AMX/RAMX,RBMX/Q,BMX/RBMX,ALU/OR,R(PRN)_ALU"
R(PRN)_D[]Q                 "RAMX/D,AMX/RAMX,RBMX/Q,BMX/RBMX,ALU/@1,R(PRN)_ALU"
R(PRN)_K[]                  "KMX/@1,BMX/KMX,ALU/B,SHF/ALU,SPO.AC/WRITE.RAB,SPO.ACN/PRN"
R(PRN)_LA+K[].RLOG          "AMX/LA,KMX/@1,BMX/KMX,ALU/A+B.RLOG,DT/LONG,R(PRN)_ALU"
R(PRN)_LA+Q                 "AMX/LA,RBMX/Q,BMX/RBMX,ALU/A+B,SHF/ALU,SPO.AC/WRITE.RAB,SPO.ACN/PRN"
R(PRN)_LA-K[].RLOG          "AMX/LA,KMX/@1,BMX/KMX,ALU/A-B.RLOG,DT/LONG,R(PRN)_ALU"
R(PRN)_LA[]MASK             "AMX/LA,BMX/MASK,ALU/@1,SHF/ALU,SPO.AC/WRITE.RAB,SPO.ACN/PRN"
R(PRN)_LC                   "BMX/LC,ALU/B,SHF/ALU,SPO.AC/WRITE.RAB,SPO.ACN/PRN"
R(PRN)_PACK.FP              "BMX/PACKED.FL,ALU/B,SHF/ALU,SPO.AC/WRITE.RAB,SPO.ACN/PRN"
R(PRN)_Q                    "RAMX/Q,AMX/RAMX,ALU/A,SHF/ALU,SPO.AC/WRITE.RAB,SPO.ACN/PRN"
R(PRN)_Q+K[].RLOG           "RAMX/Q,AMX/RAMX,KMX/@1,BMX/KMX,ALU/A+B.RLOG,DT/LONG,R(PRN)_ALU"
R(PRN)_Q-K[].RLOG           "RAMX/Q,AMX/RAMX,KMX/@1,BMX/KMX,ALU/A-B.RLOG,DT/LONG,R(PRN)_ALU"
R(PRN+1)_ALU                "SHF/ALU,SPO.AC/WRITE.RAB,SPO.ACN/PRN+1"
R(PRN+1)_D                  "RAMX/D,AMX/RAMX,ALU/A,SHF/ALU,SPO.AC/WRITE.RAB,SPO.ACN/PRN+1"
R(PRN+1)_D.OR.Q             "RAMX/D,AMX/RAMX,RBMX/Q,BMX/RBMX,ALU/OR,SHF/ALU,SPO.AC/WRITE.RAB,SPO.ACN/PRN+1"
R(PRN+1)_K[]                "KMX/@1,BMX/KMX,ALU/B,SHF/ALU,SPO.AC/WRITE.RAB,SPO.ACN/PRN+1"
R(PRN+1)_LA                 "AMX/LA,ALU/A,SHF/ALU,SPO.AC/WRITE.RAB,SPO.ACN/PRN+1"
R(PRN+1)_LC                 "BMX/LC,ALU/B,SHF/ALU,SPO.AC/WRITE.RAB,SPO.ACN/PRN+1"
R(PRN+1)_Q                  "RAMX/Q,AMX/RAMX,ALU/A,SHF/ALU,SPO.AC/WRITE.RAB,SPO.ACN/PRN+1"

R(SC)_ALU                   "SHF/ALU,SPO.AC/WRITE.RAB,SPO.ACN/SC"
R(SC)_D                     "RAMX/D,AMX/RAMX,ALU/A,SHF/ALU,SPO.AC/WRITE.RAB,SPO.ACN/SC"
R(SC)_K[]                   "KMX/@1,BMX/KMX,ALU/B,SHF/ALU,SPO.AC/WRITE.RAB,SPO.ACN/SC"
R(SC)_LA                    "AMX/LA,ALU/A,SHF/ALU,SPO.AC/WRITE.RAB,SPO.ACN/SC"
R(SC)_LA+D                  "AMX/LA,RBMX/D,BMX/RBMX,ALU/A+B,SHF/ALU,SPO.AC/WRITE.RAB,SPO.ACN/SC"
R(SC)_LA-D                  "AMX/LA,RBMX/D,BMX/RBMX,ALU/A-B,SHF/ALU,SPO.AC/WRITE.RAB,SPO.ACN/SC"
R(SC)_LC                    "ALU_LC,R(SC)_ALU"
R(SC)_Q                     "RAMX/Q,AMX/RAMX,ALU/A,SHF/ALU,SPO.AC/WRITE.RAB,SPO.ACN/SC"

R(SP1)_ALU                  "SHF/ALU,SPO.AC/WRITE.RAB,SPO.ACN/SP1.SP1"
R(SP1)_D                    "RAMX/D,AMX/RAMX,ALU/A,SHF/ALU,SPO.AC/WRITE.RAB,SPO.ACN/SP1.SP1"
R(SP1)_K[]                  "KMX/@1,BMX/KMX,ALU/B,SHF/ALU,SPO.AC/WRITE.RAB,SPO.ACN/SP1.SP1"
R(SP1)_PACK.FP              "BMX/PACKED.FL,ALU/B,SHF/ALU,SPO.AC/WRITE.RAB,SPO.ACN/SP1.SP1"
R(SP1)_Q                    "RAMX/Q,AMX/RAMX,ALU/A,SHF/ALU,SPO.AC/WRITE.RAB,SPO.ACN/SP1.SP1"
R(SP1+1)_LC                 "BMX/LC,ALU/B,SHF/ALU,SPO.AC/WRITE.RAB,SPO.ACN/SP1+1"
R(SP1+1)_Q                  "RAMX/Q,AMX/RAMX,ALU/A,SHF/ALU,SPO.AC/WRITE.RAB,SPO.ACN/SP1+1"

R(SRC!1)_ALU                "SHF/ALU,SPO.AC/WRITE.RAB,SPO.ACN11/SRC.OR.1"
R(SRC!1)_D(B)               "RBMX/D,BMX/RBMX,ALU/B,SHF/ALU,SPO.AC/WRITE.RAB,SPO.ACN11/SRC.OR.1"
R(SRC)_ALU                  "SHF/ALU,SPO.AC/WRITE.RAB,SPO.ACN11/SRC.SRC"
```

```
R(SRC)_D                        "RAMX/D,AMX/RAMX,ALU/A,SHF/ALU,SPO.AC/WRITE.RAB,SPO.ACN11/SRC.SRC"
R(SRC)_D(B)                     "RBMX/D,BMX/RBMX,ALU/B,SHF/ALU,SPO.AC/WRITE.RAB,SPO.ACN11/SRC.SRC"
R(SRC)_D+K[].RLOG               "RAMX/D,AMX/RAMX,KMX/@1,BMX/KMX,ALU/A+B.RLOG,DT/WORD,R(SRC)_ALU"
R(SRC)_D-K[].RLOG               "RAMX/D,AMX/RAMX,KMX/@1,BMX/KMX,ALU/A-B.RLOG,DT/WORD,R(SRC)_ALU"
R(SRC)_LC                       "BMX/LC,ALU/B,R(SRC)_ALU"
R(SRC)_Q                        "RAMX/Q,AMX/RAMX,ALU/A,SHF/ALU,SPO.AC/WRITE.RAB,SPO.ACN11/SRC.SRC"

R6_D+K[].RLOG                   "SPO.R/WRITE.RAB,SPO.RAB/R6,RAMX/D,AMX/RAMX,KMX/@1,BMX/KMX,ALU/A+B.RLOG,SHF/ALU"
R6_LA+K[].RLOG                  "AMX/LA,BMX/KMX,KMX/@1,ALU/A+B.RLOG,DT/WORD,SHF/ALU,SPO.R/WRITE.RAB,SPO.RAB/R6"
R6_LA-K[].RLOG                  "AMX/LA,BMX/KMX,KMX/@1,ALU/A-B.RLOG,DT/WORD,SHF/ALU,SPO.R/WRITE.RAB,SPO.RAB/R6"

RC(SC)_0-LC                     "ALU_0-LC,RC(SC)_ALU"
RC(SC)_ALU                      "SHF/ALU,SPO/WRITE.RC.SC"
RC(SC)_ALU.RIGHT                "SPO/WRITE.RC.SC,SHF/RIGHT"
RC(SC)_D                        "ALU_D,RC(SC)_ALU"
RC(SC)_Q                        "ALU_Q,RC(SC)_ALU"

RC[]&VA_D+Q                     "RAMX/D,AMX/RAMX,RBMX/Q,BMX/RBMX,ALU/A+B,VAK/LOAD,SHF/ALU,SPO.R/WRITE.RC,SPO.RC/@1"
RC[]_0                          "AMX/RAMX.0XT,DT/LONG,ALU/A,SHF/ALU,SPO.R/WRITE.RC,SPO.RC/@1"
RC[]_0+K[]+1                    "AMX/RAMX.0XT,DT/LONG,KMX/@2,BMX/KMX,ALU/A+B+1,SHF/ALU,SPO.R/WRITE.RC,SPO.RC/@1"
RC[]_0+LC+1                     "AMX/RAMX.0XT,DT/LONG,BMX/LC,ALU/A+B+1,SHF/ALU,SPO.R/WRITE.RC,SPO.RC/@1"
RC[]_0+MASK+1                   "AMX/RAMX.0XT,DT/LONG,BMX/MASK,ALU/A+B+1,SHF/ALU,SPO.R/WRITE.RC,SPO.RC/@1"
RC[]_0+MASK+1.RIGHT2            "AMX/RAMX.0XT,DT/LONG,BMX/MASK,ALU/A+B+1,SHF/RIGHT2,SPO.R/WRITE.RC,SPO.RC/@1"
RC[]_0-D                        "AMX/RAMX.0XT,DT/LONG,RBMX/D,BMX/RBMX,ALU/A-B,SHF/ALU,SPO.R/WRITE.RC,SPO.RC/@1"
RC[]_ALU                        "SHF/ALU,SPO.R/WRITE.RC,SPO.RC/@1"
RC[]_ALU.LEFT                   "SHF/LEFT,SPO.R/WRITE.RC,SPO.RC/@1"
RC[]_ALU.LEFT2                  "SPO.R/WRITE.RC,SPO.RC/@1,SHF/ALU.DT,DT/LONG"
RC[]_ALU.LEFT3                  "SPO.R/WRITE.RC,SPO.RC/@1,SHF/LEFT3"
RC[]_ALU.RIGHT                  "SHF/RIGHT,SPO.R/WRITE.RC,SPO.RC/@1"
RC[]_ALU.RIGHT2                 "SHF/RIGHT2,SPO.R/WRITE.RC,SPO.RC/@1"
RC[]_D                          "RAMX/D,AMX/RAMX,ALU/A,SHF/ALU,SPO.R/WRITE.RC,SPO.RC/@1"
RC[]_D(B)                       "RBMX/D,BMX/RBMX,ALU/B,SHF/ALU,SPO.R/WRITE.RC,SPO.RC/@1"
RC[]_D+K[]                      "RAMX/D,AMX/RAMX,BMX/KMX,KMX/@2,ALU/A+B,SHF/ALU,SPO.R/WRITE.RC,SPO.RC/@1"
RC[]_D-K[]                      "RAMX/D,AMX/RAMX,BMX/KMX,KMX/@2,ALU/A-B,SHF/ALU,SPO.R/WRITE.RC,SPO.RC/@1"
RC[]_D.0XT[]                    "RAMX/D,AMX/RAMX.0XT,DT/@2,ALU/A,SHF/ALU,SPO.R/WRITE.RC,SPO.RC/@1"
RC[]_D.AND.K[]                  "RAMX/D,AMX/RAMX,BMX/KMX,KMX/@2,ALU/AND,SHF/ALU,SPO.R/WRITE.RC,SPO.RC/@1"
RC[]_D.AND.MASK                 "RAMX/D,AMX/RAMX,BMX/MASK,ALU/AND,SHF/ALU,SPO.R/WRITE.RC,SPO.RC/@1"
RC[]_D.ANDNOT.Q                 "RAMX/D,AMX/RAMX,RBMX/Q,BMX/RBMX,ALU/ANDNOT,SHF/ALU,SPO.R/WRITE.RC,SPO.RC/@1"
RC[]_D.CTX                      "RAMX/D,AMX/RAMX,ALU/A,SHF/ALU.DT,DT/INST.DEP,SPO.R/WRITE.RC,SPO.RC/@1"
RC[]_D.LEFT                     "RAMX/D,AMX/RAMX,ALU/A,SHF/LEFT,SPO.R/WRITE.RC,SPO.RC/@1"
RC[]_D.LEFT3                    "RAMX/D,AMX/RAMX,ALU/A,SHF/LEFT3,SPO.R/WRITE.RC,SPO.RC/@1"
RC[]_D.OR.K[]                   "RAMX/D,AMX/RAMX,KMX/@2,BMX/KMX,ALU/OR,SHF/ALU,SPO.R/WRITE.RC,SPO.RC/@1"
RC[]_D.OR.Q                     "RAMX/D,AMX/RAMX,RBMX/Q,BMX/RBMX,ALU/OR,SHF/ALU,SPO.R/WRITE.RC,SPO.RC/@1"
RC[]_D.ORNOT.K[]                "SPO.RC/@1,SPO.R/WRITE.RC,ALU/ORNOT,AMX/RAMX,RAMX/D,BMX/KMX,KMX/@2,SHF/ALU"
RC[]_D.SXT[]                    "RAMX/D,AMX/RAMX.SXT,DT/@2,ALU/A,SHF/ALU,SPO.R/WRITE.RC,SPO.RC/@1"
RC[]_K[]                        "KMX/@2,BMX/KMX,ALU/B,SHF/ALU,SPO.R/WRITE.RC,SPO.RC/@1"
RC[]_K[]+1                      "AMX/RAMX.0XT,DT/LONG,KMX/@2,BMX/KMX,ALU/A+B+1,SHF/ALU,SPO.R/WRITE.RC,SPO.RC/@1"
RC[]_K[].LEFT2                  "KMX/@2,BMX/KMX,ALU/B,SHF/ALU.DT,DT/LONG,SPO.R/WRITE.RC,SPO.RC/@1"
RC[]_K[].LEFT3                  "KMX/@2,BMX/KMX,ALU/B,SHF/LEFT3,SPO.R/WRITE.RC,SPO.RC/@1"
RC[]_K[].RIGHT2                 "KMX/@2,BMX/KMX,ALU/B,SHF/RIGHT2,SPO.R/WRITE.RC,SPO.RC/@1"
RC[]_LA                         "AMX/LA,ALU/A,SHF/ALU,SPO.R/WRITE.RC,SPO.RC/@1"
RC[]_LA+LB.CTX                  "AMX/LA,BMX/LB,ALU/A+B,SHF/ALU.DT,DT/INST.DEP,SPO.R/WRITE.RC,SPO.RC/@1"
RC[]_LA-K[]                     "AMX/LA,KMX/@2,BMX/KMX,ALU/A-B,SHF/ALU,SPO.R/WRITE.RC,SPO.RC/@1"
RC[]_LA.AND.K[]                 "ALU_LA.AND.K[@2],RC[@1]_ALU"
RC[]_LA.CTX                     "AMX/LA,ALU/A,SHF/ALU,DT,DT/INST.DEP,SPO.R/WRITE.RC,SPO.RC/@1"
RC[]_LB                         "BMX/LB,ALU/B,SHF/ALU,SPO.R/WRITE.RC,SPO.RC/@1"
RC[]_LB.LEFT                    "BMX/LB,ALU/B,SHF/LEFT,SPO.R/WRITE.RC,SPO.RC/@1"
RC[]_LC                         "BMX/LC,ALU/B,SHF/ALU,SPO.R/WRITE.RC,SPO.RC/@1"
RC[]_NOT.Q                      "RAMX/Q,AMX/RAMX,ALU/NOTA,RC[@1]_ALU"
RC[]_PACK.FP                    "BMX/PACKED.FL,ALU/B,SHF/ALU,SPO.R/WRITE.RC,SPO.RC/@1"
RC[]_PC                         "BMX/PC,ALU/B,SHF/ALU,SPO.R/WRITE.RC,SPO.RC/@1"
RC[]_Q                          "RAMX/Q,AMX/RAMX,ALU/A,SHF/ALU,SPO.R/WRITE.RC,SPO.RC/@1"
RC[]_Q+1                        "ALU_0+Q+1,RC[@1]_ALU"
```

```
RC[]_Q+KC[]                 'RAMX/Q,AMX/RAMX,BMX/KMX,KMX/@2,ALU/A+B,SHF/ALU,SPO.R/WRITE.RC,SPO.RC/@1'
RC[]_Q+LC                   'ALU/A+B,RAMX/Q,AMX/RAMX,BMX/LC,SPO.R/WRITE.RC,SPO.RC/@1'
RC[]_Q+PC                   'RAMX/Q,AMX/RAMX,BMX/PC,ALU/A+B,SHF/ALU,SPO.R/WRITE.RC,SPO.RC/@1'
RC[]_Q+PC+1                 'RAMX/Q,AMX/RAMX,BMX/PC,ALU/A+B+1,SHF/ALU,SPO.R/WRITE.RC,SPO.RC/@1'
RC[]_Q-KC[]                 'RAMX/Q,AMX/RAMX,BMX/KMX,KMX/@2,ALU/A-B,SHF/ALU,SPO.R/WRITE.RC,SPO.RC/@1'
RC[]_Q-LC                   'ALU/A-B,RAMX/Q,AMX/RAMX,BMX/LC,SPO.R/WRITE.RC,SPO.RC/@1'
RC[]_Q-MASK-1               'RAMX/Q,AMX/RAMX,BMX/MASK,ALU/A-B-1,SHF/ALU,SPO.R/WRITE.RC,SPO.RC/@1'
RC[]_Q.OXT[]                'RAMX/Q,AMX/RAMX.OXT,DT/@2,ALU/A,SHF/ALU,SPO.R/WRITE.RC,SPO.RC/@1'
RC[]_Q.AND.KC[]             'RAMX/Q,AMX/RAMX,BMX/KMX,KMX/@2,ALU/AND,SHF/ALU,SPO.R/WRITE.RC,SPO.RC/@1'
RC[]_Q.ANDNOT.KC[]          'RAMX/Q,AMX/RAMX,BMX/KMX,KMX/@2,ALU/ANDNOT,SHF/ALU,SPO.R/WRITE.RC,SPO.RC/@1'
RC[]_Q.LEFT                 'RAMX/Q,AMX/RAMX,ALU/A,SHF/LEFT,SPO.R/WRITE.RC,SPO.RC/@1'
RC[]_Q.LEFT3                'RAMX/Q,AMX/RAMX,ALU/A,SHF/LEFT3,SPO.R/WRITE.RC,SPO.RC/@1'
RC[]_Q.RIGHT                'RAMX/Q,AMX/RAMX,ALU/A,SHF/RIGHT,SPO.R/WRITE.RC,SPO.RC/@1'
RC[]_Q.RIGHT2               'ALU_Q,SHF/RIGHT2,SPO.R/WRITE.RC,SPO.RC/@1'
RC[]_Q.SXT[]                'RAMX/Q,AMX/RAMX.SXT,DT/@2,ALU/A,SHF/ALU,SPO.R/WRITE.RC,SPO.RC/@1'
RC[]_RLOG.RIGHT             'BMX/0,MSC/READ.RLOG,ALU/B,SHF/RIGHT,SPO.R/WRITE.RC,SPO.RC/@1'

RC[]%VA_LA+KC[]             'AMX/LA,KMX/@2,BMX/KMX,ALU/A+B,VAK/LOAD,SHF/ALU,SPO.R/WRITE.RAB,SPO.RAB/@1'
RC[]%VA_LA-KC[]             'AMX/LA,KMX/@2,BMX/KMX,ALU/A-B,VAK/LOAD,SHF/ALU,SPO.R/WRITE.RAB,SPO.RAB/@1'
RC[]%VA_LA-KC[].RLOG        'AMX/LA,KMX/@2,BMX/KMX,ALU/A-B.RLOG,DT/LONG,VAK/LOAD,SHF/ALU,SPO.R/WRITE.RAB,SPO.RAB/@1'
RC[]%VA_Q-KC[]              'RAMX/Q,AMX/RAMX,KMX/@2,BMX/KMX,ALU/A-B,VAK/LOAD,SPO.R/WRITE.RAB,SPO.RAB/@1'
RC[]_0                      'SPO.R/WRITE.RAB,SPO.RAB/@1,AMX/RAMX.OXT,DT/LONG,ALU/A,SHF/ALU'
RC[]_0+LB+1                 'AMX/RAMX.OXT,DT/LONG,BMX/LB,ALU/A+B+1,SHF/ALU,SPO.R/WRITE.RAB,SPO.RAB/@1'
RC[]_0-1                    'AMX/RAMX.OXT,DT/LONG,BMX/KMX,KMX/.1,ALU/A-B,SHF/ALU,SPO.R/WRITE.RAB,SPO.RAB/@1'
RC[]_0-D                    'AMX/RAMX.OXT,DT/LONG,RBMX/D,BMX/RBMX,ALU/A-B,SHF/ALU,SPO.R/WRITE.RAB,SPO.RAB/@1'
RC[]_0-KC[]                 'AMX/RAMX.OXT,DT/LONG,KMX/@2,BMX/KMX,ALU/A-B,SHF/ALU,SPO.R/WRITE.RAB,SPO.RAB/@1'
RC[]_0-LB                   'AMX/RAMX.OXT,DT/LONG,BMX/LB,ALU/A-B,SHF/ALU,SPO.R/WRITE.RAB,SPO.RAB/@1'
RC[]_0-Q                    'AMX/RAMX.OXT,DT/LONG,RBMX/Q,BMX/RBMX,ALU/A-B,SHF/ALU,SPO.R/WRITE.RAB,SPO.RAB/@1'
RC[]_ALU                    'SHF/ALU,SPO.R/WRITE.RAB,SPO.RAB/@1'
RC[]_ALU.LEFT               'SPO.R/WRITE.RAB,SPO.RAB/@1,SHF/LEFT'
RC[]_ALU.LEFT3              'SPO.R/WRITE.RAB,SPO.RAB/@1,SHF/LEFT3'
RC[]_ALU.RIGHT              'SHF/RIGHT,SPO.R/WRITE.RAB,SPO.RAB/@1'
RC[]_ALU.RIGHT2             'SPO.R/WRITE.RAB,SPO.RAB/@1,SHF/RIGHT2'
RC[]_D                      'SPO.R/WRITE.RAB,SPO.RAB/@1,RAMX/D,AMX/RAMX,ALU/A,SHF/ALU'
RC[]_D+KC[]                 'SPO.R/WRITE.RAB,SPO.RAB/@1,RAMX/D,AMX/RAMX,KMX/@2,BMX/KMX,ALU/A+B,SHF/ALU'
RC[]_D+Q                    'SPO.R/WRITE.RAB,SPO.RAB/@1,RAMX/D,AMX/RAMX,RBMX/Q,BMX/RBMX,ALU/A+B,SHF/ALU'
RC[]_D+Q+1                  'SPO.R/WRITE.RAB,SPO.RAB/@1,RAMX/D,AMX/RAMX,RBMX/Q,BMX/RBMX,ALU/A+B+1,SHF/ALU'
RC[]_D-KC[]                 'SPO.R/WRITE.RAB,SPO.RAB/@1,RAMX/D,AMX/RAMX,KMX/@2,BMX/KMX,ALU/A-B,SHF/ALU'
RC[]_D-LC-1                 'ALU_D-LC-1,RC@1]_ALU'
RC[]_D-Q                    'SPO.R/WRITE.RAB,SPO.RAB/@1,RAMX/D,AMX/RAMX,RBMX/Q,BMX/RBMX,ALU/A-B,SHF/ALU'
RC[]_D.AND.KC[]             'SPO.R/WRITE.RAB,SPO.RAB/@1,ALU/AND,AMX/RAMX,RAMX/D,BMX/KMX,KMX/@2,SHF/ALU'
RC[]_D.OR.LC                'SPO.R/WRITE.RAB,SPO.RAB/@1,ALU/OR,AMX/RAMX,RAMX/D,BMX/LC,SHF/ALU'
RC[]_D.OR.PACK.FP           'SPO.R/WRITE.RAB,SPO.RAB/@1,ALU/OR,AMX/RAMX,RAMX/D,BMX/PACKED.FL,SHF/ALU'
RC[]_D.OR.Q                 'SPO.R/WRITE.RAB,SPO.RAB/@1,RAMX/D,AMX/RAMX,RBMX/Q,BMX/RBMX,ALU/OR,SHF/ALU'
RC[]_KC[]                   'BMX/KMX,KMX/@2,ALU/B,SHF/ALU,SPO.R/WRITE.RAB,SPO.RAB/@1'
RC[]_LA                     'SPO.R/WRITE.RAB,SPO.RAB/@1,AMX/LA,ALU/A,SHF/ALU'
RC[]_LA+D                   'AMX/LA,RBMX/D,BMX/RBMX,ALU/A+B,SHF/ALU,SPO.R/WRITE.RAB,SPO.RAB/@1'
RC[]_LA+D+1                 'AMX/LA,RBMX/D,BMX/RBMX,ALU/A+B+1,SHF/ALU,SPO.R/WRITE.RAB,SPO.RAB/@1'
RC[]_LA+KC[]                'AMX/LA,BMX/KMX,KMX/@2,ALU/A+B,SHF/ALU,SPO.R/WRITE.RAB,SPO.RAB/@1'
RC[]_LA+KC[]+1              'AMX/LA,BMX/KMX,KMX/@2,ALU/A+B+1,RC@1]_ALU'
RC[]_LA+KC[].RLOG           'AMX/LA,BMX/KMX,KMX/@2,ALU/A+B.RLOG,DT/LONG,SHF/ALU,SPO.R/WRITE.RAB,SPO.RAB/@1'
RC[]_LA+LC                  'AMX/LA,BMX/LC,ALU/A+B,SHF/ALU,SPO.R/WRITE.RAB,SPO.RAB/@1'
RC[]_LA+MASK+1              'AMX/LA,BMX/MASK,ALU/A+B+1,RC@1]_ALU'
RC[]_LA+Q                   'AMX/LA,RBMX/Q,BMX/RBMX,ALU/A+B,SHF/ALU,SPO.R/WRITE.RAB,SPO.RAB/@1'
RC[]_LA-D                   'AMX/LA,RBMX/D,BMX/RBMX,ALU/A-B,SHF/ALU,SPO.R/WRITE.RAB,SPO.RAB/@1'
RC[]_LA-KC[]                'AMX/LA,BMX/KMX,KMX/@2,ALU/A-B,SHF/ALU,SPO.R/WRITE.RAB,SPO.RAB/@1'
RC[]_LA-KC[].RLOG           'AMX/LA,BMX/KMX,KMX/@2,ALU/A-B.RLOG,DT/LONG,SHF/ALU,SPO.R/WRITE.RAB,SPO.RAB/@1'
RC[]_LA-MASK-1              'ALU/A-B-1,AMX/LA,BMX/MASK,SPO.R/WRITE.RAB,SPO.RAB/@1,SHF/ALU'
RC[]_LA-Q                   'AMX/LA,RBMX/Q,BMX/RBMX,ALU/A-B,SHF/ALU,SPO.R/WRITE.RAB,SPO.RAB/@1'
RC[]_LA.AND.KC[]            'AMX/LA,BMX/KMX,KMX/@2,ALU/AND,SHF/ALU,SPO.R/WRITE.RAB,SPO.RAB/@1'
RC[]_LA.OR.D                'AMX/LA,RBMX/D,BMX/RBMX,ALU/OR,SHF/ALU,SPO.R/WRITE.RAB,SPO.RAB/@1'
RC[]_LA.ORNOT.MASK          'AMX/LA,BMX/MASK,ALU/ORNOT,SHF/ALU,SPO.R/WRITE.RAB,SPO.RAB/@1'
RC[]_LB                     'BMX/LB,ALU/B,SHF/ALU,SPO.R/WRITE.RAB,SPO.RAB/@1'
```

```
R[]_LC                          "BMX/LC,ALU/B,SHF/ALU,SPO.R/WRITE.RAB,SPO.RAB/@1"
R[]_LC.RIGHT                    "BMX/LC,ALU/B,SHF/RIGHT,SPO.R/WRITE.RAB,SPO.RAB/@1"
R[]_NOT.Q                       "AMX/RAMX.OXT,DT/LONG,ALU/NOTA,R[@1]_ALU"
R[]_NOT.D                       "RAMX/D,AMX/RAMX,ALU/NOTA,R[@1]_ALU"
R[]_NOT.MASK                    "BMX/MASK,AMX/RAMX.OXT,DT/LONG,ALU/ORNOT,SHF/ALU,SPO.R/WRITE.RAB,SPO.RAB/@1"
R[]_NOT.Q                       "RAMX/Q,AMX/RAMX,ALU/NOTA,R[@1]_ALU"
R[]_PACK.FF                     "BMX/PACKED.FL,ALU/B,SHF/ALU,SPO.R/WRITE.RAB,SPO.RAB/@1"
R[]_Q                           "SPO.R/WRITE.RAB,SPO.RAB/@1,RAMX/Q,AMX/RAMX,ALU/A,SHF/ALU"
R[]_Q+1                         "ALU_Q+Q+1,R[@1]_ALU"
R[]_Q+5                         "SPO.R/WRITE.RAB,SPO.RAB/@1,ALU/A+B+1,BMX/KMX,KMX/.4,AMX/RAMX,RAMX/Q,SHF/ALU"
R[]_Q+K[]                       "SPO.R/WRITE.RAB,SPO.RAB/@1,RAMX/Q,AMX/RAMX,BMX/KMX,KMX/@2,ALU/A+B,SHF/ALU"
R[]_Q+LB                        "SPO.R/WRITE.RAB,SPO.RAB/@1,ALU/A+B,AMX/RAMX,BMX/LB,RAMX/Q,SHF/ALU"
R[]_Q+LC                        "SPO.R/WRITE.RAB,SPO.RAB/@1,RAMX/Q,AMX/RAMX,BMX/LC,ALU/A+B,SHF/ALU"
R[]_Q-D                         "SPO.R/WRITE.RAB,SPO.RAB/@1,RAMX/Q,AMX/RAMX,RBMX/D,BMX/RBMX,ALU/A-B,SHF/ALU"
R[]_Q-D-1                       "SPO.R/WRITE.RAB,SPO.RAB/@1,ALU/A-B-1,AMX/RAMX,RAMX/Q,BMX/RBMX,RBMX/D,SHF/ALU"
R[]_Q-K[]                       "SPO.R/WRITE.RAB,SPO.RAB/@1,RAMX/Q,AMX/RAMX,BMX/KMX,KMX/@2,ALU/A-B,SHF/ALU"
R[]_Q-K[].RLOG                  "RAMX/Q,AMX/RAMX,BMX/KMX,KMX/@2,ALU/A-B.RLOG,DT/LONG,SHF/ALU,SPO.R/WRITE.RAB,SPO.RAB/@1"
R[]_Q-LC                        "SPO.R/WRITE.RAB,SPO.RAB/@1,RAMX/Q,AMX/RAMX,BMX/LC,ALU/A-B,SHF/ALU"
R[]_Q.AND.K[]                   "ALU/AND,SPO.R/WRITE.RAB,SPO.RAB/@1,AMX/RAMX,RAMX/Q,BMX/KMX,KMX/@2"
R[]_Q.ANDNOT.K[]                "SPO.R/WRITE.RAB,SPO.RAB/@1,ALU/ANDNOT,AMX/RAMX,RAMX/Q,BMX/KMX,KMX/@2,SHF/ALU"
R[]_Q.OR.D                      "SPO.R/WRITE.RAB,SPO.RAB/@1,ALU/OR,AMX/RAMX,RAMX/Q,BMX/RBMX,RBMX/D,SHF/ALU"
R[]_Q.ORNOT.K[]                 "SPO.R/WRITE.RAB,SPO.RAB/@1,RAMX/Q,AMX/RAMX,BMX/KMX,KMX/@2,ALU/ORNOT,SHF/ALU"
R[]_Q.RIGHT.1                   "ALU_Q,SHF/RIGHT,SPO.R/WRITE.RAB,SPO.RAB/@1"
R[]_RLOG.RIGHT.1                "BMX/0,MSC/READ.RLOG,ALU/B,SHF/RIGHT,SPO.R/WRITE.RAB,SPO.RAB/@1"

SC$STATE_STATE-R[](EXP)         "LAB_R[@1],AMX/LA,EBMX/AMX.EXP,MSC/LOAD.STATE,EALU/A-B,SMX/EALU,SCK/LOAD"
SC_0(A)                         "AMX/RAMX.OXT,DT/LONG,EBMX/AMX.EXP,EALU/B,SMX/EALU,SCK/LOAD"
SC_0-K[]                        "BMX/KMX,KMX/@1,AMX/RAMX.OXT,DT/LONG,ALU/A-B,SMX/ALU,SCK/LOAD"
SC_ALU                          "SMX/ALU,SCK/LOAD"
SC_ALU(EXP)                     "SMX/ALU.EXP,SCK/LOAD"
SC_D                            "RAMX/D,AMX/RAMX,ALU/A,SMX/ALU,SCK/LOAD"
SC_D(EXP)                       "RAMX/D,AMX/RAMX,ALU/A,SMX/ALU.EXP,SCK/LOAD"
SC_D(EXP)(A)                    "RAMX/D,AMX/RAMX,EBMX/AMX.EXP,EALU/B,SMX/EALU,SCK/LOAD"
SC_D(EXP)(B)                    "RBMX/D,BMX/RBMX,ALU/B,SMX/ALU.EXP,SCK/LOAD"
SC_D-K[]                        "RAMX/D,AMX/RAMX,KMX/@1,BMX/KMX,ALU/A-B,SMX/ALU,SCK/LOAD"
SC_D.OXT[]-K[]                  "RAMX/D,AMX/RAMX.OXT,DT/@1,KMX/@2,BMX/KMX,ALU/A-B,SMX/ALU,SCK/LOAD"
SC_D.OXT[].XOR.K[]              "RAMX/D,AMX/RAMX.OXT,DT/@1,BMX/KMX,KMX/@2,ALU/XOR,SC_ALU"
SC_D.AND.K[]                    "RAMX/D,AMX/RAMX,KMX/@1,BMX/KMX,ALU/AND,SMX/ALU,SCK/LOAD"
SC_D.OR.K[]                     "RAMX/D,AMX/RAMX,KMX/@1,BMX/KMX,ALU/OR,SMX/ALU,SCK/LOAD"
SC_D.SXT[]                      "RAMX/D,AMX/RAMX.SXT,DT/@1,ALU/A,SMX/ALU,SCK/LOAD"
SC_EALU                         "SMX/EALU,SCK/LOAD"
SC_FE                           "SMX/FE,SCK/LOAD"
SC_K[]                          "KMX/@1,EBMX/KMX,EALU/B,SMX/EALU,SCK/LOAD"
SC_K[].ALU                      "KMX/@1,BMX/KMX,ALU/B,SMX/ALU,SCK/LOAD"
SC_LA                           "AMX/LA,ALU/A,SMX/ALU,SCK/LOAD"
SC_LA.AND.K[]                   "AMX/LA,KMX/@1,BMX/KMX,ALU/AND,SMX/ALU,SCK/LOAD"
SC_LC(EXP)                      "BMX/LC,ALU/B,SMX/ALU.EXP,SCK/LOAD"
SC_NABS(SC-FE)                  "EBMX/FE,EALU/NABS.A-B,SMX/EALU,SCK/LOAD"
SC_PSLADDR                      "SMX/EALU,EBMX/KMX,SCK/LOAD,KMX/.F,EALU/B"
SC_Q                            "RAMX/Q,AMX/RAMX,ALU/A,SMX/ALU,SCK/LOAD"
SC_Q(EXP)                       "RAMX/Q,AMX/RAMX,EBMX/AMX.EXP,EALU/B,SMX/EALU,SCK/LOAD"
SC_Q(EXP)(B)                    "RBMX/Q,BMX/RBMX,ALU/B,SMX/ALU.EXP,SCK/LOAD"
SC_Q+K[]                        "RAMX/Q,AMX/RAMX,BMX/KMX,KMX/@1,ALU/A+B,SMX/ALU,SCK/LOAD"
SC_Q-K[]                        "RAMX/Q,AMX/RAMX,BMX/KMX,KMX/@1,ALU/A-B,SMX/ALU,SCK/LOAD"
SC_Q.AND.K[]                    "RAMX/Q,AMX/RAMX,BMX/KMX,KMX/@1,ALU/AND,SMX/ALU,SCK/LOAD"
SC_Q.OR.K[]                     "RAMX/Q,AMX/RAMX,BMX/KMX,KMX/@1,ALU/OR,SMX/ALU,SCK/LOAD"
SC_Q.SXT[]                      "RAMX/Q,AMX/RAMX.SXT,DT/@1,ALU/A,SMX/ALU,SCK/LOAD"
SC_RC[]                         "SPO.R/LOAD.LC,SPO.RC/@1,BMX/LC,ALU/B,SMX/ALU,SCK/LOAD"
SC_RC[](EXP)                    "SPO.R/LOAD.LC,SPO.RC/@1,BMX/LC,ALU/B,SMX/ALU.EXP,SCK/LOAD"
SC_R[]                          "SPO.R/LOAD.LAB,SPO.RAB/@1,AMX/LA,ALU/A,SMX/ALU,SCK/LOAD"
SC_R[](EXP)                     "SPO.R/LOAD.LAB,SPO.RAB/@1,AMX/LA,ALU/A,SMX/ALU.EXP,SCK/LOAD"
SC_R[].AND.K[]                  "ALU/AND,AMX/LA,SPO.R/LOAD.LAB,SPO.RAB/@1,BMX/KMX,KMX/@2,SMX/ALU,SCK/LOAD"
SC_SC+1                         "EALU/A+1,SMX/EALU,SCK/LOAD"
```

```
SC_SC+EXP(Q)(A)                 "EALU/A+B,EBMX/AMX.EXP,SMX/EALU,SCK/LOAD,AMX/RAMX,RAMX/Q"
SC_SC+FE                        "EBMX/FE,EALU/A+B,SMX/EALU,SCK/LOAD"
SC_SC+K[]                       "KMX/@1,EBMX/KMX,EALU/A+B,SMX/EALU,SCK/LOAD"
SC_SC+SHF.VAL                   "EALU/A+B,EBMX/SHF.VAL,SMX/EALU,SCK/LOAD"
SC_SC-FE                        "EBMX/FE,EALU/A-B,SMX/EALU,SCK/LOAD"
SC_SC-K[]                       "KMX/@1,EBMX/KMX,EALU/A-B,SMX/EALU,SCK/LOAD"
SC_SC-SHF.VAL                   "EBMX/SHF.VAL,EALU/A-B,SMX/EALU,SCK/LOAD"
SC_SC.ANDNOT.FE                 "EBMX/FE,EALU/ANDNOT,SMX/EALU,SCK/LOAD"
SC_SC.ANDNOT.K[]                "KMX/@1,EBMX/KMX,EALU/ANDNOT,SMX/EALU,SCK/LOAD"
SC_SC.OR.K[]                    "KMX/@1,EBMX/KMX,EALU/OR,SMX/EALU,SCK/LOAD"
SC_SHF.VAL                      "EBMX/SHF.VAL,EALU/B,SMX/EALU,SCK/LOAD"
SC_STATE                        "EALU/A,MSC/LOAD.STATE,SMX/EALU,SCK/LOAD"
SC_STATE.ANDNOT.K[]             "EALU/ANDNOT,EBMX/KMX,MSC/LOAD.STATE,SMX/EALU,SCK/LOAD,KMX/@1"
SC_STATE.OR.K[]                 "EALU/OR,EBMX/KMX,MSC/LOAD.STATE,SMX/EALU,SCK/LOAD,KMX/@1"
SD_NOT.SD                       "SGN/NOT.SD"
SD_SS                           "SGN/SD.FROM.SS"
SS_0&SD_0                       "SGN/CLR.SD+SS"
SS_ALU15                        "SGN/LOAD.SS"
SS_SD                           "SGN/SS.FROM.SD"
SS_SS.XOR.ALU15&SD_ALU15        "SGN/SS.XOR.ALU"
STATE_0(A)                      "AMX/RAMX.OXT,DT/LONG,EBMX/AMX.EXP,EALU/B,MSC/LOAD.STATE"
STATE_AMX.EXP                   "EBMX/AMX.EXP,EALU/B,MSC/LOAD.STATE"
STATE_D(EXP)                    "RAMX/D,AMX/RAMX,EBMX/AMX.EXP,EALU/B,MSC/LOAD.STATE"
STATE_FE                        "EBMX/FE,EALU/B,MSC/LOAD.STATE"
STATE_FIRST                     "STATE_K[ZERO]"                         ;EDITPC STATES
STATE_INNEROBJ                  "STATE_K[.1]"                           ;MATCHC STATES
STATE_INNERSRC                  "STATE_K[.3]"
STATE_K[]                       "KMX/@1,EBMX/KMX,EALU/B,MSC/LOAD.STATE"
STATE_OUTER                     "STATE_K[ZERO]"
STATE_PREDEC                    "STATE_K[.80]"
STATE_Q(EXP)                    "RAMX/Q,AMX/RAMX,EBMX/AMX.EXP,EALU/B,MSC/LOAD.STATE"
STATE_SC.VIA.KMX                "MSC/LOAD.STATE,EALU/B,EBMX/KMX,KMX/SC"
STATE_SKPLONG                   "STATE_K[.4]"                           ;SKPC STATES
STATE_STATE+1                   "EALU/A+1,MSC/LOAD.STATE"
STATE_STATE+FE                  "EBMX/FE,EALU/A+B,MSC/LOAD.STATE"
STATE_STATE+K[]                 "KMX/@1,EBMX/KMX,EALU/A+B,MSC/LOAD.STATE"
STATE_STATE-FE                  "EBMX/FE,EALU/A-B,MSC/LOAD.STATE"
STATE_STATE-K[]                 "KMX/@1,EBMX/KMX,EALU/A-B,MSC/LOAD.STATE"
STATE_STATE.AN.SKPLONG          "STATE_STATE.ANDNOT.K[.4]"
STATE_STATE.AN.5TO0             "STATE_STATE.ANDNOT.K[.3F]"
STATE_STATE.AN.6TO4             "STATE_STATE.ANDNOT.K[.7F]"
STATE_STATE.AN.DESTDBL          "STATE_STATE.ANDNOT.K[.6]"
STATE_STATE.AN.NOTPREDEC        "STATE_STATE.ANDNOT.K[.7F]"
STATE_STATE.AN.PREDECZERO       "STATE_STATE.ANDNOT.K[.C0]"
STATE_STATE.ANDNOT.FE           "EBMX/FE,EALU/ANDNOT,MSC/LOAD.STATE"
STATE_STATE.ANDNOT.K[]          "KMX/@1,EBMX/KMX,EALU/ANDNOT,MSC/LOAD.STATE"
STATE_STATE.ANDNOT.SHF.VAL      "MSC/LOAD.STATE,EBMX/SHF.VAL,EALU/ANDNOT"
STATE_STATE.OR.FE               "EALU/OR,EBMX/FE,MSC/LOAD.STATE"
STATE_STATE.OR.K[]              "KMX/@1,EBMX/KMX,EALU/OR,MSC/LOAD.STATE"
STATE_STATE.OR.ADJINP           "STATE_STATE.OR.K[.3]"
STATE_STATE.OR.DEST             "STATE_STATE.OR.K[.4]"
STATE_STATE.OR.DESTDBL          "STATE_STATE.OR.K[.6]"
STATE_STATE.OR.FILL             "STATE_STATE.OR.K[.7]"
STATE_STATE.OR.FLOAT            "STATE_STATE.OR.K[.60]"
STATE_STATE.OR.MOVE             "STATE_STATE.OR.K[.50]"
STATE_STATE.OR.PATT1            "STATE_STATE.OR.K[.1]"
STATE_STATE.OR.PATT2            "STATE_STATE.OR.K[.2]"
SWAPD                           "DK/BYTE.SWAP"

VA_ALU                          "VAK/LOAD"
VA_D                            "RAMX/D,AMX/RAMX,ALU/A,VAK/LOAD"
VA_D+K[]                        "RAMX/D,AMX/RAMX,KMX/@1,BMX/KMX,ALU/A+B,VAK/LOAD"
VA_D+LC                         "RAMX/D,AMX/RAMX,BMX/LC,ALU/A+B,VAK/LOAD"
```

```
VA_D+Q                  "RAMX/D,AMX/RAMX,RBMX/Q,BMX/RBMX,ALU/A+B,VAK/LOAD"
VA_D.OXT[]+Q            "RAMX/D,AMX/RAMX.OXT,DT/@1,BMX/RBMX,ALU/A+B,VAK/LOAD"
VA_D.ANDNOT.K[]         "RAMX/D,AMX/RAMX,BMX/KMX,KMX/@1,ALU/ANDNOT,VAK/LOAD"
VA_K[]                  "KMX/@1,BMX/KMX,ALU/B,VAK/LOAD"
VA_LA                   "AMX/LA,ALU/A,VAK/LOAD"
VA_LA+D                 "AMX/LA,RBMX/D,BMX/RBMX,ALU/A+B,VAK/LOAD"
VA_LA+K[]               "AMX/LA,BMX/KMX,KMX/@1,ALU/A+B,VAK/LOAD"
VA_LA+K[]+1             "AMX/LA,BMX/KMX,KMX/@1,ALU/A+B+1,VAK/LOAD"
VA_LA+PC                "AMX/LA,BMX/PC,ALU/A+B,VAK/LOAD"
VA_LA+Q                 "AMX/LA,RBMX/Q,BMX/RBMX,ALU/A+B,VAK/LOAD"
VA_LA-D                 "AMX/LA,RBMX/D,BMX/RBMX,ALU/A-B,VAK/LOAD"
VA_LA-K[]               "AMX/LA,BMX/KMX,KMX/@1,ALU/A-B,VAK/LOAD"
VA_LA-K[]-1             "AMX/LA,BMX/KMX,KMX/@1,ALU/A-B-1,VAK/LOAD"
VA_LA-Q                 "VAK/LOAD,ALU/A-B,AMX/LA,BMX/RBMX,RBMX/Q,SHF/ALU"
VA_LA.AND.LC            "AMX/LA,BMX/LC,ALU/AND,VAK/LOAD"
VA_LA.ANDNOT.K[]        "AMX/LA,BMX/KMX,KMX/@1,ALU/ANDNOT,VAK/LOAD"
VA_LB+D.OXT             "BMX/LB,ALU/A+B,AMX/RAMX.OXT,DT/BYTE,VAK/LOAD"
VA_PC                   "BMX/PC,ALU/B,VAK/LOAD"
VA_Q                    "RAMX/Q,AMX/RAMX,ALU/A,VAK/LOAD"
VA_Q+D                  "VAK/LOAD,ALU/A+B,AMX/RAMX,BMX/RBMX,RAMX/Q,RBMX/D,SHF/ALU"
VA_Q+K[]                "RAMX/Q,AMX/RAMX,KMX/@1,BMX/KMX,ALU/A+B,VAK/LOAD"
VA_Q+LB                 "RAMX/Q,AMX/RAMX,BMX/LB,ALU/A+B,VAK/LOAD"
VA_Q+LB.PC              "RAMX/Q,AMX/RAMX,BMX/PC.OR.LB,ALU/A+B,VAK/LOAD"
VA_Q+LC                 "RAMX/Q,AMX/RAMX,BMX/LC,ALU/A+B,VAK/LOAD"
VA_Q+PC                 "RAMX/Q,AMX/RAMX,BMX/PC,ALU/A+B,VAK/LOAD"
VA_Q-K[]                "RAMX/Q,AMX/RAMX,KMX/@1,BMX/KMX,ALU/A-B,VAK/LOAD"
VA_Q-LB                 "RAMX/Q,AMX/RAMX,BMX/LB,ALU/A-B,VAK/LOAD"
VA_Q.ANDNOT.K[]         "RAMX/Q,AMX/RAMX,KMX/@1,BMX/KMX,ALU/ANDNOT,VAK/LOAD"
VA_RC[]                 "SPO.R/LOAD.LC,SPO.RC/@1,BMX/LC,ALU/B,VAK/LOAD"
VA_R[]                  "SPO.R/LOAD.LAB,SPO.RAB/@1,AMX/LA,ALU/A,VAK/LOAD"
VA_VA+4                 "PCK/VA+4"
```

```
.TOC    "Macro definition      : Non-transfer macros"

B.FORK                          "LAB_R(SP1),QK/ID,CLR.IB.COND,PC_PC+N,SUB/SPEC,J/B.FORK"
BYTE                            "DT/BYTE"

C.FORK                          "SUB/SPEC,J/C.FORK"
CACHE.INVALIDATE                "MCT/INVALIDATE,VAK/NOP"
CALL                            "SUB/CALL"
CALL[]                          "CALL,J/@1"
CHK.FLT.OPR                     "MSC/CHK.FLT.OPR"
CHK.ODD.ADDR                    "MSC/CHK.ODD.ADDR"
CLK.UBCC                        "CCK/LOAD.UBCC"

CLR.FPD                         "MSC/CLR.FPD"
CLR.IB.COND                     "IBC/CLR.1-5.COND"
CLR.IB.OPC                      "IBC/CLR.0,IEK/ISTR"
CLR.IB.SPEC                     "IBC/CLR.1"
CLR.IB0-1                       "IBC/CLR.0.1,IEK/ISTR"
CLR.IB0-3                       "IBC/CLR.0-3"           !DISCARD -11 INSTR & OPERAND
CLR.IB2-3                       "IBC/CLR.2.3"           !11 MODE DISCARD ISTREAM OPERAND
CLR.IB2-5                       "IBC/CLR.1-5.COND"      !2ND PART OF Q/D IMMEDIATE
CLR.NEST.ERR                    "MSC/CLR.NEST.ERR"
CLR.SD&SS                       "SGN/CLR.SD+SS"

E.FORK                          "SUB/SPEC,J/E.FORK"
EXCEPT.ACK                      "IEK/EACK"

FLUSH.IB                        "IBC/FLUSH,VAK/LOAD,IEK/ISTR"

G.FORK                          "SUB/SPEC,J/G.FORK"

INHIBIT.IB                      "MCT/MEM.NOP"
INTRPT.ACK                      "IEK/IACK"
INTRPT.STROBE                   "IEK/ISTR"
IRD                             "IRD0,CLK.UBCC,IRD1,SUB/SPEC,J/A.FORK"
IRD.11                          "LA_R(DST)&LB_R(SRC),D_LB.PC,VAK/LOAD,Q_IB.DATA,SC_K[.10],PCK/PC+N,MSC/IRD,SUB/SPEC,J/DP0"
IRD0                            "LA_R(SP2)&LB_R(SP1),D&VA_LB,SC_ALU(EXP),FE_LA(EXP),SS_ALU15"
IRD1                            "MSC/IRD,QK/ID,MCT/ALLOW.IB.READ,IBC/CLR.1-5.COND,PCK/PC+N"

LOAD.ACC.CC                     "MSC/LOAD.ACC.CC"
LOAD.IB                         "VAK/NOP,MCT/READ.V.NEWPC"
LOAD.IB.11                      "VAK/NOP,MCT/READ.V.NEWPC"
LONG                            "DT/LONG"

MEMORY.NOP                      "MCT/MEM.NOP"
MUL.0XT                         "SI/MUL+,SC_SC-K[.1],BEN/MUL"
MUL.1XT                         "SI/MUL-,SC_SC-K[.1],BEN/MUL"
MULM.DONE                       "D_D.RIGHT2,SI/MUL-,INTRPT.STROBE"
MULP.DONE                       "D_D.RIGHT2,SI/MUL+,INTRPT.STROBE"

POLY.DONE                       "ACF/CONTROL,ACM/POLY.DONE"

RETURN0                         "SUB/RET,J/0"
RETURN1                         "SUB/RET,J/1"
RETURN10                        "SUB/RET,J/10"
RETURN100                       "SUB/RET,J/100"
RETURN10C                       "SUB/RET,J/10C"
RETURN10E                       "SUB/RET,J/10E"
RETURN12                        "SUB/RET,J/12"
RETURN18                        "SUB/RET,J/18"
RETURN1F                        "SUB/RET,J/1F"
RETURN2                         "SUB/RET,J/2"
RETURN20                        "SUB/RET,J/20"
```

```
RETURN24                    "SUB/RET,J/24"
RETURN3                     "SUB/RET,J/3"
RETURN4                     "SUB/RET,J/4"
RETURN40                    "SUB/RET,J/40"
RETURN60                    "SUB/RET,J/60"
RETURN61                    "SUB/RET,J/61"
RETURN8                     "SUB/RET,J/8"
RETURN9                     "SUB/RET,J/9"
RETURNF                     "SUB/RET,J/0F"
RETURN[]                    "SUB/RET,J/@1"

SET.CC(BYTE)                "CCK/INST.DEP,DT/BYTE"
SET.CC(INST)                "CCK/INST.DEP,DT/INST.DEP"
SET.CC(LONG)                "CCK/INST.DEP,DT/LONG"
SET.CC(ROR)                 "CCK/ROR"
SET.CC(WORD)                "CCK/INST.DEP,DT/WORD"
SET.FPD                     "MSC/SET.FPD"
SET.NEST.ERR                "MSC/SET.NEST.ERR"
SET.PSL.C(AMX)              "CCK/C_AMX0"
SET.V                       "CCK/SET.V"
SPEC                        "LAB_R(SP1),Q_IB.DATA,CLR.IB.COND,PC_PC+N,MCT/ALLOW.IB.READ,SUB/SPEC,J/C.FORK"
SPECG                       "LAB_R(SP1),Q_IB.DATA,CLR.IB.COND,PC_PC+N,MCT/ALLOW.IB.READ,SUB/SPEC,J/G.FORK"
START.IB                    "IBC/START"
STOP.IB                     "IBC/STOP"

TEST.TB.RCHK                "MCT/TEST.RCHK,VAK/NOP"
TEST.TB.WCHK                "MCT/TEST.WCHK,VAK/NOP"
TRAP.ACC[]                  "ACF/TRAP,ACM/@1"

WORD                        "DT/WORD"
WRITE.DEST                  "LAB_R(SP1),QK/ID,CLR.IB.COND,PC_PC+N,SUB/SPEC,J/WRD"
WRITE.G.DEST                "LAB_R(SP1), QK/ID,CLR.IB.COND,PC_PC+N,SUB/SPEC,J/WRG"
```

```
.TOC    "Macro definition      ; Branch enable macros"

AC.LOW?                         "BEN/INTERRUPT"  !,J3/3"
ACC.SYNC?                       "BEN/ACCEL"      !,J3/3"
ACCEL?                          "BEN/ACCEL"
ALIGNED?                        "BEN/TB.TEST"    !,J5/17"
ALU.N?                          "BEN/ALU"        !,J4/07"
ALU1-0?                         "BEN/ALU1-0"
ALU?                            "BEN/ALU"

BCDSGN?                         "BEN/DECIMAL"    !,J2/2"

C31?                            "BEN/C31"
CONSOLE.MODE?                   "BEN/PSL.MODE"   !,J5/1B"

D(1)?                           "BEN/MUL"
D.B0?                           "BEN/D.BYTES"    !,J4/0E"
D.B1?                           "BEN/D.BYTES"    !,J4/0D"
D.B2?                           "BEN/D.BYTES"    !,J4/0B"
D.BYTES?                        "BEN/D.BYTES"
D.NE.0?                         "BEN/SIGNS"      !,J3/5" !PREFERED FORM
D0?                             "BEN/D3-0"       !,J4/0E"
D2-0?                           "BEN/D3-0"       !,J4/08"
D2?                             "BEN/D3-0"       !,J4/0B"
D3-0?                           "BEN/D3-0"
D31?                            "BEN/SIGNS"      !,J3/6"
D3?                             "BEN/D3-0"       !,J4/07"
DATA.TYPE?                      "BEN/DATA.TYPE"
DBL?                            "BEN/DATA.TYPE"

EALU.N?                         "BEN/EALU"       !,J4/07"
EALU.Z?                         "BEN/EALU"       !,J4/0B"
EALU?                           "BEN/EALU"
END.DP1?                        "BEN/END.DP1"

FPD?                            "BEN/LAST.REF"   !,J4/07"

IB.TEST?                        "BEN/IB.TEST"
INT?                            "BEN/INTERRUPT"
INTERRUPT.REQ?                  "BEN/INTERRUPT"  !,J3/5"
IR0.C31?                        "BEN/ALU"
IR0?                            "BEN/ALU"        !,J4/0D"
IR1?                            "BEN/IR2-1"      !,J3/6"
IR2-1?                          "BEN/IR2-1"

LAST.REF?                       "BEN/LAST.REF"

MODE.LSS.ASTLVL?                "BEN/REI"        !,J3/3"
MUL?                            "BEN/MUL"

NEST.ERR?                       "BEN/LAST.REF"   !,J4/0B"

PC.MODES?                       "BEN/PC.MODES"
PSL.C?                          "BEN/PSL.CC"     !,J4/0E"
PSL.CC?                         "BEN/PSL.CC"
PSL.MODE?                       "BEN/PSL.MODE"
PSL.N?                          "BEN/PSL.CC"     !,J4/7"
PSL.V?                          "BEN/PSL.CC"     !,J4/0D"
PSL.Z?                          "BEN/PSL.CC"     !,J4/0B"
PTE.VALID?                      "BEN/TB.TEST"    !,J5/0F"

Q31?                            "BEN/SIGNS"      !,J3/3"
QUAD?                           "BEN/DATA.TYPE"
```

```
RLOG.EMPTY?                     "BEN/ALU1-0"    #,J4/7"
ROR?                            "BEN/ROR"

SC.GT.0?                        "BEN/SC"
SC.NE.0?                        "BEN/MUL"       #,J3/3"
SC?                             "BEN/SC"
SIGNS?                          "BEN/SIGNS"
SRC.PC?                         "BEN/SRC.PC"    #COMP MODE, BEN ON SRC R = PC
SS?                             "BEN/EALU"      #,J4/0E"
STATE(7)?                       "STATE7-4?"
STATE0?                         "BEN/STATE3-0"  #,J4/0E"
STATE1-0?                       "BEN/STATE3-0"  #,J4/0C"
STATE1?                         "BEN/STATE3-0"  #,J4/0D"
STATE2?                         "BEN/STATE3-0"  #,J4/0B"
STATE3-0?                       "BEN/STATE3-0"
STATE3?                         "BEN/STATE3-0"  #,J4/07"
STATE4?                         "BEN/STATE7-4"
STATE5?                         "BEN/STATE7-4"
STATE6?                         "BEN/STATE7-4"
STATE7-4?                       "BEN/STATE7-4"

TB.TEST?                        "BEN/TB.TEST"

VA31-30?                        "BEN/PSL.MODE"  #,J5/07"
VA31?                           "BEN/PSL.MODE"  #,J5/0F"

Z?                              "BEN/Z"
ZONED?                          "BEN/DECIMAL"   #,J2/1"

.BIN            #MAKE LISTING ROOM FOR BINARY FROM HERE ON
```

APPENDIX B

SAMPLE MICROPROGRAM FOR SYSTEM REVISION ≥ 7

This appendix contains a sample VAX 11/780 microprogram, which performs an unsigned binary search on a vector of longwords in main memory. The parameters of the routine, the value to be searched for and the beginning and end of the vector, are passed in registers.

A command file that assembles, loads, and executes this sample microprogram is provided in the VAX 11/780 WCS kit. To invoke this file in the VMS environment, type:

```
@[SYSEXE]WCSTOLTST
```

This command file assembles the input listing (Section B.1) and produces the listing file (Section B.2) and the object file (Section B.3) which are written to [VAXWCSTOL]SAMPLE.MCR and [VAXWCSTOL]SAMPLE.ULD. It then loads the object file into the extended WCS and runs the test program BSTEST (Appendix D). BSTEST executes an XFC instruction, which causes the sample microprogram loaded in the WCS to be executed. If the microprogram executes properly, BSTEST prints the following message on the terminal:

```
"Successful Test Completion"
```

B.1 THE INPUT FILE (.MIC)

```
        .TOC "Binary search routine"
        .REGION /1C00,1FFF              ;User wcs space.
        .BOUNDS/BSERCH:1C00,1FFF        ;This defines the report boundries
                                        ;for the U-code microword summary page
                                        ;and names the report boundary BSERCH.

; Sample microcode to perform an unsigned binary search through
; a vector of aligned longwords in main memory.
;
; INPUTS
;       R0 - Search comparand.  Routine succeeds by finding a
;               memory cell containing same data as R0.
;       R1 - Lower address bound.  Aligned longword address of
;               lowest address of vector to be searched.
;       R2 - Upper address bound.  Aligned longword address of
;               highest address of vector to be searched.
; It is implied that R1 lssu R2, and that the memory between the
; addresses in R1 and R2 contains a sorted vector, in ascending
; unsigned order.
;
; Outputs if search finds a match.
;       CC<Z> - Clear
;       R0    - Search comparand.
;       R1    - Match address.  Address of longword containing same data as R0.
;       R2    - Used by search for temporary address values.
;
; Outputs if search does not find a match.
;       CC<Z> - Set
;       R0    - Search comparand.
;       R1    - Used by search for temporary address values.
;       R2    - Used by search for temporary address values.
;
```

```
SRCH:   ;-----------------------;
        Q_R[R2],                ;GET UPPER BOUND ADDR TO Q
        STATE_K[ZERO]           ;INITIALIZE STATE REGISTER

        ;-----------------------;
        D_R[R0]                 ;GET COMPARAND TO HOLD IN RC

        ;-----------------------;
        ALU_D,                  ;PREPARE TO WRITE COMPARAND TO RC
        LAB_R1&RC[T1]_ALU       ;WRITE COMPARAND, GET LOWER BOUND


SRCH.1: ;-----------------------;
        Q_(LA+Q).RIGHT,         ;COMPUTE MIDPOINT ADDRESS
        INTRPT.STROBE,          ;TEST FOR INTERRUPT REQUESTS
        STATE0?                 ;IS IT TIME TO STOP?

=0
SRCH.2: ;0---------------------;STATE0=0.  KEEP LOOKING FOR MATCH.
        Q_Q.ANDNOT.K[.3],       ;FORCE LONGWORD ALIGNMENT
        VA_ALU,                 ;GET READY TO READY MIDPOINT OF VECTOR
        LC_RC[T1],              ;LATCH COMPARAND INTO LC
        INT?,J/SRCH.3           ;IS THERE AN INTERRUPT REQUEST?

        ;1---------------------;STATE0=1. SEARCH FAILED.  NO MATCH.
        ALU_K[ZERO],            ;
        CCK/NZ_ALU.VC_0,LONG,   ;RETURN Z=1 TO FLAG FAILURE.
        CLR.IB.OPC,PC_PC+1,     ;MOVE ON TO THE NEXT INSTRUCTION
        J/IRD                   ;

=110
SRCH.3: ;110-------------------;NO INTERRUPT REQUESTS
        D[LONG]_CACHE,          ;READY MIDPOINT ENTRY OF VECTOR
        ALU_R[R2].XOR.Q,        ;COMPARE MIDPOINT EQL UPPER BOUND
        CLK.UBCC,J/SRCH.4       ;

        ;111-------------------;INTERRUPT REQUEST IS UP
        J/INT.B                 ;TAKE IT.  RESUME FROM REG'S AS IS.
```

```
; WE HAVE ALSO SET THE MICROBRACH Z BIT ACCORDING TO A COMPARE OF
; THE MEMORY ADDRESS WITH THE CURRENT UPPER BOUND.  IF THEY ARE
; EQUAL, THIS IS THE LAST POSSIBLE COMPARISON.  A MATCH FAILURE
; HERE IMPLIES THAT THERE IS NO MATCH TO BE FOUND.

SRCH.4: ;-----------------------;
        ALU_D-LC,               ;COMPARE MEMORY TO COMPARAND
        LONG,CLK.UBCC,          ;RECORD COMPARE RESULT
        LA_RA[R1],              ;LATCH LOWER BOUND INTO LA (LB HAS ???
        Z?                      ;IS MIDPOINT EQL UPPER BOUND?

=0      ;0----------------------;ALU Z=0.  NOT END OF SEARCH
        ALU?,J/SRCH.5           ;TEST RESULT OF COMPARE

        ;1----------------------;ALU Z=1.  END OF SEARCH
        STATE_K[.1],            ;SET STATE0 TO MARK END OF SEARCH.
        ALU?                    ;CHECK FOR LAST CHANCE MATCH

=1010
SRCH.5: ;1010-------------------;ALU Z=0, C=1.  R0 GTRU MEM
        Q_Q+K[.4],              ;LOWER LIMIT MUST BE GREATER THAN THIS
        R[R1]_ALU,              ;REMEMBER IN R1.
        J/SRCH.6                ;

        ;1011-------------------;ALU Z=0, C=0.  R0 LSSU MEM
        Q_Q-K[.4],              ;UPPER LIMIT MUST BE LESS THAN THIS
        R[R2]_ALU,              ;REMEMBER IN R2
        J/SRCH.1                ;GO TRY AGAIN

=1111   ;1111-------------------;ALU Z=1, C=1.  R0 EQL MEM
        R[R1]_Q,                ;FOUND IT!
        CCK/NZ_ALU,VC_0,LONG,   ;SET Z=0 TO INDICATE MATCH
        CLR.IB.OPC,PC_PC+1,     ;GO TO NEXT INSTRUCTION
        J/IRD

SRCH.6: ;-----------------------;
        Q_(Q+LB).RIGHT,         ;COMPUTE NEW MIDPOINT, LOOP
        INTRPT.STROBE,          ;
        STATE0?,J/SRCH.2        ;CHECK FOR END, LOOP

; DEFINE LABLES TO INTERFACE WITH PCS
0062:   IRD:
04F8:   INT.B:
```

B.2 THE LISTING FILE (.MCR)

```
$ NEWSAM.MCR                MICRO2  1L(02)    18-JAN-82  16:15:06                Page    1
$                           Table of Contents

$ 2     Machine definition    : Control word chart
$ 56    Machine definition    : ACF, ACM, ADS, ALU, AMX
$ 97    Machine definition    : BEN, BMX
$ 150   Machine definition    : CCK, CID, DK, DT
$ 205   Machine definition    : EALU, EBMX, FEK, FS, IEK, IBC
$ 255   Machine definition    : ID.ADDR, J
$ 330   Machine definition    : KMX
$ 405   Machine definition    : MCT, MSC
$ 452   Machine definition    : PCK, QK, RAMX, RBMX
$ 487   Machine definition    : SCK, SGN, SHF, SI, SMX
$ 529   Machine definition    : SPO, SPO.AC, SPO.ACN, SPO.ACN11, SPO.R
$ 568   Machine definition    : SPO.RAB, SPO.RC, SUB, VAK
$ 617   Machine definition    : Validity checks
$ 624   Macro definition      : Register transfer macros
$ 1538  Macro definition      : Non-transfer macros
$ 1634  Macro definition      : Branch enable macros
$ 1729  Binary search routine
```

```
; NEWSAM.MCR        MICRO2  1L(02)     18-JAN-82  16:15:06                                  Page     2
; VAXDEF.MIC

                                ;1      .NOLIST                 ;Inhibit listing for VAXDEF.MIC
                                ;1728   .LIST
```

```
                        ;1729           .TOC "Binary search routine"
                        ;1730           .REGION /1C00,1FFF              ;User wcs space.
                        ;1731           .BOUNDS/BSERCH:1C00,1FFF        ;This defines the report boundries
                        ;1732                                           ;for the U-code microword summary page
                        ;1733                                           ;and names the report boundary BSERCH.
                        ;1734
                        ;1735   ; Sample microcode to perform an unsigned binary search through
                        ;1736   ; a vector of aligned longwords in main memory.
                        ;1737   ;
                        ;1738   ; INPUTS
                        ;1739   ;       R0 - Search comparand.  Routine succeeds by finding a
                        ;1740   ;               memory cell containing same data as R0.
                        ;1741   ;       R1 - Lower address bound.  Aligned longword address of
                        ;1742   ;               lowest address of vector to be searched.
                        ;1743   ;       R2 - Upper address bound.  Aligned longword address of
                        ;1744   ;               highest address of vector to be searched.
                        ;1745   ; It is implied that R1 lssu R2, and that the memory between the
                        ;1746   ; addresses in R1 and R2 contains a sorted vector, in ascending
                        ;1747   ; unsigned order.
                        ;1748   ;
                        ;1749   ; Outputs if search finds a match.
                        ;1750   ;       CC<Z> - Clear
                        ;1751   ;       R0    - Search comparand.
                        ;1752   ;       R1    - Match address.  Address of longword containing same data as R0.
                        ;1753   ;       R2    - Used by search for temporary address values.
                        ;1754   ;
                        ;1755   ; Outputs if search does not find a match.
                        ;1756   ;       CC<Z> - Set
                        ;1757   ;       R0    - Search comparand.
                        ;1758   ;       R1    - Used by search for temporary address values.
                        ;1759   ;       R2    - Used by search for temporary address values.
                        ;1760   ;
```

```
; NEWSAM.MCR              MICRO2  1L(02)    18-JAN-82  16:15:06                    Page  40
; BSERCH.MIC              Binary search routine

                                    ;1761
                                    ;1762   SRCH:   ;----------------------;
                                    ;1763           Q_RCR2],               ;GET UPPER BOUND ADDR TO Q
U 1C04, 0000,003C,19C0,FA10,1404,7C05   ;1764       STATE_KCZERO]          ;INITIALIZE STATE REGISTER
                                    ;1765
                                    ;1766           ;----------------------;
U 1C05, 0800,003C,0180,FA00,0000,1C08   ;1767       D_RCRO]                ;GET COMPARAND TO HOLD IN RC
                                    ;1768
                                    ;1769           ;----------------------;
                                    ;1770           ALU_D,                 ;PREPARE TO WRITE COMPARAND TO RC
U 1C08, 0001,003C,0180,FB08,0000,1C09   ;1771       LAB_R1&RCCT1]_ALU      ;WRITE COMPARAND, GET LOWER BOUND
                                    ;1772
                                    ;1773
                                    ;1774   SRCH.1: ;----------------------;
                                    ;1775           Q_(LA+Q).RIGHT,        ;COMPUTE MIDPOINT ADDRESS
                                    ;1776           INTRPT.STROBE,         ;TEST FOR INTERRUPT REQUESTS
U 1C09, 005C,1714,01C0,F800,4000,1C00   ;1777       STATE0?                ;IS IT TIME TO STOP?
                                    ;1778
                                    ;1779   =0
                                    ;1780   SRCH.2: ;0---------------------;STATE0=0.  KEEP LOOKING FOR MATCH.
                                    ;1781           Q_Q.ANDNOT.KC.3],      ;FORCE LONGWORD ALIGNMENT
                                    ;1782           VA_ALU,                ;GET READY TO READY MIDPOINT OF VECTOR
                                    ;1783           LC_RCCT1],             ;LATCH COMPARAND INTO LC
U 1C00, 0019,2E24,0DC0,F908,0200,1C06   ;1784       INT?,J/SRCH.3          ;IS THERE AN INTERRUPT REQUEST?
                                    ;1785
                                    ;1786           ;1---------------------;STATE0=1. SEARCH FAILED.  NO MATCH.
                                    ;1787           ALU_KCZERO],           ;
                                    ;1788           CCK/NZ_ALU.VC_0,LONG,  ;RETURN Z=1 TO FLAG FAILURE.
                                    ;1789           CLR.IB.OPC,PC_PC+1,    ;MOVE ON TO THE NEXT INSTRUCTION
U 1C01, C018,0038,1980,F804,4050,0062   ;1790       J/IRD                  ;
                                    ;1791
                                    ;1792   =110
                                    ;1793   SRCH.3: ;110-------------------;NO INTERRUPT REQUESTS
                                    ;1794           DCLONG]_CACHE,         ;READY MIDPOINT ENTRY OF VECTOR
                                    ;1795           ALU_RCR2].XOR.Q,       ;COMPARE MIDPOINT EQL UPPER BOUND
U 1C06, 001C,0020,0180,4210,0010,1C0C   ;1796       CLK.UBCC,J/SRCH.4      ;
                                    ;1797
                                    ;1798           ;111-------------------;INTERRUPT REQUEST IS UP
U 1C07, 0000,003C,0180,F800,0000,04F8   ;1799       J/INT.B                ;TAKE IT.  RESUME FROM REG'S AS IS.
```

```
; NEWSAM.MCR              MICRO2  1L(02)    18-JAN-82  16:15:06                    Page   41
; BSERCH.MIC              Binary search routine

                                    #1800   ; WE HAVE ALSO SET THE MICROBRACH Z BIT ACCORDING TO A COMPARE OF
                                    #1801   ; THE MEMORY ADDRESS WITH THE CURRENT UPPER BOUND.  IF THEY ARE
                                    #1802   ; EQUAL, THIS IS THE LAST POSSIBLE COMPARISON.  A MATCH FAILURE
                                    #1803   ; HERE IMPLIES THAT THERE IS NO MATCH TO BE FOUND.
                                    #1804
                                    #1805   SRCH.4: ;-----------------------;
                                    #1806           ALU_D-LC,               ;COMPARE MEMORY TO COMPARAND
                                    #1807           LONG,CLK,UBCC,          ;RECORD COMPARE RESULT
                                    #1808           LA_RA[R1],              ;LATCH LOWER BOUND INTO LA (LB HAS ???
U 1C0C, 0011,0100,0180,F888,0010,1C02   #1809           Z?                      ;IS MIDPOINT EQL UPPER BOUND?
                                    #1810
                                    #1811   =0      ;0----------------------;ALU Z=0.  NOT END OF SEARCH
U 1C02, 0000,1B3C,0180,F800,0000,1C0A   #1812           ALU?,J/SRCH.5           ;TEST RESULT OF COMPARE
                                    #1813
                                    #1814           ;1----------------------;ALU Z=1.  END OF SEARCH
                                    #1815           STATE_K[.1],            ;SET STATE0 TO MARK END OF SEARCH.
U 1C03, 0000,1B3C,0580,F800,1404,7C0A   #1816           ALU?                    ;CHECK FOR LAST CHANCE MATCH
                                    #1817
                                    #1818   =1010
                                    #1819   SRCH.5: ;1010--------------------;ALU Z=0, C=1.  R0 GTRU MEM
                                    #1820           Q_Q+K[.4],              ;LOWER LIMIT MUST BE GREATER THAN THIS
                                    #1821           R[R1]_ALU,              ;REMEMBER IN R1.
U 1C0A, 0019,2014,11C0,FA88,0000,1C0D   #1822           J/SRCH.6                ;
                                    #1823
                                    #1824           ;1011--------------------;ALU Z=0, C=0.  R0 LSSU MEM
                                    #1825           Q_Q-K[.4],              ;UPPER LIMIT MUST BE LESS THAN THIS
                                    #1826           R[R2]_ALU,              ;REMEMBER IN R2
U 1C0B, 0019,2000,11C0,FA90,0000,1C09   #1827           J/SRCH.1                ;GO TRY AGAIN
                                    #1828
                                    #1829   =1111    ;1111--------------------;ALU Z=1, C=1.  R0 EQL MEM
                                    #1830           R[R1]_Q,                ;FOUND IT!
                                    #1831           CCK/NZ_ALU.VC_0,LONG,   ;SET Z=0 TO INDICATE MATCH
                                    #1832           CLR.IB.OPC,PC_PC+1,     ;GO TO NEXT INSTRUCTION
U 1C0F, C001,203C,0180,FA8C,4050,0062   #1833           J/IRD
                                    #1834
                                    #1835   SRCH.6: ;-----------------------;
                                    #1836           Q_(Q+LB).RIGHT,         ;COMPUTE NEW MIDPOINT, LOOP
                                    #1837           INTRPT.STROBE,          ;
U 1C0D, 004D,3714,01C0,F800,4000,1C00   #1838           STATE0?,J/SRCH.2        ;CHECK FOR END, LOOP
                                    #1839
                                    #1840   ; DEFINE LABLES TO INTERFACE WITH PCS
                                    #1841   0062:   IRD:
                                    #1842   04F8:   INT.B:
```

```
# NEWSAM.MCR              MICRO2  1L(02)    18-JAN-82  16:15:06                    Page   42
#                         Cross Reference Listing - Field Names and Defined Values

J                          326 #
        INT.B              1799    1842 #
        IRD                1790    1833    1841 #
        SRCH               1762 #
        SRCH.1             1774 #  1827
        SRCH.2             1780 #  1838
        SRCH.3             1784    1793 #
        SRCH.4             1796    1805 #
        SRCH.5             1812    1819 #
        SRCH.6             1822    1835 #
```

```
; NEWSAM.MCR              MICRO2  1L(02)    18-JAN-82  16:15:06          Page  43
;                         Cross Reference Listing - Macro Names

AC.LOW?                    1636 #
ACC.SYNC?                  1637 #
ACCEL?                     1638 #
ALIGNED?                   1639 #
ALU.N?                     1640 #
ALU1-0?                    1641 #
ALU?                       1642 #  1812     1816
ALU_-1                     626 #
ALU_0(A)                   627 #
ALU_0+D                    628 #
ALU_0+D+1                  629 #
ALU_0+KC]                  630 #
ALU_0+KC]+1                631 #
ALU_0+LB+1                 632 #
ALU_0+LC                   633 #
ALU_0+LC+1                 634 #
ALU_0+MASK+1               635 #
ALU_0+Q                    636 #
ALU_0+Q+1                  637 #
ALU_0-D                    638 #
ALU_0-D-1                  639 #
ALU_0-KC]                  640 #
ALU_0-KC]-1                641 #
ALU_0-LB                   642 #
ALU_0-LC                   643 #
ALU_0-LC-1                 644 #
ALU_0-Q                    645 #
ALU_0-Q-1                  646 #
ALU_0C]D                   647 #
ALU_0C]LC                  648 #
ALU_D                      649 #   1770
ALU_D(B)                   650 #
ALU_D+KC]                  651 #
ALU_D+KC]+1                652 #
ALU_D+KC].RLOG             653 #
ALU_D+LB                   654 #
ALU_D+LC                   655 #
ALU_D+LC+1                 656 #
ALU_D+LC+PSL.C             657 #
ALU_D+Q                    658 #
ALU_D+Q+1                  659 #
ALU_D+Q+PSL.C              660 #
ALU_D+RLOG                 661 #
ALU_D-KC]                  662 #
ALU_D-KC]-1                663 #
ALU_D-LB                   664 #
ALU_D-LB.RLOG              665 #
ALU_D-LC                   666 #   1806
ALU_D-LC-1                 667 #
ALU_D-Q                    668 #
ALU_D-Q-1                  669 #
ALU_D.OXTC]                670 #
ALU_D.OXTC]+KC]            671 #
ALU_D.OXTC]+LC             672 #
ALU_D.OXTC]+Q              673 #
```

```
# NEWSAM.MCR              MICRO2  1L(02)    18-JAN-82  16:15:06                    Page   44
#                         Cross Reference Listing - Macro Names

ALU_D.OXTC]-KC]                674 #
ALU_D.OXTC]-Q                  675 #
ALU_D.OXTC].AND.KC]            676 #
ALU_D.OXTC].ANDNOT.KC]         677 #
ALU_D.OXTC].OR.Q               678 #
ALU_D.AND.KC]                  679 #
ALU_D.AND.MASK                 680 #
ALU_D.ANDNOT.KC]               681 #
ALU_D.ANDNOT.MASK              682 #
ALU_D.ANDNOT.Q                 683 #
ALU_D.OR.KC]                   684 #
ALU_D.OR.LC                    685 #
ALU_D.OR.Q                     686 #
ALU_D.OR.RCC]                  687 #
ALU_D.ORNOT.MASK               688 #
ALU_D.SXTC]                    689 #
ALU_D.SXTC]+KC]                690 #
ALU_D.SXTC]+Q                  691 #
ALU_D.SXTC].AND.KC]            693 #
ALU_D.SXTC].ANDNOT.KC]         692 #
ALU_D.XOR.KC]                  694 #
ALU_D.XOR.LC                   695 #
ALU_D.XOR.Q                    696 #
ALU_D.XOR.RCC]                 697 #
ALU_D.XOR.RC]                  698 #
ALU_DC]KC]                     699 #
ALU_DC]LB                      700 #
ALU_DC]LC                      701 #
ALU_DC]Q                       702 #
ALU_KC]                        703 #    1787
ALU_LA                         704 #
ALU_LA+KC]                     705 #
ALU_LA+KC]+1                   706 #
ALU_LA+KC].RLOG                707 #
ALU_LA+LB                      708 #
ALU_LA+LC                      709 #
ALU_LA+LC+1                    710 #
ALU_LA+LC+PSL.C                711 #
ALU_LA+Q                       712 #
ALU_LA-D                       713 #
ALU_LA-D-1                     714 #
ALU_LA-KC]                     715 #
ALU_LA-KC]-1                   716 #
ALU_LA-KC].RLOG                717 #
ALU_LA-LC                      718 #
ALU_LA-Q                       719 #
ALU_LA-Q-1                     720 #
ALU_LA.AND.KC]                 721 #
ALU_LA.AND.LC                  722 #
ALU_LA.ANDNOT.KC]              723 #
ALU_LA.ANDNOT.MASK             724 #
ALU_LA.OR.KC]                  725 #
ALU_LA.XOR.LC                  726 #
ALU_LAC]D                      727 #
ALU_LAC]LB                     728 #
```

```
# NEWSAM.MCR                MICRO2  1L(02)    18-JAN-82  16:15:06                    Page   47
#                           Cross Reference Listing - Macro Names

D.B2?                          1652 #
D.BYTES?                       1653 #
D.NE.0?                        1654 #
D0?                            1655 #
D2-0?                          1656 #
D2?                            1657 #
D3-0?                          1658 #
D31?                           1659 #
D3?                            1660 #
DATA.TYPE?                     1661 #
DBL?                           1662 #
D[]_CACHE                      817 #   1794
D[]_CACHE.IBCHK                818 #
D[]_CACHE.LK                   819 #
D[]_CACHE.NOCHK                820 #
D[]_CACHE.P                    821 #
D[]_CACHE.WCHK                 822 #
D_0                            824 #
D_0+K[]+1                      825 #
D_0+LC+1                       826 #
D_0-D                          827 #
D_0-K[]                        828 #
D_0-Q                          829 #
D_0-Q-1                        830 #
D_ACCEL&SYNC                   831 #
D_ALU                          832 #
D_ALU(FRAC)                    833 #
D_ALU.LEFT                     834 #
D_ALU.LEFT2                    835 #
D_ALU.LEFT3                    836 #
D_ALU.RIGHT                    837 #
D_ALU.RIGHT2                   838 #
D_BLANK                        839 #
D_CACHE.INST.DEP               840 #
D_CACHE.LK[]                   841 #
D_CACHE.WCHK[]                 842 #
D_CACHE[]                      843 #
D_D(FRAC)                      844 #
D_D+K[]                        845 #
D_D+K[]+1                      846 #
D_D+LB                         847 #
D_D+LC                         848 #
D_D+LC+PSL.C                   849 #
D_D+Q                          850 #
D_D+Q+1                        851 #
D_D-K[]                        852 #
D_D-LC                         853 #
D_D-Q                          854 #
D_D-Q-1                        855 #
D_D.OXT[]                      856 #
D_D.OXT[]+K[]                  857 #
D_D.OXT[]+Q                    858 #
D_D.OXT[]+Q+1                  859 #
D_D.OXT[].ANDNOT.K[]           860 #
D_D.OXT[].OR.Q                 861 #
```

Cross Reference Listing - Macro Names

D_D.OXT[].XOR.Q	862	#
D_D.OXT[].XOR.RC[]	863	#
D_D.AND.K[]	864	#
D_D.AND.K[].LEFT2	865	#
D_D.AND.K[].RIGHT	866	#
D_D.AND.LC	867	#
D_D.AND.MASK	868	#
D_D.AND.Q	869	#
D_D.AND.RC[]	870	#
D_D.ANDNOT.K[]	871	#
D_D.ANDNOT.LC	872	#
D_D.ANDNOT.PSWZ	873	#
D_D.ANDNOT.Q	874	#
D_D.ANDNOT.RC[]	875	#
D_D.LEFT	876	#
D_D.LEFT2	877	#
D_D.OR.ASCII	878	#
D_D.OR.K[]	879	#
D_D.OR.PSWC	880	#
D_D.OR.PSWV	881	#
D_D.OR.Q	882	#
D_D.OR.RC[]	883	#
D_D.OR.R[]	884	#
D_D.ORNOT.MASK	885	#
D_D.RIGHT	886	#
D_D.RIGHT(B)	887	#
D_D.RIGHT2	888	#
D_D.SWAP	889	#
D_D.SXT[]	890	#
D_D.SXT[].RIGHT	891	#
D_D.XOR.K[]	892	#
D_D.XOR.LC	893	#
D_D.XOR.Q	894	#
D_DAL.NORM	895	#
D_DAL.SC	896	#
D_D[]K[]	897	#
D_D[]MASK	898	#
D_D[]Q	899	#
D_INT.SUM	900	#
D_K[]	901	#
D_K[].RIGHT	902	#
D_K[].RIGHT2	903	#
D_LA	904	#
D_LA(FRAC)	905	#
D_LA+D+PSL.C	906	#
D_LA-D	907	#
D_LA-K[]	908	#
D_LA.AND.K[]	909	#
D_LA.RIGHT	910	#
D_LB	911	#
D_LB.PC	912	#
D_LC	913	#
D_LC(FRAC)	914	#
D_NOT.D	915	#
D_NOT.K[]	916	#

```
; NEWSAM.MCR              MICRO2  1L(02)    18-JAN-82  16:15:06                    Page  49
;                         Cross Reference Listing - Macro Names

D_NOT.MASK                 917 #
D_NOT.Q                    918 #
D_NOT.RC]                  919 #
D_PACK.FP                  920 #
D_PACK.FP.LEFT             921 #
D_PC                       922 #
D_PC.LEFT                  923 #
D_Q                        924 #
D_Q(FRAC)                  925 #
D_Q+D                      926 #
D_Q+KC]                    927 #
D_Q+LB                     928 #
D_Q+PC                     929 #
D_Q-D                      930 #
D_Q-D-1                    931 #
D_Q-KC]                    932 #
D_Q-KC]-1                  933 #
D_Q-PCSV                   934 #
D_Q.0XTC]                  935 #
D_Q.AND.KC]                936 #
D_Q.AND.LC                 937 #
D_Q.AND.MASK               938 #
D_Q.AND.RC[]               939 #
D_Q.ANDNOT.D               940 #
D_Q.ANDNOT.KC]             941 #
D_Q.ANDNOT.MASK            942 #
D_Q.ANDNOT.PSWC            943 #
D_Q.ANDNOT.PSWN            944 #
D_Q.ANDNOT.PSWZ            945 #
D_Q.LEFT                   946 #
D_Q.OR.KC]                 947 #
D_Q.OR.PSWC                948 #
D_Q.OR.RC[]                949 #
D_Q.ORNOT.MASK             950 #
D_Q.RIGHT                  951 #
D_Q.RIGHT2                 952 #
D_Q.SXTC]                  953 #
D_Q.XOR.RC[]               954 #
D_QC]D                     955 #
D_QC]KC]                   956 #
D_QC]MASK                  957 #
D_R(PRN+1)                 958 #
D_R(SC)                    959 #
D_R(SP1+1)                 960 #
D_RC(SC)                   961 #
D_RC[]                     962 #
D_RLOG                     963 #
D_RLOG.RIGHT               964 #
D_R[]                      965 #   1767
D_R[](FRAC)                966 #
D_R[].AND.KC]              967 #
D_R[].OR.KC]               968 #
D_R[].ORNOT.KC]            969 #
E.FORK                    1562 #
EALU.N?                   1664 #
```

```
EALU.Z?                          1665 #
EALU?                            1666 #
EALU_D(EXP)                      971 #
EALU_FE                          972 #
EALU_K[]                         973 #
EALU_R[](EXP)                    974 #
EALU_SC                          975 #
EALU_SC+FE                       976 #
EALU_SC+K[]                      977 #
EALU_SC-FE                       978 #
EALU_SC-K[]                      979 #
EALU_SC.ANDNOT.K[]               980 #
EALU_STATE                       981 #
END.DP1?                         1667 #
EXCEPT.ACK                       1563 #
FE&SC_K[]                        983 #
FE_0(A)                          984 #
FE_D(EXP)                        985 #
FE_EALU                          986 #
FE_K[]                           987 #
FE_LA(EXP)                       988 #
FE_NABS(SC-FE)                   989 #
FE_NABS(SC-LA(EXP))              990 #
FE_Q(EXP)                        991 #
FE_R[](EXP)                      992 #
FE_SC                            993 #
FE_SC+1                          994 #
FE_SC+FE                         995 #
FE_SC+K[]                        996 #
FE_SC+LA(EXP)                    997 #
FE_SC-FE                         998 #
FE_SC-K[]                        999 #
FE_SC-LA(EXP)                    1000 #
FE_SC-SHF.VAL                    1001 #
FE_SC.ANDNOT.FE                  1002 #
FE_SC.ANDNOT.K[]                 1003 #
FE_SC.OR.K[]                     1004 #
FE_SHF.VAL                       1005 #
FE_STATE                         1006 #
FLUSH.IB                         1565 #
FPD?                             1669 #
G.FORK                           1567 #
IB.TEST?                         1671 #
ID(SC)_D                         1008 #
ID[]_D                           1009 #
ID_D&NO.SYNC                     1010 #
ID_D.SYNC                        1011 #
INHIBIT.IB                       1569 #
INT?                             1672 #  1784
INTERRUPT.REQ?                   1673 #
INTRPT.ACK                       1570 #
INTRPT.STROBE                    1571 #  1776    1837
IR0.C31?                         1674 #
IR0?                             1675 #
IR1?                             1676 #
```

```
IR2-1?                                1677 #
IRD                                   1572 #
IRD.11                                1573 #
IRD0                                  1574 #
IRD1                                  1575 #
KC]                                   1013 #
LAB_R(DST)                            1015 #
LAB_R(PRN)                            1016 #
LAB_R(PRN+1)                          1017 #
LAB_R(SC)                             1018 #
LAB_R(SP1)                            1019 #
LAB_R(SP1+1)                          1020 #
LAB_R1&RC[]_0                         1021 #
LAB_R1&RC[]_0+LC+1                    1022 #
LAB_R1&RC[]_0-D                       1023 #
LAB_R1&RC[]_ALU                       1024 #  1771
LAB_R1&RC[]_ALU.RIGHT2                1025 #
LAB_R1&RC[]_D+LC                      1026 #
LAB_R1&RC[]_D.0XT[]+KC]               1027 #
LAB_R1&RC[]_Q-KC]                     1028 #
LAB_RC]                               1029 #
LAST.REF?                             1679 #
LA_R(DST)&LB_R(SRC)                   1031 #
LA_R(SP2)&LB_R(SP1)                   1032 #
LA_RA[]                               1033 #  1808
LC_RC(SC)                             1034 #
LC_RC[]                               1035 #  1783
LC_RC[]&R1_(LA+LB).LEFT               1036 #
LC_RC[]&R1_(LA+LB+PSL.C).LEFT         1037 #
LC_RC[]&R1_(LA+LB.RLOG).LEFT          1038 #
LC_RC[]&R1_(LA-LB).LEFT               1039 #
LC_RC[]&R1_(LA-LB.RLOG).LEFT          1040 #
LC_RC[]&R1_ALU                        1041 #
LC_RC[]&R1_D                          1042 #
LC_RC[]&R1_LA+KC]                     1043 #
LC_RC[]&R1_LA-KC]                     1044 #
LC_RC[]&R1_LB                         1045 #
LC_RC[]&R1_Q                          1046 #
LOAD.ACC.CC                           1577 #
LOAD.IB                               1578 #
LOAD.IB.11                            1579 #
LONG                                  1580 #  1788    1807    1831
MEMORY.NOP                            1582 #
MODE.LSS.ASTLVL?                      1681 #
MUL.0XT                               1583 #
MUL.1XT                               1584 #
MUL?                                  1682 #
MULM.DONE                             1585 #
MULP.DONE                             1586 #
N&Z_ALU                               1048 #
N&Z_ALU.V&C_0                         1049 #
NEST.ERR?                             1684 #
N_AMX.Z_TST                           1050 #
PC&VA_ALU                             1052 #
PC&VA_D                               1053 #
```

Cross Reference Listing - Macro Names

```
PC&VA_D+K[]                     1054 #
PC&VA_D-K[]                     1055 #
PC&VA_D-PC                      1056 #
PC&VA_D.OXT[]                   1057 #
PC&VA_D.OXT[]+PC                1058 #
PC&VA_D.SXT[]+PC                1059 #
PC&VA_K[]                       1060 #
PC&VA_PC                        1061 #
PC&VA_Q                         1062 #
PC&VA_Q+PC                      1063 #
PC&VA_Q-D                       1064 #
PC&VA_Q-K[]                     1065 #
PC&VA_Q.SXT[]+PC                1066 #
PC&VA_RC[]                      1067 #
PC&VA_R[].ANDNOT.K[]            1068 #
PC.MODES?                       1686 #
PC_PC+1                         1070 #  1789    1832
PC_PC+2                         1071 #
PC_PC+4                         1072 #
PC_PC+N                         1073 #
PC_Q+PC                         1074 #
PC_VA                           1075 #
PC_VIBA                         1076 #
POLY.DONE                       1588 #
PSL.C?                          1687 #
PSL.CC?                         1688 #
PSL.MODE?                       1689 #
PSL.N?                          1690 #
PSL.V?                          1691 #
PSL.Z?                          1692 #
PSL<C>_AMXO                     1077 #
PTE.VALID?                      1693 #
Q&VA_ALU                        1079 #
Q&VA_D                          1080 #
Q&VA_D+LC                       1081 #
Q&VA_LA                         1082 #
Q&VA_Q+LB.PC                    1083 #
Q31?                            1695 #
QD_(Q+LB)D.RIGHT2               1085 #
QD_(Q+LC)D.RIGHT2               1086 #
QD_(Q-LB)D.RIGHT2               1087 #
QD_(Q-LC)D.RIGHT2               1088 #
QD_QD.RIGHT2                    1089 #
QUAD?                           1696 #
Q_(LA+Q).RIGHT                  1091 #  1775
Q_(Q+LB).RIGHT                  1092 #  1836
Q_O                             1093 #
Q_O+LC+1                        1094 #
Q_O+MASK+1                      1095 #
Q_O+PC.RLOG                     1096 #
Q_O-D                           1097 #
Q_O-K[]                         1098 #
Q_O-LC                          1099 #
Q_O-Q                           1100 #
Q_ACCEL&SYNC                    1101 #
```

```
; NEWSAM.MCR                MICRO2  1L(02)    18-JAN-82  16:15:06                Page   53
;                           Cross Reference Listing - Macro Names

Q_ALU                          1102 #
Q_ALU(FRAC)                    1103 #
Q_ALU.LEFT                     1104 #
Q_ALU.LEFT2                    1105 #
Q_ALU.LEFT3                    1106 #
Q_ALU.RIGHT                    1107 #
Q_ALU.RIGHT2                   1108 #
Q_D                            1109 #
Q_D(FRAC)(B)                   1110 #
Q_D+KC]                        1111 #
Q_D+KC]+1                      1112 #
Q_D+KC].LEFT                   1113 #
Q_D+LC                         1114 #
Q_D-KC]                        1115 #
Q_D-LC                         1116 #
Q_D-Q                          1117 #
Q_D.OXTC]                      1118 #
Q_D.OXTC]+KC].LEFT             1119 #
Q_D.OXTC].OR.PACK.FP           1120 #
Q_D.AND.KC]                    1121 #
Q_D.AND.KC].RIGHT              1122 #
Q_D.AND.KC].RIGHT2             1123 #
Q_D.AND.RCC]                   1124 #
Q_D.ANDNOT.RCC]                1125 #
Q_D.LEFT3                      1126 #
Q_D.OR.KC]                     1127 #
Q_D.OR.RCC]                    1128 #
Q_D.RIGHT                      1129 #
Q_D.RIGHT2                     1130 #
Q_D.SXTC]                      1131 #
Q_D.XOR.Q                      1132 #
Q_DEC.CON                      1133 #
Q_IB.BDEST                     1134 #
Q_IB.DATA                      1135 #
Q_ID(SC)                       1136 #
Q_IDC]                         1137 #
Q_KC]                          1138 #
Q_KC]+1                        1139 #
Q_KC].CTX                      1140 #
Q_KC].RIGHT                    1141 #
Q_KC].RIGHT2                   1142 #
Q_LA                           1143 #
Q_LA+KC]                       1144 #
Q_LA+Q                         1145 #
Q_LA-KC]                       1146 #
Q_LA.AND.KC]                   1147 #
Q_LA.ANDNOT.RCC]               1148 #
Q_LB                           1149 #
Q_LC                           1150 #
Q_NOT.Q                        1151 #
Q_NOT.RC]                      1152 #
Q_PACK.FP                      1153 #
Q_PC                           1154 #
Q_Q(FRAC)                      1155 #
Q_Q(FRAC)(B)                   1156 #
```

```
# NEWSAM.MCR                MICRO2  1L(02)    18-JAN-82  16:15:06                Page   54
#                           Cross Reference Listing - Macro Names

Q_Q+D                          1157 #
Q_Q+K[]                        1158 #  1820
Q_Q+K[]+1                      1159 #
Q_Q+LC                         1160 #
Q_Q+PC                         1161 #
Q_Q-D                          1162 #
Q_Q-D-1                        1163 #
Q_Q-K[]                        1164 #  1825
Q_Q-K[]-1                      1165 #
Q_Q-LC                         1166 #
Q_Q-LC-1                       1167 #
Q_Q-MASK-1                     1168 #
Q_Q.OXT[]-K[]                  1169 #
Q_Q.OXT[].LEFT                 1170 #
Q_Q.OXT[].OR.D                 1171 #
Q_Q.AND.K[]                    1172 #
Q_Q.AND.K[].RIGHT              1174 #
Q_Q.AND.K[].RIGHT2             1173 #
Q_Q.AND.RC[]                   1176 #
Q_Q.AND.R[]                    1175 #
Q_Q.ANDNOT.D                   1177 #
Q_Q.ANDNOT.K[]                 1178 #  1781
Q_Q.ANDNOT.RC[]                1179 #
Q_Q.LEFT                       1180 #
Q_Q.LEFT2                      1181 #
Q_Q.OR.K[]                     1182 #
Q_Q.ORNOT.MASK                 1183 #
Q_Q.RIGHT                      1184 #
Q_Q.RIGHT2                     1185 #
Q_Q.SXT[]                      1186 #
Q_Q.XOR.K[]                    1187 #
Q_R(PRN).ANDNOT.Q              1188 #
Q_R(PRN+1)                     1189 #
Q_R(PRN+1).AND.Q               1190 #
Q_R(SC)                        1191 #
Q_R(SRC!1).AND.K[]             1192 #
Q_RC(SC)                       1193 #
Q_RC[]                         1194 #
Q_RC[](FRAC)                   1195 #
Q_R[]                          1196 #  1763
Q_R[](FRAC)                    1197 #
Q_R[].AND.K[]                  1198 #
Q_R[].AND.K[].RIGHT            1199 #
Q_R[].ANDNOT.K[]               1200 #
Q_R[].OR.K[]                   1201 #
Q_SC                           1202 #
Q_SHF                          1203 #
R(DST)_ALU                     1205 #
R(DST)_D                       1206 #
R(DST)_D.SXT[].RIGHT           1207 #
R(PRN)_Q+D.RLOG                1209 #
R(PRN)_ALU                     1210 #
R(PRN)_D                       1211 #
R(PRN)_D+K[].RLOG              1212 #
R(PRN)_D-K[].RLOG              1213 #
```

```
; NEWSAM.MCR              MICRO2  1L(02)    18-JAN-82  16:15:06                    Page   55
;                         Cross Reference Listing - Macro Names

R(PRN)_D.OR.Q                 1214 #
R(PRN)_D[]Q                   1215 #
R(PRN)_K[]                    1216 #
R(PRN)_LA+K[].RLOG            1217 #
R(PRN)_LA+Q                   1218 #
R(PRN)_LA-K[].RLOG            1219 #
R(PRN)_LA[]MASK               1220 #
R(PRN)_LC                     1221 #
R(PRN)_PACK.FP                1222 #
R(PRN)_Q                      1223 #
R(PRN)_Q+K[].RLOG             1224 #
R(PRN)_Q-K[].RLOG             1225 #
R(PRN+1)_ALU                  1226 #
R(PRN+1)_D                    1227 #
R(PRN+1)_D.OR.Q               1228 #
R(PRN+1)_K[]                  1229 #
R(PRN+1)_LA                   1230 #
R(PRN+1)_LC                   1231 #
R(PRN+1)_Q                    1232 #
R(SC)_ALU                     1234 #
R(SC)_D                       1235 #
R(SC)_K[]                     1236 #
R(SC)_LA                      1237 #
R(SC)_LA+D                    1238 #
R(SC)_LA-D                    1239 #
R(SC)_LC                      1240 #
R(SC)_Q                       1241 #
R(SP1)_ALU                    1243 #
R(SP1)_D                      1244 #
R(SP1)_K[]                    1245 #
R(SP1)_PACK.FP                1246 #
R(SP1)_Q                      1247 #
R(SP1+1)_LC                   1248 #
R(SP1+1)_Q                    1249 #
R(SRC!1)_ALU                  1251 #
R(SRC!1)_D(B)                 1252 #
R(SRC)_ALU                    1253 #
R(SRC)_D                      1254 #
R(SRC)_D(B)                   1255 #
R(SRC)_D+K[].RLOG             1256 #
R(SRC)_D-K[].RLOG             1257 #
R(SRC)_LC                     1258 #
R(SRC)_Q                      1259 #
R6_D+K[].RLOG                 1261 #
R6_LA+K[].RLOG                1262 #
R6_LA-K[].RLOG                1263 #
RC(SC)_0-LC                   1265 #
RC(SC)_ALU                    1266 #
RC(SC)_ALU.RIGHT              1267 #
RC(SC)_D                      1268 #
RC(SC)_Q                      1269 #
RC[]&VA_D+Q                   1271 #
RC[]_0                        1272 #
RC[]_0+K[]+1                  1273 #
RC[]_0+LC+1                   1274 #
```

```
; NEWSAM.MCR              MICRO2  1L(02)    18-JAN-82  16:15:06          Page   56
;                         Cross Reference Listing - Macro Names

RC[]_0+MASK+1                1275 #
RC[]_0+MASK+1.RIGHT2         1276 #
RC[]_0-D                     1277 #
RC[]_ALU                     1278 #
RC[]_ALU.LEFT                1279 #
RC[]_ALU.LEFT2               1280 #
RC[]_ALU.LEFT3               1281 #
RC[]_ALU.RIGHT               1282 #
RC[]_ALU.RIGHT2              1283 #
RC[]_D                       1284 #
RC[]_D(B)                    1285 #
RC[]_D+KC]                   1286 #
RC[]_D-KC]                   1287 #
RC[]_D.OXTC]                 1288 #
RC[]_D.AND.KC]               1289 #
RC[]_D.AND.MASK              1290 #
RC[]_D.ANDNOT.Q              1291 #
RC[]_D.CTX                   1292 #
RC[]_D.LEFT                  1293 #
RC[]_D.LEFT3                 1294 #
RC[]_D.OR.KC]                1295 #
RC[]_D.OR.Q                  1296 #
RC[]_D.ORNOT.KC]             1297 #
RC[]_D.SXTC]                 1298 #
RC[]_KC]                     1299 #
RC[]_KC]+1                   1300 #
RC[]_KC].LEFT2               1301 #
RC[]_KC].LEFT3               1302 #
RC[]_KC].RIGHT2              1303 #
RC[]_LA                      1304 #
RC[]_LA+LB.CTX               1305 #
RC[]_LA-KC]                  1306 #
RC[]_LA.AND.KC]              1307 #
RC[]_LA.CTX                  1308 #
RC[]_LB                      1309 #
RC[]_LB.LEFT                 1310 #
RC[]_LC                      1311 #
RC[]_NOT.Q                   1312 #
RC[]_PACK.FP                 1313 #
RC[]_PC                      1314 #
RC[]_Q                       1315 #
RC[]_Q+1                     1316 #
RC[]_Q+KC]                   1317 #
RC[]_Q+LC                    1318 #
RC[]_Q+PC                    1319 #
RC[]_Q+PC+1                  1320 #
RC[]_Q-KC]                   1321 #
RC[]_Q-LC                    1322 #
RC[]_Q-MASK-1                1323 #
RC[]_Q.OXTC]                 1324 #
RC[]_Q.AND.KC]               1325 #
RC[]_Q.ANDNOT.KC]            1326 #
RC[]_Q.LEFT                  1327 #
RC[]_Q.LEFT3                 1328 #
RC[]_Q.RIGHT                 1329 #
```

```
; NEWSAM.MCR              MICRO2  1L(02)    18-JAN-82  16:15:06                    Page  57
;                         Cross Reference Listing - Macro Names

RC[]_Q.RIGHT2               1330 #
RC[]_Q.SXT[]                1331 #
RC[]_RLOG.RIGHT             1332 #
RETURN0                     1590 #
RETURN1                     1591 #
RETURN10                    1592 #
RETURN100                   1593 #
RETURN10C                   1594 #
RETURN10E                   1595 #
RETURN12                    1596 #
RETURN18                    1597 #
RETURN1F                    1598 #
RETURN2                     1599 #
RETURN20                    1600 #
RETURN24                    1601 #
RETURN3                     1602 #
RETURN4                     1603 #
RETURN40                    1604 #
RETURN60                    1605 #
RETURN61                    1606 #
RETURN8                     1607 #
RETURN9                     1608 #
RETURNF                     1609 #
RETURN[]                    1610 #
RLOG.EMPTY?                 1698 #
ROR?                        1699 #
R[]&VA_LA+K[]               1334 #
R[]&VA_LA-K[]               1335 #
R[]&VA_LA-K[].RLOG          1336 #
R[]&VA_Q-K[]                1337 #
R[]_0                       1338 #
R[]_0+LB+1                  1339 #
R[]_0-1                     1340 #
R[]_0-D                     1341 #
R[]_0-K[]                   1342 #
R[]_0-LB                    1343 #
R[]_0-Q                     1344 #
R[]_ALU                     1345 #  1821    1826
R[]_ALU.LEFT                1346 #
R[]_ALU.LEFT3               1347 #
R[]_ALU.RIGHT               1348 #
R[]_ALU.RIGHT2              1349 #
R[]_D                       1350 #
R[]_D+K[]                   1351 #
R[]_D+Q                     1352 #
R[]_D+Q+1                   1353 #
R[]_D-K[]                   1354 #
R[]_D-LC-1                  1355 #
R[]_D-Q                     1356 #
R[]_D.AND.K[]               1357 #
R[]_D.OR.LC                 1358 #
R[]_D.OR.PACK.FP            1359 #
R[]_D.OR.Q                  1360 #
R[]_K[]                     1361 #
R[]_LA                      1362 #
```

```
; NEWSAM.MCR              MICRO2  1L(02)    18-JAN-82  16:15:06                    Page   58
;                         Cross Reference Listing - Macro Names

RC]_LA+D                    1363 #
RC]_LA+D+1                  1364 #
RC]_LA+KC]                  1365 #
RC]_LA+KC]+1                1366 #
RC]_LA+KC].RLOG             1367 #
RC]_LA+LC                   1368 #
RC]_LA+MASK+1               1369 #
RC]_LA+Q                    1370 #
RC]_LA-D                    1371 #
RC]_LA-KC]                  1372 #
RC]_LA-KC].RLOG             1373 #
RC]_LA-MASK-1               1374 #
RC]_LA-Q                    1375 #
RC]_LA.AND.KC]              1376 #
RC]_LA.OR.D                 1377 #
RC]_LA.ORNOT.MASK           1378 #
RC]_LB                      1379 #
RC]_LC                      1380 #
RC]_LC.RIGHT                1381 #
RC]_NOT.Q                   1382 #
RC]_NOT.D                   1383 #
RC]_NOT.MASK                1384 #
RC]_NOT.Q                   1385 #
RC]_PACK.FP                 1386 #
RC]_Q                       1387 #  1830
RC]_Q+1                     1388 #
RC]_Q+5                     1389 #
RC]_Q+KC]                   1390 #
RC]_Q+LB                    1391 #
RC]_Q+LC                    1392 #
RC]_Q-D                     1393 #
RC]_Q-D-1                   1394 #
RC]_Q-KC]                   1395 #
RC]_Q-KC].RLOG              1396 #
RC]_Q-LC                    1397 #
RC]_Q.AND.KC]               1398 #
RC]_Q.ANDNOT.KC]            1399 #
RC]_Q.OR.D                  1400 #
RC]_Q.ORNOT.KC]             1401 #
RC]_Q.RIGHT.1               1402 #
RC]_RLOG.RIGHT.1            1403 #
SC$STATE_STATE-RC](EXP)     1405 #
SC.GT.0?                    1701 #
SC.NE.0?                    1702 #
SC?                         1703 #
SC_0(A)                     1406 #
SC_0-KC]                    1407 #
SC_ALU                      1408 #
SC_ALU(EXP)                 1409 #
SC_D                        1410 #
SC_D(EXP)                   1411 #
SC_D(EXP)(A)                1412 #
SC_D(EXP)(B)                1413 #
SC_D-KC]                    1414 #
SC_D.OXTC]-KC]              1415 #
```

```
SC_D.OXT[],XOR.K[]                1416 #
SC_D.AND.K[]                      1417 #
SC_D.OR.K[]                       1418 #
SC_D.SXT[]                        1419 #
SC_EALU                           1420 #
SC_FE                             1421 #
SC_K[]                            1422 #
SC_K[].ALU                        1423 #
SC_LA                             1424 #
SC_LA.AND.K[]                     1425 #
SC_LC(EXP)                        1426 #
SC_NABS(SC-FE)                    1427 #
SC_PSLADDR                        1428 #
SC_Q                              1429 #
SC_Q(EXP)                         1430 #
SC_Q(EXP)(B)                      1431 #
SC_Q+K[]                          1432 #
SC_Q-K[]                          1433 #
SC_Q.AND.K[]                      1434 #
SC_Q.OR.K[]                       1435 #
SC_Q.SXT[]                        1436 #
SC_R[]                            1437 #
SC_R[](EXP)                       1438 #
SC_R[]                            1439 #
SC_R[](EXP)                       1440 #
SC_R[].AND.K[]                    1441 #
SC_SC+1                           1442 #
SC_SC+EXP(Q)(A)                   1443 #
SC_SC+FE                          1444 #
SC_SC+K[]                         1445 #
SC_SC+SHF.VAL                     1446 #
SC_SC-FE                          1447 #
SC_SC-K[]                         1448 #
SC_SC-SHF.VAL                     1449 #
SC_SC.ANDNOT.FE                   1450 #
SC_SC.ANDNOT.K[]                  1451 #
SC_SC.OR.K[]                      1452 #
SC_SHF.VAL                        1453 #
SC_STATE                          1454 #
SC_STATE.ANDNOT.K[]               1455 #
SC_STATE.OR.K[]                   1456 #
SD_NOT.SD                         1457 #
SD_SS                             1458 #
SET.CC(BYTE)                      1612 #
SET.CC(INST)                      1613 #
SET.CC(LONG)                      1614 #
SET.CC(ROR)                       1615 #
SET.CC(WORD)                      1616 #
SET.FPD                           1617 #
SET.NEST.ERR                      1618 #
SET.PSL.C(AMX)                    1619 #
SET.V                             1620 #
SIGNS?                            1704 #
SPEC                              1621 #
SPECG                             1622 #
```

```
; NEWSAM.MCR              MICRO2  1L(02)    18-JAN-82  16:15:06                    Page   60
;                         Cross Reference Listing - Macro Names

SRC.PC?                     1705 #
SS?                         1706 #
SS_0&SD_0                   1459 #
SS_ALU15                    1460 #
SS_SD                       1461 #
SS_SS.XOR.ALU15&SD_ALU15    1462 #
START.IB                    1623 #
STATE(7)?                   1707 #
STATE0?                     1708 #  1777    1838
STATE1-0?                   1709 #
STATE1?                     1710 #
STATE2?                     1711 #
STATE3-0?                   1712 #
STATE3?                     1713 #
STATE4?                     1714 #
STATE5?                     1715 #
STATE6?                     1716 #
STATE7-4?                   1717 #
STATE_0(A)                  1463 #
STATE_AMX.EXP               1464 #
STATE_D(EXP)                1465 #
STATE_FE                    1466 #
STATE_FIRST                 1467 #
STATE_INNEROBJ              1468 #
STATE_INNERSRC              1469 #
STATE_KC]                   1470 #  1764    1815
STATE_OUTER                 1471 #
STATE_PREDEC                1472 #
STATE_Q(EXP)                1473 #
STATE_SC.VIA.KMX            1474 #
STATE_SKPLONG               1475 #
STATE_STATE+1               1476 #
STATE_STATE+FE              1477 #
STATE_STATE+KC]             1478 #
STATE_STATE-FE              1479 #
STATE_STATE-KC]             1480 #
STATE_STATE.AN.5TO0         1482 #
STATE_STATE.AN.6TO4         1483 #
STATE_STATE.AN.DESTDBL      1484 #
STATE_STATE.AN.NOTPREDEC    1485 #
STATE_STATE.AN.PREDECZERO   1486 #
STATE_STATE.AN.SKPLONG      1481 #
STATE_STATE.ANDNOT.FE       1487 #
STATE_STATE.ANDNOT.KC]      1488 #
STATE_STATE.ANDNOT.SHF.VAL  1489 #
STATE_STATE.OR.ADJINF       1492 #
STATE_STATE.OR.DEST         1493 #
STATE_STATE.OR.DESTDBL      1494 #
STATE_STATE.OR.FE           1490 #
STATE_STATE.OR.FILL         1495 #
STATE_STATE.OR.FLOAT        1496 #
STATE_STATE.OR.KC]          1491 #
STATE_STATE.OR.MOVE         1497 #
STATE_STATE.OR.PATT1        1498 #
STATE_STATE.OR.PATT2        1499 #
```

```
STOP.IB                    1624 #
SWAPD                      1500 #
TB.TEST?                   1719 #
TEST.TB.RCHK               1626 #
TEST.TB.WCHK               1627 #
TRAP.ACC[]                 1628 #
VA31-30?                   1721 #
VA31?                      1722 #
VA_ALU                     1502 #  1782
VA_D                       1503 #
VA_D+K[]                   1504 #
VA_D+LC                    1505 #
VA_D+Q                     1506 #
VA_D.OXT[]+Q               1507 #
VA_D.ANDNOT.K[]            1508 #
VA_K[]                     1509 #
VA_LA                      1510 #
VA_LA+D                    1511 #
VA_LA+K[]                  1512 #
VA_LA+K[]+1                1513 #
VA_LA+PC                   1514 #
VA_LA+Q                    1515 #
VA_LA-D                    1516 #
VA_LA-K[]                  1517 #
VA_LA-K[]-1                1518 #
VA_LA-Q                    1519 #
VA_LA.AND.LC               1520 #
VA_LA.ANDNOT.K[]           1521 #
VA_LB+D.OXT                1522 #
VA_PC                      1523 #
VA_Q                       1524 #
VA_Q+D                     1525 #
VA_Q+K[]                   1526 #
VA_Q+LB                    1527 #
VA_Q+LB.PC                 1528 #
VA_Q+LC                    1529 #
VA_Q+PC                    1530 #
VA_Q-K[]                   1531 #
VA_Q-LB                    1532 #
VA_Q.ANDNOT.K[]            1533 #
VA_RC[]                    1534 #
VA_R[]                     1535 #
VA_VA+4                    1536 #
WORD                       1630 #
WRITE.DEST                 1631 #
WRITE.G.DEST               1632 #
Z?                         1724 #  1809
ZONED?                     1725 #
```

Cross Reference Listing - Expression Names

```
# NEWSAM.MCR                MICRO2  1L(02)    18-JAN-82  16:15:06                    Page   63
#                           Location / Line Number Index

#Location       0/8      1/9      2/A      3/B      4/C      5/D      6/E      7/F

U 0000 - 1BFF Unused
U 1C00          1784=    1790=    1812=    1816=    1764     1767     1796=    1799=
U 1C08          1771     1777     1822=    1827=    1809     1838              1833=
```

```
; NEWSAM.MCR                MICRO2  1L(02)    18-JAN-82  16:15:06                Page   64
;                           U-code Microword Summary

             BSERCH  Words not
          1C00-1FFF  in bounds
VAXDEF            0           0
BSERCH           15           0

Used             15           0
Remaining      1009

Total microwords used in memory U: 15
Total microwords remaining in memory U: 1009
Highest address used in memory U: 1C0F (hex)
```

```
# NEWSAM.MCR                    MICRO2  1L(02)    18-JAN-82  16:15:06          Page   65
$                               Error Summary

Pass 1 warnings:     0   Pass 2 warnings:     0
Pass 1 errors:       0   Pass 2 errors:       0
```

B.3 THE OBJECT FILE (.ULD)

```
;RTOL
;RADIX 16
[1C04]=0000003C19C0FA1014047C05
[1C05]=0800003C0180FA0000001C08
[1C08]=0001003C0180FB0800001C09
[1C09]=005C171401C0F80040001C00
[1C00]=00192E2400C0F90802001C06
[1C01]=C01800381980F80440500062
[1C06]=001C0020018042100010 1C0C
[1C07]=0000003C0180F800000004F8
[1C0C]=001101000180F88800101C02
[1C02]=00001B3C0180F80000001C0A
[1C03]=00001B3C0580F80014047C0A
[1C0A]=0019201411C0FA8800001C0D
[1C0B]=0019200011C0FA9000001C09
[1C0F]=C001203C0180FA8C40500062
[1C0D]=004D371401C0F80040001C00
FIELD ACF=<71:70>
 CONTROL=3
 NOP=0
 SYNC=1
 TRAP=2
FIELD ACM=<57:55>
 ABORT=1
 POLY.DONE=6
 PWR.UP=0
FIELD ADS=<47:47>
 IBA=1
 VA=0
FIELD ALU=<69:66>
 A=0F
 A+B=5
 A+B+1=4
 A+B+PSL.C=0B
 A+B.RLOG=6
 A-B=0
 A-B-1=2
 A-B.RLOG=1
 AND=0D
 ANDNOT=9
 B=0E
 INST.DEP=3
 NOTA=0A
 OR=0C
 ORNOT=7
 XOR=8
FIELD AMX=<81:80>
 LA=0
 RAMX=1
 RAMX.0XT=3
 RAMX.SXT=2
FIELD BEN=<76:72>
 ACCEL=6
 ALU=1B
 ALU1-0=15
```

```
  C31=3
  D.BYTES=18
  D3-0=19
  DATA.TYPE=8
  DECIMAL=0F
  EALU=12
  END.DP1=8
  IB.0=5
  IB.TEST=0B
  INTERRUPT=0E
  IR2-1=9
  IRC.ROM=4
  LAST.REF=11
  MUL=0C
  NOP=0
  PC.MODES=9
  PSL.CC=1A
  PSL.MODE=1C
  RET=0A
  ROR=2
  SC=14
  SIGNS=0D
  SRC.PC=0A
  STATE3-0=17
  STATE7-4=16
  TB.TEST=1D
  Z=1
FIELD BMX=<84:82>
  KMX=6
  LB=3
  LC=4
  MASK=0
  PACKED.FL=2
  PC=5
  PC.OR.LB=1
  RBMX=7
FIELD CCK=<22:20>
  C_AMX0=6
  INST.DEP=7
  LOAD.UBCC=1
  NOP=0
  NZ_ALU.VC_0=5
  NZ_ALU.VC_VC=6
  N_AMX.Z_TST.VC_VC=3
  ROR=4
  SET.V=2
FIELD CID=<45:42>
  ACK=5
  CONT=7
  NOP=1
  READ.KMX=0B
  READ.SC=9
  WRITE.KMX=0F
  WRITE.SC=0D
FIELD DK=<91:88>
  ACCEL=0A
  BYTE.SWAP=0B
  CLR=0F
  DAL.SC=0D
  DAL.SV=0E
  DIV=4
  LEFT=5
  LEFT2=1
```

```
 NOP=0
 Q=0C
 RIGHT=6
 RIGHT2=2
 SHF=8
 SHF.FL=9
FIELD DT=<79:78>
 BYTE=2
 INST.DEP=3
 LONG=0
 WORD=1
FIELD EALU=<15:13>
 A=0
 A+1=6
 A+B=4
 A-B=5
 ANDNOT=2
 B=3
 NABS.A-B=7
 OR=1
FIELD EBMX=<19:18>
 AMX.EXP=2
 FE=0
 KMX=1
 SHF.VAL=3
FIELD FEK=<24:24>
 LOAD=1
 NOP=0
FIELD FS=<42:42>
 CID=1
 MCT=0
FIELD IBC=<95:92>
 BDEST=7
 CLR.0=0C
 CLR.0-3=0E
 CLR.0.1=4
 CLR.1=0D
 CLR.1-5.COND=0F
 CLR.2.3=5
 FLUSH=2
 NOP=0
 START=3
 STOP=1
FIELD ID.ADDR=<63:58>
 ACC.0=14
 ACC.1=15
 ACC.2=16
 ACC.CS=17
 CES=0C
 CLK.CS=0A
 COMP=1C
 D.SV=2E
 DAY.TIME=1
 ESP=29
 FAULT=1B
 FPDA=2D
 IBUF=0
 INTERVAL=0B
 ISP=2C
 KSP=28
 MAINT=1D
 NXT.PER=9
 PORR=24
```

```
 POLR=3C
 PIBR=25
 PILR=3D
 PARITY=1E
 PCBB=3A
 PSL=0F
 Q.SV=2F
 RXCS=4
 RXDB=5
 SBI.ERR=19
 SBR=26
 SCBB=3B
 SILO=18
 STR=0E
 SLR=3E
 SSP=2A
 SYS.ID=3
 T0=30
 T1=31
 T2=32
 T3=33
 T4=34
 T5=35
 T6=36
 T7=37
 T8=38
 T9=39
 TBER0=12
 TBER1=13
 TBUF=10
 TIME.ADDR=1A
 TXCS=6
 TXDB=7
 UBREAK=21
 USP=2B
 USTACK=20
 VECTOR=0D
 WCS.ADDR=22
 WCS.DATA=23
FIELD IEK=<31:30>
 EACK=3
 IACK=2
 ISTR=1
 NOP=0
ADDRESS J=<12:0>
 INT.B=4F8
 IRD=62
 SRCH=1C04
 SRCH.1=1C09
 SRCH.2=1C00
 SRCH.3=1C06
 SRCH.4=1C0C
 SRCH.5=1C0A
 SRCH.6=1C0D
FIELD KMX=<63:58>
 .1=1
 .10=19
 .14=8
 .18=1F
 .19=2E
 .1A=39
 .1B=3B
 .1E=14
```

```
 .1F=23
 .1F00=24
 .2=2
 .20=1D
 .24=3A
 .28=0B
 .3=3
 .30=1E
 .3030=32
 .34=0A
 .3F=15
 .3FF=20
 .4=4
 .40=0C
 .4000=2C
 .50=0D
 .6=35
 .60=29
 .7=17
 .7C=27
 .7E=3E
 .7F=16
 .7FF0=0E
 .8=0
 .80=10
 .8000=11
 .88=31
 .9=36
 .A=3D
 .A0=9
 .B0=25
 .C=21
 .C0=34
 .D=22
 .DFCF=2B
 .E003=26
 .EF=0F
 .F=18
 .F0=33
 .FF=12
 .FF00=13
 .FFE0=28
 .FFE8=1A
 .FFF0=1B
 .FFF1=2D
 .FFF5=38
 .FFF6=37
 .FFF8=1C
 .FFF9=2F
 .FFFC=3C
 .FFFF=30
 SC=7
 SP1.CON=5
 SP2.CON=6
 ZERO=6
FIELD MCT=<47:42>
 ALLOW.IB.READ=3E
 EXTWRITE.P=28
 INVALIDATE=24
 LOCKREAD.P=3A
 LOCKREAD.V.NOCHK=1A
 LOCKREAD.V.WCHK=1C
 LOCKWRITE.P=2E
```

```
  LOCKWRITE.V.XCHK=0E
  MEM.NOP=2
  READ.INT.SUM=36
  READ.P=32
  READ.V.IBCHK=16
  READ.V.NEWPC=18
  READ.V.NOCHK=12
  READ.V.RCHK=10
  READ.V.WCHK=14
  SBI.HOLD=20
  SBI.HOLD+UNJAM=22
  TEST.RCHK=0
  TEST.WCHK=4
  VALIDATE=26
  WRITE.P=2A
  WRITE.V.NOCHK=0A
  WRITE.V.WCHK=0C
FIELD MSC=<29:26>
  CHK.CHM=1
  CHK.FLT.OPR=2
  CHK.ODD.ADDR=3
  CLR.FPD=8
  CLR.NEST.ERR=0A
  INH.CM.ADDR=0F
  IRD=4
  LOAD.ACC.CC=6
  LOAD.STATE=5
  NOP=0
  READ.RLOG=7
  RETRY.NO.TRAP=0D
  RETRY.TRAP=0E
  SECOND.REF=0C
  SET.FPD=9
  SET.NEST.ERR=0B
FIELD PCK=<34:32>
  NOP=0
  PC+1=4
  PC+2=5
  PC+4=6
  PC+N=7
  PC_IBA=2
  PC_VA=1
  VA+4=3
FIELD QK=<54:51>
  ACCEL=0B
  CLR=0F
  D=0C
  DEC.CON=0A
  ID=0E
  LEFT=5
  LEFT2=1
  NOP=0
  RIGHT=6
  RIGHT2=2
  SHF=8
  SHF.FL=9
FIELD RAMX=<77:77>
  D=0
  Q=1
FIELD RBMX=<77:77>
  D=1
  Q=0
FIELD SCK=<23:23>
```

```
 LOAD=1
 NOP=0
FIELD SGN=<50:48>
 ADD.SUB=6
 CLR.SD+SS=7
 LOAD.SS=1
 NOP=0
 NOT.SD=3
 SD.FROM.SS=4
 SS.FROM.SD=2
 SS.XOR.ALU=5
FIELD SHF=<87:85>
 ALU=0
 ALU.DT=3
 LEFT=1
 LEFT3=5
 RIGHT=2
 RIGHT2=4
FIELD SI=<57:55>
 ASHL=2
 ASHR=1
 DIV=5
 DIVD=0
 MUL+=6
 MUL-=7
 ZERO=3
FIELD SMX=<17:16>
 ALU=2
 ALU.EXP=3
 EALU=0
 FF=1
FIELD SPO=<41:35>
 LOAD.LC.SC=6
 NOP=0
 WRITE.RC.SC=7
FIELD SPO.AC=<41:38>
 LOAD.LA=2
 LOAD.LAB=1
 WRITE.RAB=3
FIELD SPO.ACN=<37:35>
 PRN=3
 PRN+1=4
 SC=5
 SP1+1=6
 SP1.SP1=0
 SP2.SP1=2
 SP2.SP2=1
FIELD SPO.ACN11=<37:35>
 DST.DST=1
 DST.SRC=2
 SC=5
 SRC.OR.1=4
 SRC.SRC=0
FIELD SPO.R=<41:39>
 LOAD.LAB=4
 LOAD.LAB1.WRITE.RC=6
 LOAD.LC=2
 LOAD.LC.WRITE.RAB1=7
 WRITE.RAB=5
 WRITE.RC=3
FIELD SPO.RAB=<38:35>
 AP=0C
 FP=0D
```

```
  R0=0
  R1=1
  R15=0F
  R2=2
  R3=3
  R4=4
  R5=5
  R6=6
  R7=7
  SP=0E
 FIELD SPO.RC=<38:35>
  LC.SV=8
  MBIT.VA=0F
  PC.SV=0C
  PTE.MASK=0F
  PTE.PA=0B
  PTE.VA=0A
  SC.SV=0D
  T0=0
  T1=1
  T2=2
  T3=3
  T4=4
  T5=5
  T6=6
  T7=7
  VA.REF=0E
  VA.SV=9
 FIELD SUB=<65:64>
  CALL=1
  NOP=0
  RET=2
  SPEC=3
 FIELD VAK=<25:25>
  LOAD=1
  NOP=0
END
```

APPENDIX C

SAMPLE MICROPROGRAM FOR SYSTEM REVISION < 7

This appendix contains a sample VAX 11/780 microprogram, which performs an unsigned binary search on a vector of longwords in main memory. The parameters of the routine, the value to be searched for and the beginning and end of the vector, are passed in registers.

A command file that assembles, loads, and executes this sample microprogram is provided in the VAX 11/780 WCS kit. To invoke this file in the VMS environment, type:

```
@[SYSEXE]WCSTOLTST
```

This command file assembles the input listing (Section C.1) and produces the listing file (Section C.2) and the object file (Section C.3) which are written to [VAXWCSTOL]SAMPLE.MCR and [VAXWCSTOL]SAMPLE.ULD. It then loads the object file into the extended WCS and runs the test program BSTEST (Appendix D). BSTEST executes an XFC instruction, which causes the sample microprogram loaded in the WCS to be executed. If the microprogram executes properly, BSTEST prints the following message on the terminal:

```
"Successful Test Completion"
```

C.1 THE INPUT FILE (.MIC)

```
        .TOC "Binary search routine"
        .REGION /1400,17FF              ;User wcs space.
        .BOUNDS/BSERCH:1400,17FF        ;This defines the report boundries
                                        ;for the U-code microword summary page
                                        ;and names the report boundary BSERCH.

; Sample microcode to perform an unsigned binary search through
; a vector of aligned longwords in main memory.
;
; INPUTS
;       R0 - Search comparand.  Routine succeeds by finding a
;               memory cell containing same data as R0.
;       R1 - Lower address bound.  Aligned longword address of
;               lowest address of vector to be searched.
;       R2 - Upper address bound.  Aligned longword address of
;               highest address of vector to be searched.
; It is implied that R1 lssu R2, and that the memory between the
; addresses in R1 and R2 contains a sorted vector, in ascending
; unsigned order.
;
; Outputs if search finds a match.
;       CC<Z> - Clear
;       R0    - Search comparand.
;       R1    - Match address.  Address of longword containing same data as R0.
;       R2    - Used by search for temporary address values.
;
; Outputs if search does not find a match.
;       CC<Z> - Set
;       R0    - Search comparand.
;       R1    - Used by search for temporary address values.
;       R2    - Used by search for temporary address values.
;
```

```
SRCH:;------------------------;
Q_R[R2],;GET UPPER BOUND ADDR TO Q
STATE_K[ZERO];INITIALIZE STATE REGISTER

;------------------------;
D_R[R0];GET COMPARAND TO HOLD IN RC

;------------------------;
ALU_D,;PREPARE TO WRITE COMPARAND TO RC
LAB_R1&RC[T1]_ALU;WRITE COMPARAND, GET LOWER BOUND


SRCH.1:;------------------------;
Q_(LA+Q).RIGHT,;COMPUTE MIDPOINT ADDRESS
INTRPT.STROBE,;TEST FOR INTERRUPT REQUESTS
STATE0?;IS IT TIME TO STOP?

=0
SRCH.2:;0----------------------;STATE0=0.  KEEP LOOKING FOR MATCH.
Q_Q.ANDNOT.K[.3],;FORCE LONGWORD ALIGNMENT
VA_ALU,;GET READY TO READY MIDPOINT OF VECTOR
LC_RC[T1],;LATCH COMPARAND INTO LC
INT?,J/SRCH.3;IS THERE AN INTERRUPT REQUEST?

;1----------------------;STATE0=1. SEARCH FAILED.  NO MATCH.
ALU_K[ZERO],;
CCK/NZ_ALU.VC_0,LONG,;RETURN Z=1 TO FLAG FAILURE.
CLR.IB.OPC,PC_PC+1,;MOVE ON TO THE NEXT INSTRUCTION
J/IRD;

=110
SRCH.3:;110--------------------;NO INTERRUPT REQUESTS
D[LONG]_CACHE,;READY MIDPOINT ENTRY OF VECTOR
ALU_R[R2].XOR.Q,;COMPARE MIDPOINT EQL UPPER BOUND
CLK.UBCC,J/SRCH.4;

;111--------------------;INTERRUPT REQUEST IS UP
J/INT.B;TAKE IT.  RESUME FROM REG'S AS IS.
```

```
; WE HAVE ALSO SET THE MICROBRACH Z BIT ACCORDING TO A COMPARE OF
; THE MEMORY ADDRESS WITH THE CURRENT UPPER BOUND.  IF THEY ARE
; EQUAL, THIS IS THE LAST POSSIBLE COMPARISON.  A MATCH FAILURE
; HERE IMPLIES THAT THERE IS NO MATCH TO BE FOUND.

SRCH.4:;-----------------------;
ALU_D-LC,;COMPARE MEMORY TO COMPARAND
LONG,CLK.UBCC,;RECORD COMPARE RESULT
LA_RA[R1],;LATCH LOWER BOUND INTO LA (LB HAS ???
Z?;IS MIDPOINT EQL UPPER BOUND?

=0;0----------------------;ALU Z=0.  NOT END OF SEARCH
ALU?,J/SRCH.5;TEST RESULT OF COMPARE

;1----------------------;ALU Z=1.  END OF SEARCH
STATE_K[.1],;SET STATE0 TO MARK END OF SEARCH.
ALU?;CHECK FOR LAST CHANCE MATCH

=1010
SRCH.5:;1010-------------------;ALU Z=0, C=1.  R0 GTRU MEM
Q_Q+K[.4],;LOWER LIMIT MUST BE GREATER THAN THIS
R[R1]_ALU,;REMEMBER IN R1.
J/SRCH.6;

;1011-------------------;ALU Z=0, C=0.  R0 LSSU MEM
Q_Q-K[.4],;UPPER LIMIT MUST BE LESS THAN THIS
R[R2]_ALU,;REMEMBER IN R2
J/SRCH.1;GO TRY AGAIN

=1111;1111-------------------;ALU Z=1, C=1.  R0 EQL MEM
R[R1]_Q,;FOUND IT!
CCK/NZ_ALU.VC_0,LONG,;SET Z=0 TO INDICATE MATCH
CLR.IB.OPC,PC_PC+1,;GO TO NEXT INSTRUCTION
J/IRD

SRCH.6:;-----------------------;
Q_(Q+LB).RIGHT,;COMPUTE NEW MIDPOINT, LOOP
INTRPT.STROBE,;
STATE0?,J/SRCH.2;CHECK FOR END, LOOP

; DEFINE LABLES TO INTERFACE WITH PCS
0062:IRD:
04F8:INT.B:
$
$
```

C.2 THE LISTING FILE (.MCR)

```
; OLDSAM.MCR                MICRO2  1L(02)    18-JAN-82  16:19:40                    Page     1
;                           Table of Contents

; 2     Machine definition    : Control word chart
; 56    Machine definition    : ACF, ACM, ADS, ALU, AMX
; 97    Machine definition    : BEN, BMX
; 150   Machine definition    : CCK, CID, DK, DT
; 205   Machine definition    : EALU, EBMX, FEK, FS, IEK, IBC
; 255   Machine definition    : ID.ADDR, J
; 330   Machine definition    : KMX
; 405   Machine definition    : MCT, MSC
; 452   Machine definition    : PCK, QK, RAMX, RBMX
; 487   Machine definition    : SCK, SGN, SHF, SI, SMX
; 529   Machine definition    : SPO, SPO.AC, SPO.ACN, SPO.ACN11, SPO.R
; 568   Machine definition    : SPO.RAB, SPO.RC, SUB, VAK
; 617   Machine definition    : Validity checks
; 624   Macro definition      : Register transfer macros
; 1538  Macro definition      : Non-transfer macros
; 1634  Macro definition      : Branch enable macros
; 1729  Binary search routine
```

```
; OLDSAM.MCR                MICRO2  1L(02)     18-JAN-82  16:19:40                    Page     2
; VAXDEF.MIC

                                    #1      .NOLIST                     #Inhibit listing for VAXDEF.MIC
                                    #1728   .LIST
```

```
; OLDSAM.MCR              MICRO2  1L(02)    18-JAN-82  16:19:40                                    Page  39
; BSERCH.MIC              Binary search routine

                #1729           .TOC "Binary search routine"
                #1730           .REGION /1400,17FF              ;User wcs space.
                #1731           .BOUNDS/BSERCH:1400,17FF        ;This defines the report boundries
                #1732                                           ;for the U-code microword summary page
                #1733                                           ;and names the report boundary BSERCH.
                #1734
                #1735   ; Sample microcode to perform an unsigned binary search through
                #1736   ; a vector of aligned longwords in main memory.
                #1737   ;
                #1738   ; INPUTS
                #1739   ;       R0 - Search comparand.  Routine succeeds by finding a
                #1740   ;               memory cell containing same data as R0.
                #1741   ;       R1 - Lower address bound.  Aligned longword address of
                #1742   ;               lowest address of vector to be searched.
                #1743   ;       R2 - Upper address bound.  Aligned longword address of
                #1744   ;               highest address of vector to be searched.
                #1745   ; It is implied that R1 lssu R2, and that the memory between the
                #1746   ; addresses in R1 and R2 contains a sorted vector, in ascending
                #1747   ; unsigned order.
                #1748   ;
                #1749   ; Outputs if search finds a match.
                #1750   ;       CC<Z> - Clear
                #1751   ;       R0    - Search comparand.
                #1752   ;       R1    - Match address.  Address of longword containing same data as R0.
                #1753   ;       R2    - Used by search for temporary address values.
                #1754   ;
                #1755   ; Outputs if search does not find a match.
                #1756   ;       CC<Z> - Set
                #1757   ;       R0    - Search comparand.
                #1758   ;       R1    - Used by search for temporary address values.
                #1759   ;       R2    - Used by search for temporary address values.
                #1760   ;
```

```
; OLDSAM.MCR              MICRO2  1L(02)    18-JAN-82  16:19:40                    Page   40
; BSERCH.MIC              Binary search routine

                                        ;1761
                                        ;1762   SRCH:   ;-----------------------;
                                        ;1763           Q_RCR2],                ;GET UPPER BOUND ADDR TO Q
U 1404, 0000,003C,19C0,FA10,1404,7405   ;1764           STATE_K[ZERO]           ;INITIALIZE STATE REGISTER
                                        ;1765
                                        ;1766           ;-----------------------;
U 1405, 0800,003C,0180,FA00,0000,1408   ;1767           D_RCR0]                 ;GET COMPARAND TO HOLD IN RC
                                        ;1768
                                        ;1769           ;-----------------------;
                                        ;1770           ALU_D,                  ;PREPARE TO WRITE COMPARAND TO RC
U 1408, 0001,003C,0180,FB08,0000,1409   ;1771           LAB_R1&RC[T1]_ALU       ;WRITE COMPARAND, GET LOWER BOUND
                                        ;1772
                                        ;1773
                                        ;1774   SRCH.1: ;-----------------------;
                                        ;1775           Q_(LA+Q).RIGHT,         ;COMPUTE MIDPOINT ADDRESS
                                        ;1776           INTRPT.STROBE,          ;TEST FOR INTERRUPT REQUESTS
U 1409, 005C,1714,01C0,F800,4000,1400   ;1777           STATE0?                 ;IS IT TIME TO STOP?
                                        ;1778
                                        ;1779   =0
                                        ;1780   SRCH.2: ;0----------------------;STATE0=0.  KEEP LOOKING FOR MATCH.
                                        ;1781           Q_Q.ANDNOT.K[.3],       ;FORCE LONGWORD ALIGNMENT
                                        ;1782           VA_ALU,                 ;GET READY TO READY MIDPOINT OF VECTOR
                                        ;1783           LC_RC[T1],              ;LATCH COMPARAND INTO LC
U 1400, 0019,2E24,0DC0,F908,0200,1406   ;1784           INT?,J/SRCH.3           ;IS THERE AN INTERRUPT REQUEST?
                                        ;1785
                                        ;1786           ;1----------------------;STATE0=1. SEARCH FAILED.  NO MATCH.
                                        ;1787           ALU_K[ZERO],            ;
                                        ;1788           CCK/NZ_ALU.VC_0,LONG,   ;RETURN Z=1 TO FLAG FAILURE.
                                        ;1789           CLR.IB.OPC,PC_PC+1,     ;MOVE ON TO THE NEXT INSTRUCTION
U 1401, C018,0038,1980,F804,4050,0062   ;1790           J/IRD                   ;
                                        ;1791
                                        ;1792   =110
                                        ;1793   SRCH.3: ;110--------------------;NO INTERRUPT REQUESTS
                                        ;1794           D[LONG]_CACHE,          ;READY MIDPOINT ENTRY OF VECTOR
                                        ;1795           ALU_RCR2].XOR.Q,        ;COMPARE MIDPOINT EQL UPPER BOUND
U 1406, 001C,0020,0180,4210,0010,140C   ;1796           CLK.UBCC,J/SRCH.4       ;
                                        ;1797
                                        ;1798           ;111--------------------;INTERRUPT REQUEST IS UP
U 1407, 0000,003C,0180,F800,0000,04F8   ;1799           J/INT.B                 ;TAKE IT.  RESUME FROM REG'S AS IS.
```

```
; OLDSAM.MCR              MICRO2  1L(02)    18-JAN-82  16:19:40                    Page  41
; BSEARCH.MIC             Binary search routine

                                    #1800   ; WE HAVE ALSO SET THE MICROBRACH Z BIT ACCORDING TO A COMPARE OF
                                    #1801   ; THE MEMORY ADDRESS WITH THE CURRENT UPPER BOUND.  IF THEY ARE
                                    #1802   ; EQUAL, THIS IS THE LAST POSSIBLE COMPARISON.  A MATCH FAILURE
                                    #1803   ; HERE IMPLIES THAT THERE IS NO MATCH TO BE FOUND.
                                    #1804
                                    #1805   SRCH.4: ;-----------------------;
                                    #1806           ALU_D-LC,               ;COMPARE MEMORY TO COMPARAND
                                    #1807           LONG,CLK.UBCC,          ;RECORD COMPARE RESULT
                                    #1808           LA_RAC[1],              ;LATCH LOWER BOUND INTO LA (LB HAS ???
U 140C, 0011,0100,0180,F888,0010,1402   #1809           Z?                      ;IS MIDPOINT EQL UPPER BOUND?
                                    #1810
                                    #1811   =0      ;0----------------------;ALU Z=0.  NOT END OF SEARCH
U 1402, 0000,1B3C,0180,F800,0000,140A   #1812           ALU?,J/SRCH.5           ;TEST RESULT OF COMPARE
                                    #1813
                                    #1814           ;1----------------------;ALU Z=1.  END OF SEARCH
                                    #1815           STATE_RC.1],            ;SET STATE0 TO MARK END OF SEARCH.
U 1403, 0000,1B3C,0580,F800,1404,740A   #1816           ALU?                    ;CHECK FOR LAST CHANCE MATCH
                                    #1817
                                    #1818   =1010
                                    #1819   SRCH.5: ;1010--------------------;ALU Z=0, C=1.  R0 GTRU MEM
                                    #1820           Q_Q+RC.4],              ;LOWER LIMIT MUST BE GREATER THAN THIS
                                    #1821           RCR1]_ALU,              ;REMEMBER IN R1.
U 140A, 0019,2014,11C0,FA88,0000,140D   #1822           J/SRCH.6                ;
                                    #1823
                                    #1824           ;1011--------------------;ALU Z=0, C=0.  R0 LSSU MEM
                                    #1825           Q_Q-RC.4],              ;UPPER LIMIT MUST BE LESS THAN THIS
                                    #1826           RCR2]_ALU,              ;REMEMBER IN R2
U 140B, 0019,2000,11C0,FA90,0000,1409   #1827           J/SRCH.1                ;GO TRY AGAIN
                                    #1828
                                    #1829   =1111    ;1111--------------------;ALU Z=1, C=1.  R0 EQL MEM
                                    #1830           RCR1]_Q,                ;FOUND IT!
                                    #1831           CCK/NZ_ALU.VC_0,LONG,   ;SET Z=0 TO INDICATE MATCH
                                    #1832           CLR.IB.OPC,PC_PC+1,     ;GO TO NEXT INSTRUCTION
U 140F, C001,203C,0180,FA8C,4050,0062   #1833           J/IRD
                                    #1834
                                    #1835   SRCH.6: ;-----------------------;
                                    #1836           Q_(Q+LB).RIGHT,         ;COMPUTE NEW MIDPOINT, LOOP
                                    #1837           INTRPT.STROBE,          ;
U 140D, 004D,3714,01C0,F800,4000,1400   #1838           STATE0?,J/SRCH.2        ;CHECK FOR END, LOOP
                                    #1839
                                    #1840   ; DEFINE LABLES TO INTERFACE WITH PCS
                                    #1841   0062:   IRD:
                                    #1842   04F8:   INT.B:
```

```
# OLDSAM.MCR                MICRO2  1L(02)    18-JAN-82  16:19:40                    Page   42
#                           Cross Reference Listing - Field Names and Defined Values

J                              326 #
        INT.B                  1799    1842 #
        IRD                    1790    1833    1841 #
        SRCH                   1762 #
        SRCH.1                 1774 #  1827
        SRCH.2                 1780 #  1838
        SRCH.3                 1784    1793 #
        SRCH.4                 1796    1805 #
        SRCH.5                 1812    1819 #
        SRCH.6                 1822    1835 #
```

```
; OLDSAM.MCR              MICRO2  1L(02)    18-JAN-82  16:19:40                    Page   43
;                         Cross Reference Listing - Macro Names

AC.LOW?                    1636 #
ACC.SYNC?                  1637 #
ACCEL?                     1638 #
ALIGNED?                   1639 #
ALU.N?                     1640 #
ALU1-0?                    1641 #
ALU?                       1642 #  1812    1816
ALU_-1                      626 #
ALU_0(A)                    627 #
ALU_0+D                     628 #
ALU_0+D+1                   629 #
ALU_0+K[]                   630 #
ALU_0+K[]+1                 631 #
ALU_0+LB+1                  632 #
ALU_0+LC                    633 #
ALU_0+LC+1                  634 #
ALU_0+MASK+1                635 #
ALU_0+Q                     636 #
ALU_0+Q+1                   637 #
ALU_0-D                     638 #
ALU_0-D-1                   639 #
ALU_0-K[]                   640 #
ALU_0-K[]-1                 641 #
ALU_0-LB                    642 #
ALU_0-LC                    643 #
ALU_0-LC-1                  644 #
ALU_0-Q                     645 #
ALU_0-Q-1                   646 #
ALU_0[]D                    647 #
ALU_0[]LC                   648 #
ALU_D                       649 #  1770
ALU_D(B)                    650 #
ALU_D+K[]                   651 #
ALU_D+K[]+1                 652 #
ALU_D+K[].RLOG              653 #
ALU_D+LB                    654 #
ALU_D+LC                    655 #
ALU_D+LC+1                  656 #
ALU_D+LC+PSL.C              657 #
ALU_D+Q                     658 #
ALU_D+Q+1                   659 #
ALU_D+Q+PSL.C               660 #
ALU_D+RLOG                  661 #
ALU_D-K[]                   662 #
ALU_D-K[]-1                 663 #
ALU_D-LB                    664 #
ALU_D-LB.RLOG               665 #
ALU_D-LC                    666 #  1806
ALU_D-LC-1                  667 #
ALU_D-Q                     668 #
ALU_D-Q-1                   669 #
ALU_D.OXT[]                 670 #
ALU_D.OXT[]+K[]             671 #
ALU_D.OXT[]+LC              672 #
ALU_D.OXT[]+Q               673 #
```

```
; OLDSAM.MCR              MICRO2  1L(02)    18-JAN-82  16:19:40                    Page   44
;                         Cross Reference Listing - Macro Names

ALU_D.OXT[]-K[]             674 #
ALU_D.OXT[]-Q               675 #
ALU_D.OXT[].AND.K[]         676 #
ALU_D.OXT[].ANDNOT.K[]      677 #
ALU_D.OXT[].OR.Q            678 #
ALU_D.AND.K[]               679 #
ALU_D.AND.MASK              680 #
ALU_D.ANDNOT.K[]            681 #
ALU_D.ANDNOT.MASK           682 #
ALU_D.ANDNOT.Q              683 #
ALU_D.OR.K[]                684 #
ALU_D.OR.LC                 685 #
ALU_D.OR.Q                  686 #
ALU_D.OR.RC[]               687 #
ALU_D.ORNOT.MASK            688 #
ALU_D.SXT[]                 689 #
ALU_D.SXT[]+K[]             690 #
ALU_D.SXT[]+Q               691 #
ALU_D.SXT[].AND.K[]         693 #
ALU_D.SXT[].ANDNOT.K[]      692 #
ALU_D.XOR.K[]               694 #
ALU_D.XOR.LC                695 #
ALU_D.XOR.Q                 696 #
ALU_D.XOR.RC[]              697 #
ALU_D.XOR.R[]               698 #
ALU_D[]K[]                  699 #
ALU_D[]LB                   700 #
ALU_D[]LC                   701 #
ALU_D[]Q                    702 #
ALU_K[]                     703 #   1787
ALU_LA                      704 #
ALU_LA+K[]                  705 #
ALU_LA+K[]+1                706 #
ALU_LA+K[].RLOG             707 #
ALU_LA+LB                   708 #
ALU_LA+LC                   709 #
ALU_LA+LC+1                 710 #
ALU_LA+LC+PSL.C             711 #
ALU_LA+Q                    712 #
ALU_LA-D                    713 #
ALU_LA-D-1                  714 #
ALU_LA-K[]                  715 #
ALU_LA-K[]-1                716 #
ALU_LA-K[].RLOG             717 #
ALU_LA-LC                   718 #
ALU_LA-Q                    719 #
ALU_LA-Q-1                  720 #
ALU_LA.AND.K[]              721 #
ALU_LA.AND.LC               722 #
ALU_LA.ANDNOT.K[]           723 #
ALU_LA.ANDNOT.MASK          724 #
ALU_LA.OR.K[]               725 #
ALU_LA.XOR.LC               726 #
ALU_LA[]D                   727 #
ALU_LA[]LB                  728 #
```

```
; OLDSAM.MCR              MICRO2  1L(02)    18-JAN-82  16:19:40          Page  45
;                         Cross Reference Listing - Macro Names

ALU_LAC]Q                    729 #
ALU_LB                       730 #
ALU_LC                       731 #
ALU_NOT.D                    732 #
ALU_NOT.KC]                  733 #
ALU_NOT.RCC]                 734 #
ALU_PACK.FP                  735 #
ALU_PC                       736 #
ALU_Q                        737 #
ALU_Q(B)                     738 #
ALU_Q+KC]                    739 #
ALU_Q+KC]+1                  740 #
ALU_Q+LB                     741 #
ALU_Q+LB+1                   742 #
ALU_Q+LC                     743 #
ALU_Q+LC+1                   744 #
ALU_Q+LC+PSL.C               745 #
ALU_Q+MASK                   746 #
ALU_Q-D                      747 #
ALU_Q-D-1                    748 #
ALU_Q-KC]                    749 #
ALU_Q-LB                     750 #
ALU_Q-LC                     751 #
ALU_Q-MASK-1                 752 #
ALU_Q.OXTC]                  753 #
ALU_Q.OXTC]+D                754 #
ALU_Q.OXTC]+D+1              755 #
ALU_Q.OXTC]+KC]              756 #
ALU_Q.OXTC]-D                757 #
ALU_Q.OXTC]-KC]              758 #
ALU_Q.OXTC].ANDNOT.KC]       759 #
ALU_Q.OXTC].OR.D             761 #
ALU_Q.OXTC].OR.KC]           760 #
ALU_Q.AND.D                  762 #
ALU_Q.AND.KC]                763 #
ALU_Q.ANDNOT.KC]             764 #
ALU_Q.ANDNOT.MASK            765 #
ALU_Q.ANDNOT.RC]             766 #
ALU_Q.OR.KC]                 767 #
ALU_Q.OR.LC                  768 #
ALU_Q.ORNOT.KC]              769 #
ALU_Q.SXTC]                  770 #
ALU_Q.SXTC]+KC]              771 #
ALU_Q.SXTC]+LB               772 #
ALU_Q.SXTC]+LB+1             773 #
ALU_Q.SXTC]+PC               774 #
ALU_Q.SXTC].ANDNOT.KC]       775 #
ALU_Q.XOR.D                  776 #
ALU_Q.XOR.KC]                777 #
ALU_Q.XOR.LC                 778 #
ALU_Q.XOR.RCC]               779 #
ALU_QC]D                     780 #
ALU_R(DST)                   781 #
ALU_R(SC).ANDNOT.KC]         782 #
ALU_R(SP1)+KC].RLOG          783 #
```

```
ALU_RC(SC)                    784 #
ALU_RC[]                      785 #
ALU_RLOG                      786 #
ALU_R[]                       787 #
ALU_R[]-K[]                   788 #
ALU_R[].AND.K[]               789 #
ALU_R[].AND.LC                790 #
ALU_R[].ANDNOT.K[]            791 #
ALU_R[].ANDNOT.MASK           792 #
ALU_R[].OR.K[]                793 #
ALU_R[].ORNOT.K[]             794 #
ALU_R[].XOR.K[]               795 #
ALU_R[].XOR.Q                 796 #   1795
B.FORK                        1540 #
BCDSGN?                       1644 #
BYTE                          1541 #
C.FORK                        1543 #
C31?                          1646 #
CACHE.INVALIDATE              1544 #
CACHE.P_D[]                   798 #
CACHE[]_D                     799 #
CACHE_D(QUAD)                 800 #
CACHE_D.INST.DEP              801 #
CACHE_D[]                     802 #
CACHE_D[].LK                  803 #
CACHE_D[].NOCHK               804 #
CALL                          1545 #
CALL[]                        1546 #
CHK.FLT.OPR                   1547 #
CHK.ODD.ADDR                  1548 #
CLK.UBCC                      1549 #  1796    1807
CLR.FPD                       1551 #
CLR.IB.COND                   1552 #
CLR.IB.OPC                    1553 #  1789    1832
CLR.IB.SPEC                   1554 #
CLR.IB0-1                     1555 #
CLR.IB0-3                     1556 #
CLR.IB2-3                     1557 #
CLR.IB2-5                     1558 #
CLR.NEST.ERR                  1559 #
CLR.SD&SS                     1560 #
CONSOLE.MODE?                 1647 #
D&Q_D+Q                       806 #
D&RC[]_PC                     807 #
D&VA_ALU                      808 #
D&VA_D+LC                     809 #
D&VA_D+Q                      810 #
D&VA_D-K[]                    811 #
D&VA_LA                       812 #
D&VA_LB                       813 #
D&VA_Q                        814 #
D&VA_Q+LB.PC                  815 #
D(1)?                         1649 #
D.B0?                         1650 #
D.B1?                         1651 #
```

Cross Reference Listing - Macro Names

```
D.B2?                                1652 #
D.BYTES?                             1653 #
D.NE.0?                              1654 #
D0?                                  1655 #
D2-0?                                1656 #
D2?                                  1657 #
D3-0?                                1658 #
D31?                                 1659 #
D3?                                  1660 #
DATA.TYPE?                           1661 #
DBL?                                 1662 #
DC]_CACHE                            817 #   1794
DC]_CACHE.IBCHK                      818 #
DC]_CACHE.LK                         819 #
DC]_CACHE.NOCHK                      820 #
DC]_CACHE.P                          821 #
DC]_CACHE.WCHK                       822 #
D_0                                  824 #
D_0+KC]+1                            825 #
D_0+LC+1                             826 #
D_0-D                                827 #
D_0-KC]                              828 #
D_0-Q                                829 #
D_0-Q-1                              830 #
D_ACCEL%SYNC                         831 #
D_ALU                                832 #
D_ALU(FRAC)                          833 #
D_ALU.LEFT                           834 #
D_ALU.LEFT2                          835 #
D_ALU.LEFT3                          836 #
D_ALU.RIGHT                          837 #
D_ALU.RIGHT2                         838 #
D_BLANK                              839 #
D_CACHE.INST.DEP                     840 #
D_CACHE.LKC]                         841 #
D_CACHE.WCHKC]                       842 #
D_CACHEC]                            843 #
D_D(FRAC)                            844 #
D_D+KC]                              845 #
D_D+KC]+1                            846 #
D_D+LB                               847 #
D_D+LC                               848 #
D_D+LC+PSL.C                         849 #
D_D+Q                                850 #
D_D+Q+1                              851 #
D_D-KC]                              852 #
D_D-LC                               853 #
D_D-Q                                854 #
D_D-Q-1                              855 #
D_D.OXTC]                            856 #
D_D.OXTC]+KC]                        857 #
D_D.OXTC]+Q                          858 #
D_D.OXTC]+Q+1                        859 #
D_D.OXTC].ANDNOT.KC]                 860 #
D_D.OXTC].OR.Q                       861 #
```

Cross Reference Listing - Macro Names

```
; OLDSAM.MCR                MICRO2  1L(02)    18-JAN-82  16:19:40                Page   49
;                           Cross Reference Listing - Macro Names

D_NOT.MASK                    917 #
D_NOT.Q                       918 #
D_NOT.RC]                     919 #
D_PACK.FP                     920 #
D_PACK.FP.LEFT                921 #
D_PC                          922 #
D_PC.LEFT                     923 #
D_Q                           924 #
D_Q(FRAC)                     925 #
D_Q+D                         926 #
D_Q+KC]                       927 #
D_Q+LB                        928 #
D_Q+PC                        929 #
D_Q-D                         930 #
D_Q-D-1                       931 #
D_Q-KC]                       932 #
D_Q-KC]-1                     933 #
D_Q-PCSV                      934 #
D_Q.OXTC]                     935 #
D_Q.AND.KC]                   936 #
D_Q.AND.LC                    937 #
D_Q.AND.MASK                  938 #
D_Q.AND.RCC]                  939 #
D_Q.ANDNOT.D                  940 #
D_Q.ANDNOT.KC]                941 #
D_Q.ANDNOT.MASK               942 #
D_Q.ANDNOT.PSWC               943 #
D_Q.ANDNOT.PSWN               944 #
D_Q.ANDNOT.PSWZ               945 #
D_Q.LEFT                      946 #
D_Q.OR.KC]                    947 #
D_Q.OR.PSWC                   948 #
D_Q.OR.RCC]                   949 #
D_Q.ORNOT.MASK                950 #
D_Q.RIGHT                     951 #
D_Q.RIGHT2                    952 #
D_Q.SXTC]                     953 #
D_Q.XOR.RCC]                  954 #
D_QC]D                        955 #
D_QC]KC]                      956 #
D_QC]MASK                     957 #
D_R(PRN+1)                    958 #
D_R(SC)                       959 #
D_R(SP1+1)                    960 #
D_RC(SC)                      961 #
D_RCC]                        962 #
D_RLOG                        963 #
D_RLOG.RIGHT                  964 #
D_RC]                         965 #    1767
D_RC](FRAC)                   966 #
D_RC].AND.KC]                 967 #
D_RC].OR.KC]                  968 #
D_RC].ORNOT.KC]               969 #
E.FORK                       1562 #
EALU.N?                      1664 #
```

```
EALU.Z?                          1665 #
EALU?                            1666 #
EALU_D(EXP)                      971 #
EALU_FE                          972 #
EALU_K[]                         973 #
EALU_R[](EXP)                    974 #
EALU_SC                          975 #
EALU_SC+FE                       976 #
EALU_SC+K[]                      977 #
EALU_SC-FE                       978 #
EALU_SC-K[]                      979 #
EALU_SC.ANDNOT.K[]               980 #
EALU_STATE                       981 #
END.DP1?                         1667 #
EXCEPT.ACK                       1563 #
FE&SC_K[]                        983 #
FE_0(A)                          984 #
FE_D(EXP)                        985 #
FE_EALU                          986 #
FE_K[]                           987 #
FE_LA(EXP)                       988 #
FE_NABS(SC-FE)                   989 #
FE_NABS(SC-LA(EXP))              990 #
FE_Q(EXP)                        991 #
FE_R[](EXP)                      992 #
FE_SC                            993 #
FE_SC+1                          994 #
FE_SC+FE                         995 #
FE_SC+K[]                        996 #
FE_SC+LA(EXP)                    997 #
FE_SC-FE                         998 #
FE_SC-K[]                        999 #
FE_SC-LA(EXP)                    1000 #
FE_SC-SHF.VAL                    1001 #
FE_SC.ANDNOT.FE                  1002 #
FE_SC.ANDNOT.K[]                 1003 #
FE_SC.OR.K[]                     1004 #
FE_SHF.VAL                       1005 #
FE_STATE                         1006 #
FLUSH.IB                         1565 #
FPD?                             1669 #
G.FORK                           1567 #
IB.TEST?                         1671 #
ID(SC)_D                         1008 #
ID[]_D                           1009 #
ID_D&NO.SYNC                     1010 #
ID_D.SYNC                        1011 #
INHIBIT.IB                       1569 #
INT?                             1672 #  1784
INTERRUPT.REQ?                   1673 #
INTRPT.ACK                       1570 #
INTRPT.STROBE                    1571 #  1776    1837
IR0.C31?                         1674 #
IR0?                             1675 #
IR1?                             1676 #
```

```
; OLDSAM.MCR                MICRO2  1L(02)    18-JAN-82  16:19:40                Page   51
;                           Cross Reference Listing - Macro Names

IR2-1?                      1677 #
IRD                         1572 #
IRD.11                      1573 #
IRD0                        1574 #
IRD1                        1575 #
KC]                         1013 #
LAB_R(DST)                  1015 #
LAB_R(PRN)                  1016 #
LAB_R(PRN+1)                1017 #
LAB_R(SC)                   1018 #
LAB_R(SP1)                  1019 #
LAB_R(SP1+1)                1020 #
LAB_R1&RC[]_0               1021 #
LAB_R1&RC[]_0+LC+1          1022 #
LAB_R1&RC[]_0-D             1023 #
LAB_R1&RC[]_ALU             1024 #  1771
LAB_R1&RC[]_ALU.RIGHT2      1025 #
LAB_R1&RC[]_D+LC            1026 #
LAB_R1&RC[]_D.OXT[]+KC]     1027 #
LAB_R1&RC[]_Q-KC]           1028 #
LAB_RC]                     1029 #
LAST.REF?                   1679 #
LA_R(DST)&LB_R(SRC)         1031 #
LA_R(SP2)&LB_R(SP1)         1032 #
LA_RAC]                     1033 #  1808
LC_RC(SC)                   1034 #
LC_RC[]                     1035 #  1783
LC_RC[]&R1_(LA+LB).LEFT     1036 #
LC_RC[]&R1_(LA+LB+PSL.C).LEFT  1037 #
LC_RC[]&R1_(LA+LB.RLOG).LEFT   1038 #
LC_RC[]&R1_(LA-LB).LEFT     1039 #
LC_RC[]&R1_(LA-LB.RLOG).LEFT   1040 #
LC_RC[]&R1_ALU              1041 #
LC_RC[]&R1_D                1042 #
LC_RC[]&R1_LA+KC]           1043 #
LC_RC[]&R1_LA-KC]           1044 #
LC_RC[]&R1_LB               1045 #
LC_RC[]&R1_Q                1046 #
LOAD.ACC.CC                 1577 #
LOAD.IB                     1578 #
LOAD.IB.11                  1579 #
LONG                        1580 #  1788    1807
MEMORY.NOP                  1582 #
MODE.LSS.ASTLVL?            1681 #
MUL.0XT                     1583 #
MUL.1XT                     1584 #
MUL?                        1682 #
MULM.DONE                   1585 #
MULP.DONE                   1586 #
N&Z_ALU                     1048 #
N&Z_ALU.V&C_0               1049 #
NEST.ERR?                   1684 #
N_AMX.Z_TST                 1050 #
PC&VA_ALU                   1052 #
PC&VA_D                     1053 #
```

```
; OLDSAM.MCR              MICRO2  1L(02)    18-JAN-82  16:19:40                    Page   52
;                         Cross Reference Listing - Macro Names

PC&VA_D+K[]                 1054 #
PC&VA_D-K[]                 1055 #
PC&VA_D-PC                  1056 #
PC&VA_D.OXT[]               1057 #
PC&VA_D.OXT[]+PC            1058 #
PC&VA_D.SXT[]+PC            1059 #
PC&VA_K[]                   1060 #
PC&VA_PC                    1061 #
PC&VA_Q                     1062 #
PC&VA_Q+PC                  1063 #
PC&VA_Q-D                   1064 #
PC&VA_Q-K[]                 1065 #
PC&VA_Q.SXT[]+PC            1066 #
PC&VA_RC[]                  1067 #
PC&VA_R[].ANDNOT.K[]        1068 #
PC.MODES?                   1686 #
PC_PC+1                     1070 #  1789    1832
PC_PC+2                     1071 #
PC_PC+4                     1072 #
PC_PC+N                     1073 #
PC_Q+PC                     1074 #
PC_VA                       1075 #
PC_VIBA                     1076 #
POLY.DONE                   1588 #
PSL.C?                      1687 #
PSL.CC?                     1688 #
PSL.MODE?                   1689 #
PSL.N?                      1690 #
PSL.V?                      1691 #
PSL.Z?                      1692 #
PSL<C>_AMX0                 1077 #
PTE.VALID?                  1693 #
Q&VA_ALU                    1079 #
Q&VA_D                      1080 #
Q&VA_D+LC                   1081 #
Q&VA_LA                     1082 #
Q&VA_Q+LB.PC                1083 #
Q31?                        1695 #
QD_(Q+LB)D.RIGHT2           1085 #
QD_(Q+LC)D.RIGHT2           1086 #
QD_(Q-LB)D.RIGHT2           1087 #
QD_(Q-LC)D.RIGHT2           1088 #
QD_QD.RIGHT2                1089 #
QUAD?                       1696 #
Q_(LA+Q).RIGHT              1091 #  1775
Q_(Q+LB).RIGHT              1092 #  1836
Q_0                         1093 #
Q_0+LC+1                    1094 #
Q_0+MASK+1                  1095 #
Q_0+PC.RLOG                 1096 #
Q_0-D                       1097 #
Q_0-K[]                     1098 #
Q_0-LC                      1099 #
Q_0-Q                       1100 #
Q_ACCEL&SYNC                1101 #
```

```
Q_ALU                          1102 #
Q_ALU(FRAC)                    1103 #
Q_ALU.LEFT                     1104 #
Q_ALU.LEFT2                    1105 #
Q_ALU.LEFT3                    1106 #
Q_ALU.RIGHT                    1107 #
Q_ALU.RIGHT2                   1108 #
Q_D                            1109 #
Q_D(FRAC)(B)                   1110 #
Q_D+KE]                        1111 #
Q_D+KE]+1                      1112 #
Q_D+KE].LEFT                   1113 #
Q_D+LC                         1114 #
Q_D-KE]                        1115 #
Q_D-LC                         1116 #
Q_D-Q                          1117 #
Q_D.OXTE]                      1118 #
Q_D.OXTE]+KE].LEFT             1119 #
Q_D.OXTE].OR.PACK.FP           1120 #
Q_D.AND.KE]                    1121 #
Q_D.AND.KE].RIGHT              1122 #
Q_D.AND.KE].RIGHT2             1123 #
Q_D.AND.RCE]                   1124 #
Q_D.ANDNOT.RCE]                1125 #
Q_D.LEFT3                      1126 #
Q_D.OR.KE]                     1127 #
Q_D.OR.RCE]                    1128 #
Q_D.RIGHT                      1129 #
Q_D.RIGHT2                     1130 #
Q_D.SXTE]                      1131 #
Q_D.XOR.Q                      1132 #
Q_DEC.CON                      1133 #
Q_IB.BDEST                     1134 #
Q_IB.DATA                      1135 #
Q_ID(SC)                       1136 #
Q_IDE]                         1137 #
Q_KE]                          1138 #
Q_KE]+1                        1139 #
Q_KE].CTX                      1140 #
Q_KE].RIGHT                    1141 #
Q_KE].RIGHT2                   1142 #
Q_LA                           1143 #
Q_LA+KE]                       1144 #
Q_LA+Q                         1145 #
Q_LA-KE]                       1146 #
Q_LA.AND.KE]                   1147 #
Q_LA.ANDNOT.RCE]               1148 #
Q_LB                           1149 #
Q_LC                           1150 #
Q_NOT.Q                        1151 #
Q_NOT.RE]                      1152 #
Q_PACK.FP                      1153 #
Q_PC                           1154 #
Q_Q(FRAC)                      1155 #
Q_Q(FRAC)(B)                   1156 #
```

```
; OLDSAM.MCR              MICRO2  1L(02)    18-JAN-82  16:19:40                    Page   54
;                         Cross Reference Listing - Macro Names

Q_Q+D                     1157 #
Q_Q+K[]                   1158 #  1820
Q_Q+K[]+1                 1159 #
Q_Q+LC                    1160 #
Q_Q+PC                    1161 #
Q_Q-D                     1162 #
Q_Q-D-1                   1163 #
Q_Q-K[]                   1164 #  1825
Q_Q-K[]-1                 1165 #
Q_Q-LC                    1166 #
Q_Q-LC-1                  1167 #
Q_Q-MASK-1                1168 #
Q_Q.0XT[]-K[]             1169 #
Q_Q.0XT[].LEFT            1170 #
Q_Q.0XT[].OR.D            1171 #
Q_Q.AND.K[]               1172 #
Q_Q.AND.K[].RIGHT         1174 #
Q_Q.AND.K[].RIGHT2        1173 #
Q_Q.AND.RC[]              1176 #
Q_Q.AND.R[]               1175 #
Q_Q.ANDNOT.D              1177 #
Q_Q.ANDNOT.K[]            1178 #  1781
Q_Q.ANDNOT.RC[]           1179 #
Q_Q.LEFT                  1180 #
Q_Q.LEFT2                 1181 #
Q_Q.OR.K[]                1182 #
Q_Q.ORNOT.MASK            1183 #
Q_Q.RIGHT                 1184 #
Q_Q.RIGHT2                1185 #
Q_Q.SXT[]                 1186 #
Q_Q.XOR.K[]               1187 #
Q_R(PRN).ANDNOT.Q         1188 #
Q_R(PRN+1)                1189 #
Q_R(PRN+1).AND.Q          1190 #
Q_R(SC)                   1191 #
Q_R(SRC!1).AND.K[]        1192 #
Q_RC(SC)                  1193 #
Q_RC[]                    1194 #
Q_RC[](FRAC)              1195 #
Q_R[]                     1196 #  1763
Q_R[](FRAC)               1197 #
Q_R[].AND.K[]             1198 #
Q_R[].AND.K[].RIGHT       1199 #
Q_R[].ANDNOT.K[]          1200 #
Q_R[].OR.K[]              1201 #
Q_SC                      1202 #
Q_SHF                     1203 #
R(DST)_ALU                1205 #
R(DST)_D                  1206 #
R(DST)_D.SXT[].RIGHT      1207 #
R(PRN)_0+D.RLOG           1209 #
R(PRN)_ALU                1210 #
R(PRN)_D                  1211 #
R(PRN)_D+K[].RLOG         1212 #
R(PRN)_D-K[].RLOG         1213 #
```

```
; OLDSAM.MCR              MICRO2  1L(02)    18-JAN-82  16:19:40          Page   55
;                         Cross Reference Listing - Macro Names

R(PRN)_D.OR.Q                1214 #
R(PRN)_D[]Q                  1215 #
R(PRN)_K[]                   1216 #
R(PRN)_LA+K[].RLOG           1217 #
R(PRN)_LA+Q                  1218 #
R(PRN)_LA-K[].RLOG           1219 #
R(PRN)_LA[]MASK              1220 #
R(PRN)_LC                    1221 #
R(PRN)_PACK.FP               1222 #
R(PRN)_Q                     1223 #
R(PRN)_Q+K[].RLOG            1224 #
R(PRN)_Q-K[].RLOG            1225 #
R(PRN+1)_ALU                 1226 #
R(PRN+1)_D                   1227 #
R(PRN+1)_D.OR.Q              1228 #
R(PRN+1)_K[]                 1229 #
R(PRN+1)_LA                  1230 #
R(PRN+1)_LC                  1231 #
R(PRN+1)_Q                   1232 #
R(SC)_ALU                    1234 #
R(SC)_D                      1235 #
R(SC)_K[]                    1236 #
R(SC)_LA                     1237 #
R(SC)_LA+D                   1238 #
R(SC)_LA-D                   1239 #
R(SC)_LC                     1240 #
R(SC)_Q                      1241 #
R(SP1)_ALU                   1243 #
R(SP1)_D                     1244 #
R(SP1)_K[]                   1245 #
R(SP1)_PACK.FP               1246 #
R(SP1)_Q                     1247 #
R(SP1+1)_LC                  1248 #
R(SP1+1)_Q                   1249 #
R(SRC!1)_ALU                 1251 #
R(SRC!1)_D(B)                1252 #
R(SRC)_ALU                   1253 #
R(SRC)_D                     1254 #
R(SRC)_D(B)                  1255 #
R(SRC)_D+K[].RLOG            1256 #
R(SRC)_D-K[].RLOG            1257 #
R(SRC)_LC                    1258 #
R(SRC)_Q                     1259 #
R6_D+K[].RLOG                1261 #
R6_LA+K[].RLOG               1262 #
R6_LA-K[].RLOG               1263 #
RC(SC)_0-LC                  1265 #
RC(SC)_ALU                   1266 #
RC(SC)_ALU.RIGHT             1267 #
RC(SC)_D                     1268 #
RC(SC)_Q                     1269 #
RC[]&VA_D+Q                  1271 #
RC[]_0                       1272 #
RC[]_0+K[]+1                 1273 #
RC[]_0+LC+1                  1274 #
```

Cross Reference Listing - Macro Names

RC[]_0+MASK+1	1275	#
RC[]_0+MASK+1.RIGHT2	1276	#
RC[]_0-D	1277	#
RC[]_ALU	1278	#
RC[]_ALU.LEFT	1279	#
RC[]_ALU.LEFT2	1280	#
RC[]_ALU.LEFT3	1281	#
RC[]_ALU.RIGHT	1282	#
RC[]_ALU.RIGHT2	1283	#
RC[]_D	1284	#
RC[]_D(B)	1285	#
RC[]_D+KC]	1286	#
RC[]_D-KC]	1287	#
RC[]_D.OXTC]	1288	#
RC[]_D.AND.KC]	1289	#
RC[]_D.AND.MASK	1290	#
RC[]_D.ANDNOT.Q	1291	#
RC[]_D.CTX	1292	#
RC[]_D.LEFT	1293	#
RC[]_D.LEFT3	1294	#
RC[]_D.OR.KC]	1295	#
RC[]_D.OR.Q	1296	#
RC[]_D.ORNOT.KC]	1297	#
RC[]_D.SXTC]	1298	#
RC[]_KC]	1299	#
RC[]_KC]+1	1300	#
RC[]_KC].LEFT2	1301	#
RC[]_KC].LEFT3	1302	#
RC[]_KC].RIGHT2	1303	#
RC[]_LA	1304	#
RC[]_LA+LB.CTX	1305	#
RC[]_LA-KC]	1306	#
RC[]_LA.AND.KC]	1307	#
RC[]_LA.CTX	1308	#
RC[]_LB	1309	#
RC[]_LB.LEFT	1310	#
RC[]_LC	1311	#
RC[]_NOT.Q	1312	#
RC[]_PACK.FP	1313	#
RC[]_PC	1314	#
RC[]_Q	1315	#
RC[]_Q+1	1316	#
RC[]_Q+KC]	1317	#
RC[]_Q+LC	1318	#
RC[]_Q+PC	1319	#
RC[]_Q+PC+1	1320	#
RC[]_Q-KC]	1321	#
RC[]_Q-LC	1322	#
RC[]_Q-MASK-1	1323	#
RC[]_Q.OXTC]	1324	#
RC[]_Q.AND.KC]	1325	#
RC[]_Q.ANDNOT.KC]	1326	#
RC[]_Q.LEFT	1327	#
RC[]_Q.LEFT3	1328	#
RC[]_Q.RIGHT	1329	#

Cross Reference Listing - Macro Names

```
RC[]_Q.RIGHT2                   1330 #
RC[]_Q.SXT[]                    1331 #
RC[]_RLOG.RIGHT                 1332 #
RETURN0                         1590 #
RETURN1                         1591 #
RETURN10                        1592 #
RETURN100                       1593 #
RETURN10C                       1594 #
RETURN10E                       1595 #
RETURN12                        1596 #
RETURN18                        1597 #
RETURN1F                        1598 #
RETURN2                         1599 #
RETURN20                        1600 #
RETURN24                        1601 #
RETURN3                         1602 #
RETURN4                         1603 #
RETURN40                        1604 #
RETURN60                        1605 #
RETURN61                        1606 #
RETURN8                         1607 #
RETURN9                         1608 #
RETURNF                         1609 #
RETURN[]                        1610 #
RLOG.EMPTY?                     1698 #
ROR?                            1699 #
RC[]&VA_LA+KC[]                 1334 #
RC[]&VA_LA-KC[]                 1335 #
RC[]&VA_LA-KC[].RLOG            1336 #
RC[]&VA_Q-KC[]                  1337 #
RC[]_0                          1338 #
RC[]_0+LB+1                     1339 #
RC[]_0-1                        1340 #
RC[]_0-D                        1341 #
RC[]_0-KC[]                     1342 #
RC[]_0-LB                       1343 #
RC[]_0-Q                        1344 #
RC[]_ALU                        1345 #  1821    1826
RC[]_ALU.LEFT                   1346 #
RC[]_ALU.LEFT3                  1347 #
RC[]_ALU.RIGHT                  1348 #
RC[]_ALU.RIGHT2                 1349 #
RC[]_D                          1350 #
RC[]_D+KC[]                     1351 #
RC[]_D+Q                        1352 #
RC[]_D+Q+1                      1353 #
RC[]_D-KC[]                     1354 #
RC[]_D-LC-1                     1355 #
RC[]_D-Q                        1356 #
RC[]_D.AND.KC[]                 1357 #
RC[]_D.OR.LC                    1358 #
RC[]_D.OR.PACK.FP               1359 #
RC[]_D.OR.Q                     1360 #
RC[]_KC[]                       1361 #
RC[]_LA                         1362 #
```

```
; OLDSAM.MCR                MICRO2  1L(02)    18-JAN-82  16:19:40                    Page   58
;                           Cross Reference Listing - Macro Names

RC]_LA+D                      1363 #
RC]_LA+D+1                    1364 #
RC]_LA+KC]                    1365 #
RC]_LA+KC]+1                  1366 #
RC]_LA+KC].RLOG               1367 #
RC]_LA+LC                     1368 #
RC]_LA+MASK+1                 1369 #
RC]_LA+Q                      1370 #
RC]_LA-D                      1371 #
RC]_LA-KC]                    1372 #
RC]_LA-KC].RLOG               1373 #
RC]_LA-MASK-1                 1374 #
RC]_LA-Q                      1375 #
RC]_LA.AND.KC]                1376 #
RC]_LA.OR.D                   1377 #
RC]_LA.ORNOT.MASK             1378 #
RC]_LB                        1379 #
RC]_LC                        1380 #
RC]_LC.RIGHT                  1381 #
RC]_NOT.D                     1382 #
RC]_NOT.D                     1383 #
RC]_NOT.MASK                  1384 #
RC]_NOT.Q                     1385 #
RC]_PACK.FF                   1386 #
RC]_Q                         1387 #  1830
RC]_Q+1                       1388 #
RC]_Q+5                       1389 #
RC]_Q+KC]                     1390 #
RC]_Q+LB                      1391 #
RC]_Q+LC                      1392 #
RC]_Q-D                       1393 #
RC]_Q-D-1                     1394 #
RC]_Q-KC]                     1395 #
RC]_Q-KC].RLOG                1396 #
RC]_Q-LC                      1397 #
RC]_Q.AND.KC]                 1398 #
RC]_Q.ANDNOT.KC]              1399 #
RC]_Q.OR.D                    1400 #
RC]_Q.ORNOT.KC]               1401 #
RC]_Q.RIGHT.1                 1402 #
RC]_RLOG.RIGHT.1              1403 #
SC&STATE_STATE-RC](EXP)       1405 #
SC.GT.0?                      1701 #
SC.NE.0?                      1702 #
SC?                           1703 #
SC_0(A)                       1406 #
SC_0-KC]                      1407 #
SC_ALU                        1408 #
SC_ALU(EXP)                   1409 #
SC_D                          1410 #
SC_D(EXP)                     1411 #
SC_D(EXP)(A)                  1412 #
SC_D(EXP)(B)                  1413 #
SC_D-KC]                      1414 #
SC_D.0XTC]-KC]                1415 #
```

```
; OLDSAM.MCR              MICRO2  1L(02)    18-JAN-82  16:19:40          Page  59
;                         Cross Reference Listing - Macro Names

SC_D.OXT[].XOR.K[]          1416 #
SC_D.AND.K[]                1417 #
SC_D.OR.K[]                 1418 #
SC_D.SXT[]                  1419 #
SC_EALU                     1420 #
SC_FE                       1421 #
SC_K[]                      1422 #
SC_K[].ALU                  1423 #
SC_LA                       1424 #
SC_LA.AND.K[]               1425 #
SC_LC(EXP)                  1426 #
SC_NABS(SC-FE)              1427 #
SC_PSLADDR                  1428 #
SC_Q                        1429 #
SC_Q(EXP)                   1430 #
SC_Q(EXP)(B)                1431 #
SC_Q+K[]                    1432 #
SC_Q-K[]                    1433 #
SC_Q.AND.K[]                1434 #
SC_Q.OR.K[]                 1435 #
SC_Q.SXT[]                  1436 #
SC_RC[]                     1437 #
SC_RC[](EXP)                1438 #
SC_R[]                      1439 #
SC_R[](EXP)                 1440 #
SC_R[].AND.K[]              1441 #
SC_SC+1                     1442 #
SC_SC+EXP(Q)(A)             1443 #
SC_SC+FE                    1444 #
SC_SC+K[]                   1445 #
SC_SC+SHF.VAL               1446 #
SC_SC-FE                    1447 #
SC_SC-K[]                   1448 #
SC_SC-SHF.VAL               1449 #
SC_SC.ANDNOT.FE             1450 #
SC_SC.ANDNOT.K[]            1451 #
SC_SC.OR.K[]                1452 #
SC_SHF.VAL                  1453 #
SC_STATE                    1454 #
SC_STATE.ANDNOT.K[]         1455 #
SC_STATE.OR.K[]             1456 #
SD_NOT.SD                   1457 #
SD_SS                       1458 #
SET.CC(BYTE)                1612 #
SET.CC(INST)                1613 #
SET.CC(LONG)                1614 #
SET.CC(ROR)                 1615 #
SET.CC(WORD)                1616 #
SET.FPD                     1617 #
SET.NEST.ERR                1618 #
SET.PSL.C(AMX)              1619 #
SET.V                       1620 #
SIGNS?                      1704 #
SPEC                        1621 #
SPECG                       1622 #
```

```
SRC.PC?                          1705 #
SS?                              1706 #
SS_0&SD_0                        1459 #
SS_ALU15                         1460 #
SS_SD                            1461 #
SS_SS.XOR.ALU15&SD_ALU15         1462 #
START.IB                         1623 #
STATE(7)?                        1707 #
STATE0?                          1708 #  1777    1838
STATE1-0?                        1709 #
STATE1?                          1710 #
STATE2?                          1711 #
STATE3-0?                        1712 #
STATE3?                          1713 #
STATE4?                          1714 #
STATE5?                          1715 #
STATE6?                          1716 #
STATE7-4?                        1717 #
STATE_0(A)                       1463 #
STATE_AMX.EXP                    1464 #
STATE_D(EXP)                     1465 #
STATE_FE                         1466 #
STATE_FIRST                      1467 #
STATE_INNEROBJ                   1468 #
STATE_INNERSRC                   1469 #
STATE_K[]                        1470 #  1764    1815
STATE_OUTER                      1471 #
STATE_PREDEC                     1472 #
STATE_Q(EXP)                     1473 #
STATE_SC.VIA.KMX                 1474 #
STATE_SKPLONG                    1475 #
STATE_STATE+1                    1476 #
STATE_STATE+FE                   1477 #
STATE_STATE+K[]                  1478 #
STATE_STATE-FE                   1479 #
STATE_STATE-K[]                  1480 #
STATE_STATE.AN.5TO0              1482 #
STATE_STATE.AN.6TO4              1483 #
STATE_STATE.AN.DESTDBL           1484 #
STATE_STATE.AN.NOTPREDEC         1485 #
STATE_STATE.AN.PREDECZERO        1486 #
STATE_STATE.AN.SKPLONG           1481 #
STATE_STATE.ANDNOT.FE            1487 #
STATE_STATE.ANDNOT.K[]           1488 #
STATE_STATE.ANDNOT.SHF.VAL       1489 #
STATE_STATE.OR.ADJINP            1492 #
STATE_STATE.OR.DEST              1493 #
STATE_STATE.OR.DESTDBL           1494 #
STATE_STATE.OR.FE                1490 #
STATE_STATE.OR.FILL              1495 #
STATE_STATE.OR.FLOAT             1496 #
STATE_STATE.OR.K[]               1491 #
STATE_STATE.OR.MOVE              1497 #
STATE_STATE.OR.PATT1             1498 #
STATE_STATE.OR.PATT2             1499 #
```

```
; OLDSAM.MCR              MICRO2  1L(02)    18-JAN-82  16:19:40                    Page   61
;                         Cross Reference Listing - Macro Names

STOP.IB                    1624 #
SWAPD                      1500 #
TB.TEST?                   1719 #
TEST.TB.RCHK               1626 #
TEST.TB.WCHK               1627 #
TRAP.ACC[]                 1628 #
VA31-30?                   1721 #
VA31?                      1722 #
VA_ALU                     1502 #  1782
VA_D                       1503 #
VA_D+K[]                   1504 #
VA_D+LC                    1505 #
VA_D+Q                     1506 #
VA_D.OXT[]+Q               1507 #
VA_D.ANDNOT.K[]            1508 #
VA_K[]                     1509 #
VA_LA                      1510 #
VA_LA+D                    1511 #
VA_LA+K[]                  1512 #
VA_LA+K[]+1                1513 #
VA_LA+PC                   1514 #
VA_LA+Q                    1515 #
VA_LA-D                    1516 #
VA_LA-K[]                  1517 #
VA_LA-K[]-1                1518 #
VA_LA-Q                    1519 #
VA_LA.AND.LC               1520 #
VA_LA.ANDNOT.K[]           1521 #
VA_LB+D.OXT                1522 #
VA_PC                      1523 #
VA_Q                       1524 #
VA_Q+D                     1525 #
VA_Q+K[]                   1526 #
VA_Q+LB                    1527 #
VA_Q+LB.PC                 1528 #
VA_Q+LC                    1529 #
VA_Q+PC                    1530 #
VA_Q-K[]                   1531 #
VA_Q-LB                    1532 #
VA_Q.ANDNOT.K[]            1533 #
VA_RC[]                    1534 #
VA_R[]                     1535 #
VA_VA+4                    1536 #
WORD                       1630 #
WRITE.DEST                 1631 #
WRITE.Q.DEST               1632 #
Z?                         1724 #  1809
ZONED?                     1725 #
```

Cross Reference Listing - Expression Names

```
; OLDSAM.MCR                MICRO2  1L(02)    18-JAN-82  16:19:40                    Page  63
;                           Location / Line Number Index

;Location       0/8     1/9     2/A     3/B     4/C     5/D     6/E     7/F

U 0000 - 13FF Unused
U 1400          1784=   1790=   1812=   1816=   1764    1767    1796=   1799=
U 1408          1771    1777    1822=   1827=   1809    1838            1833=
```

```
; OLDSAM.MCR                MICRO2  1L(02)    18-JAN-82  16:19:40                Page   64
;                           U-code Microword Summary

              BSERCH  Words not
            1400-17FF  in bounds
VAXDEF             0          0
BSERCH            15          0

Used              15          0
Remaining       1009

Total microwords used in memory U: 15
Total microwords remaining in memory U: 1009
Highest address used in memory U: 140F (hex)
```

```
; OLDSAM.MCR                    MICRO2  1L(02)     18-JAN-82  16:19:40                    Page   65
;                               Error Summary

Pass 1 warnings:     0   Pass 2 warnings:     0
Pass 1 errors:       0   Pass 2 errors:       0
```

C.3 THE OBJECT FILE (.ULD)

```
;RTOL
;RADIX 16
[1404]=0000003C19C0FA1014047405
[1405]=0800003C0180FA0000001408
[1408]=0001003C0180FB0800001409
[1409]=005C171401C0F80040001400
[1400]=00192E240DC0F90802001406
[1401]=C01800381980F80440500062
[1406]=001C002001804210001014OC
[1407]=0000003C0180F800000004F8
[140C]=00110100018OF88800101402
[1402]=00001B3C0180F8000000140A
[1403]=00001B3C0580F8001404740A
[140A]=0019201411C0FA880000140D
[140B]=0019200011C0FA9000001409
[140F]=C001203C0180FA8C40500062
[140D]=004D371401C0F80040001400
FIELD ACF=<71:70>
 CONTROL=3
 NOP=0
 SYNC=1
 TRAP=2
FIELD ACM=<57:55>
 ABORT=1
 POLY.DONE=6
 PWR.UP=0
FIELD ADS=<47:47>
 IBA=1
 VA=0
FIELD ALU=<69:66>
 A=0F
 A+B=5
 A+B+1=4
 A+B+PSL.C=0B
 A+B.RLOG=6
 A-B=0
 A-B-1=2
 A-B.RLOG=1
 AND=0D
 ANDNOT=9
 B=0E
 INST.DEP=3
 NOTA=0A
 OR=0C
 ORNOT=7
 XOR=8
FIELD AMX=<81:80>
 LA=0
 RAMX=1
 RAMX.OXT=3
 RAMX.SXT=2
FIELD BEN=<76:72>
 ACCEL=6
 ALU=1B
 ALU!-0=15
```

```
 C31=3
 D.BYTES=18
 D3-0=19
 DATA.TYPE=8
 DECIMAL=0F
 EALU=12
 END.DP1=8
 IB.0=5
 IB.TEST=0B
 INTERRUPT=0E
 IR2-1=9
 IRC.ROM=4
 LAST.REF=11
 MUL=0C
 NOP=0
 PC.MODES=9
 PSL.CC=1A
 PSL.MODE=1C
 REI=0A
 ROR=2
 SC=14
 SIGNS=0D
 SRC.PC=0A
 STATE3-0=17
 STATE7-4=16
 TB.TEST=1D
 Z=1
FIELD BMX=<84:82>
 KMX=6
 LB=3
 LC=4
 MASK=0
 PACKED.FL=2
 PC=5
 PC.OR.LB=1
 RBMX=7
FIELD CCK=<22:20>
 C_AMX0=6
 INST.DEP=7
 LOAD.UBCC=1
 NOP=0
 NZ_ALU.VC_0=5
 NZ_ALU.VC_VC=6
 N_AMX.Z_TST.VC_VC=3
 ROR=4
 SET.V=2
FIELD CID=<45:42>
 ACK=5
 CONT=7
 NOP=1
 READ.KMX=0B
 READ.SC=9
 WRITE.KMX=0F
 WRITE.SC=0D
FIELD DK=<91:88>
 ACCEL=0A
 BYTE.SWAP=0B
 CLR=0F
 DAL.SC=0D
 DAL.SV=0E
 DIV=4
 LEFT=5
 LEFT2=1
```

```
 NOP=0
 Q=0C
 RIGHT=6
 RIGHT2=2
 SHF=8
 SHF.FL=9
FIELD DT=<79:78>
 BYTE=2
 INST.DEP=3
 LONG=0
 WORD=1
FIELD EALU=<15:13>
 A=0
 A+1=6
 A+B=4
 A-B=5
 ANDNOT=2
 B=3
 NABS.A-B=7
 OR=1
FIELD EBMX=<19:18>
 AMX.EXP=2
 FE=0
 KMX=1
 SHF.VAL=3
FIELD FEK=<24:24>
 LOAD=1
 NOP=0
FIELD FS=<42:42>
 CID=1
 MCT=0
FIELD IBC=<95:92>
 BDEST=7
 CLR.0=0C
 CLR.0-3=0E
 CLR.0.1=4
 CLR.1=0D
 CLR.1-5.COND=0F
 CLR.2.3=5
 FLUSH=2
 NOP=0
 START=3
 STOP=1
FIELD ID.ADDR=<63:58>
 ACC.0=14
 ACC.1=15
 ACC.2=16
 ACC.CS=17
 CES=0C
 CLK.CS=0A
 COMP=1C
 D.SV=2E
 DAY.TIME=1
 ESP=29
 FAULT=1B
 FPDA=2D
 IBUF=0
 INTERVAL=0B
 ISP=2C
 KSP=28
 MAINT=1D
 NXT.PER=9
 POBR=24
```

```
 POLR=3C
 P1BR=25
 P1LR=3D
 PARITY=1E
 PCBB=3A
 PSL=0F
 Q.SV=2F
 RXCS=4
 RXDB=5
 SBI.ERR=19
 SBR=26
 SCBB=3B
 SILO=18
 SIR=0E
 SLR=3E
 SSP=2A
 SYS.ID=3
 T0=30
 T1=31
 T2=32
 T3=33
 T4=34
 T5=35
 T6=36
 T7=37
 T8=38
 T9=39
 TBER0=12
 TBER1=13
 TBUF=10
 TIME.ADDR=1A
 TXCS=6
 TXDB=7
 UBREAK=21
 USP=2B
 USTACK=20
 VECTOR=0D
 WCS.ADDR=22
 WCS.DATA=23
FIELD IEK=<31:30>
 EACK=3
 IACK=2
 ISTR=1
 NOP=0
ADDRESS J=<12:0>
 INT.B=4F8
 IRD=62
 SRCH=1404
 SRCH.1=1409
 SRCH.2=1400
 SRCH.3=1406
 SRCH.4=140C
 SRCH.5=140A
 SRCH.6=140D
FIELD KMX=<63:58>
 .1=1
 .10=19
 .14=8
 .18=1F
 .19=2E
 .1A=39
 .1B=3B
 .1E=14
```

```
.1F=23
.1F00=24
.2=2
.20=1D
.24=3A
.28=0B
.3=3
.30=1E
.3030=32
.34=0A
.3F=15
.3FF=20
.4=4
.40=0C
.4000=2C
.50=0D
.6=35
.60=29
.7=17
.7C=27
.7E=3E
.7F=16
.7FF0=0E
.8=0
.80=10
.8000=11
.88=31
.9=36
.A=3D
.A0=9
.B0=25
.C=21
.C0=34
.D=22
.DFCF=2B
.E003=26
.EF=0F
.F=18
.F0=33
.FF=12
.FF00=13
.FFE0=28
.FFE8=1A
.FFF0=1B
.FFF1=2D
.FFF5=38
.FFF6=37
.FFF8=1C
.FFF9=2F
.FFFC=3C
.FFFF=30
SC=7
SP1.CON=5
SP2.CON=6
ZERO=6
FIELD MCT=<47:42>
 ALLOW.IB.READ=3E
 EXTWRITE.P=28
 INVALIDATE=24
 LOCKREAD.P=3A
 LOCKREAD.V.NOCHK=1A
 LOCKREAD.V.WCHK=1C
 LOCKWRITE.P=2E
```

```
  LOCKWRITE.V.XCHK=0E
  MEM.NOP=2
  READ.INT.SUM=36
  READ.P=32
  READ.V.IBCHK=16
  READ.V.NEWPC=18
  READ.V.NOCHK=12
  READ.V.RCHK=10
  READ.V.WCHK=14
  SBI.HOLD=20
  SBI.HOLD+UNJAM=22
  TEST.RCHK=0
  TEST.WCHK=4
  VALIDATE=26
  WRITE.P=2A
  WRITE.V.NOCHK=0A
  WRITE.V.WCHK=0C
FIELD MSC=<29:26>
  CHK.CHM=1
  CHK.FLT.OPR=2
  CHK.ODD.ADDR=3
  CLR.FPD=8
  CLR.NEST.ERR=0A
  INH.CM.ADDR=0F
  IRD=4
  LOAD.ACC.CC=6
  LOAD.STATE=5
  NOP=0
  READ.RLOG=7
  RETRY.NO.TRAP=0D
  RETRY.TRAP=0E
  SECOND.REF=0C
  SET.FPD=9
  SET.NEST.ERR=0B
FIELD PCK=<34:32>
  NOP=0
  PC+1=4
  PC+2=5
  PC+4=6
  PC+N=7
  PC_IBA=2
  PC_VA=1
  VA+4=3
FIELD QK=<54:51>
  ACCEL=0B
  CLR=0F
  D=0C
  DEC.CON=0A
  ID=0E
  LEFT=5
  LEFT2=1
  NOP=0
  RIGHT=6
  RIGHT2=2
  SHF=8
  SHF.FL=9
FIELD RAMX=<77:77>
  D=0
  Q=1
FIELD RBMX=<77:77>
  D=1
  Q=0
FIELD SCK=<23:23>
```

```
 LOAD=1
 NOP=0
FIELD SGN=<50:48>
 ADD.SUB=6
 CLR.SD+SS=7
 LOAD.SS=1
 NOP=0
 NOT.SD=3
 SD.FROM.SS=4
 SS.FROM.SD=2
 SS.XOR.ALU=5
FIELD SHF=<87:85>
 ALU=0
 ALU.DT=3
 LEFT=1
 LEFT3=5
 RIGHT=2
 RIGHT2=4
FIELD SI=<57:55>
 ASHL=2
 ASHR=1
 DIV=5
 DIVD=0
 MUL+=6
 MUL-=7
 ZERO=3
FIELD SMX=<17:16>
 ALU=2
 ALU.EXP=3
 EALU=0
 FE=1
FIELD SPO=<41:35>
 LOAD.LC.SC=6
 NOP=0
 WRITE.RC.SC=7
FIELD SPO.AC=<41:38>
 LOAD.LA=2
 LOAD.LAB=1
 WRITE.RAB=3
FIELD SPO.ACN=<37:35>
 PRN=3
 PRN+1=4
 SC=5
 SP1+1=6
 SP1.SP1=0
 SP2.SP1=2
 SP2.SP2=1
FIELD SPO.ACN11=<37:35>
 DST.DST=1
 DST.SRC=2
 SC=5
 SRC.OR.1=4
 SRC.SRC=0
FIELD SPO.R=<41:39>
 LOAD.LAB=4
 LOAD.LAB1.WRITE.RC=6
 LOAD.LC=2
 LOAD.LC.WRITE.RAB1=7
 WRITE.RAB=5
 WRITE.RC=3
FIELD SPO.RAB=<38:35>
 AP=0C
 FP=0D
```

```
  R0=0
  R1=1
  R15=0F
  R2=2
  R3=3
  R4=4
  R5=5
  R6=6
  R7=7
  SP=0E
 FIELD SPO.RC=<38:35>
  LC.SV=8
  MBIT.VA=0F
  PC.SV=0C
  PTE.MASK=0F
  PTE.PA=0B
  PTE.VA=0A
  SC.SV=0D
  T0=0
  T1=1
  T2=2
  T3=3
  T4=4
  T5=5
  T6=6
  T7=7
  VA.REF=0E
  VA.SV=9
 FIELD SUB=<65:64>
  CALL=1
  NOP=0
  RET=2
  SPEC=3
 FIELD VAK=<25:25>
  LOAD=1
  NOP=0
 END
```

APPENDIX D

THE TEST PROGRAM

```
        .TITLE  BSTEST - PROGRAM TO EXERCISE BSERCH TEST MICROCODE
        .PSECT  BSTEST

;Open the terminal for output
BEGIN:  PUSHL   #1                      ;Allow writes
        PUSHL   #0                      ;No name
        PUSHL   #0                      ;Name length = 0
        PUSHL   #-1                     ;TTY chanel
        CALLS   #4,FILOPN               ;Open terminal for output. 4 parameters

;Save current XFC SCB vector. Set XFC vector to 2 for access to user microcode
        $CMKRNL_S  ST_VEC               ;Change mode to kernal & set vector.
        BLBS    R0,INITA                ;Branch if no error setting vector.
        $EXIT_S R0                      ;Exit with error status.

;Initialize each longword in ARRAY with with its address.
INITA:  CLRL    R0
        MOVAL   ARRAY,R1
2$:     MOVL    R1,(R1)+
        AOBLSS  #1000,R0,2$

;Start with R0 equal to the address of ARRAY minus one. Do binary search on
;ARRAY and check that search produced the correct result. Increment R0 by one
;and re-search ARRAY, checking the results, until R0 is one more than the
;highest value in ARRAY
        MOVAL   ARRAY-1,R0              ;INIT COMPARAND
LOOP:   MOVAL   ARRAY,R1                ;LOWER BOUND
        MOVAL   ARRAY+3996,R2           ;UPPER BOUND
        .BYTE   ^XFC                    ;INVOKE THE SPECIAL MICROCODE
        BEQL    NOMCH                   ;BRANCH IF NO MATCH FOUND

;   Match. See if it should have matched.
MATCH:  BITL    #3,R0                   ;Should have matched if R0 is
        BEQL    R1CHK                   ;longword allisned.
        PUSHAB  BDFND                   ;Push address of error message
        PUSHL   BDFNDL                  ;Push length of error message
        BRW     ENDIT                   ;and so report error

;   Match. See if R1 has the correct value.
R1CHK:  CMPL    R0,(R1)                 ;SEE IF R1 HAS THE CORRECT VALUE
        BEQL    BUMP                    ;ALL OK
        PUSHAB  BDADD                   ;Push address of error message
        PUSHL   BDADDL                  ;Push length of error message
        BRW     ENDIT                   ;and so report error

;   No match. See if it should not have matched.
NOMCH:  BITL    #3,R0                   ;Should not match if R0 is not
        BNEQ    BUMP                    ;longword allisned.
        PUSHAB  NOMAT                   ;Push address of error message
        PUSHL   NOMATL                  ;Push length of error message
        BRW     ENDIT                   ;and so report error

;Increment R0 and branch to top if not done.
BUMP:   AOBLEQ  #ARRAY+3997,R0,LOOP
```

```
        .PAGE
;All done with no errors. Report successful completion.
        PUSHAB  DONE                    ;Push address of completion message
        PUSHL   DONEL                   ;Push length of completion message

;Output ending message, restore original XFC vector, and exit.
ENDIT:  PUSHL   #-1
        CALLS   #3,FILOUT
        PUSHL   #-1
        CALLS   #1,FILCLS
        $CMKRNL_S  RSTOR                ;Change mode to kernal & restore vector
        $EXIT_S R0                      ;Exit with status

;Routine to save and set a new XFC SCB vector.
ST_VEC: .WORD   0
        MOVL    EXE$GL_SCB,R4           ;R4 gets SCBB
        MOVL    ^X14(R4),OLDV           ;Save the vector at SCBB+14(hex).
        MOVL    #2,^X14(R4)             ;Make the new vector at SCBB+14(hex)=2
        MOVL    #1,R0                   ;Indicate success and
        RET                             ;return

;Routine to restore original XFC SCB vextor.
RSTOR:  .WORD   0
        MOVL    EXE$GL_SCB,R4           ;R4 gets SCBB
        MOVL    OLDV,^X14(R4)           ;Restore original vector
        MOVL    #1,R0                   ;Indicate success and
        RET                             ;return

        .PAGE
BDFND:  .ASCII  "Search reports a match when it should not."
BDFNDL: .LONG   .-BDFND

BDADD:  .ASCII  "Search reports wrong address an match."
BDADDL: .LONG   .-BDADD

NOMAT:  .ASCII  "Search does not find match when it should."
NOMATL: .LONG   .-NOMAT

DONE:   .ASCII  "BSTEST successful completion."
DONEL:  .LONG   .-DONE

.PSECT  ARRAY,LONG
OLDV:   .LONG   0
        .LONG   0
        .BLKL   1
ARRAY:  .BLKL   1000                    ;BLOCK OF 1000 LONGWORDS
        .LONG   0

.END    BEGIN
```

INDEX

VAX-11/780 MICROPROGRAMMING
TOOLS USER'S GUIDE
AA-H306B-TE

READER'S COMMENTS

NOTE: This form is for document comments only. DIGITAL will use comments submitted on this form at the company's discretion. If you require a written reply and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Did you find this manual understandable, usable, and well-organized? Please make suggestions for improvement.

Did you find errors in this manual? If so, specify the error and the page number.

Please indicate the type of reader that you most nearly represent.

- ☐ Assembly language programmer
- ☐ Higher-level language programmer
- ☐ Occasional programmer (experienced)
- ☐ User with little programming experience
- ☐ Student programmer
- ☐ Other (please specify)_____

Name_____ Date_____

Organization_____

Street_____

City_____ State_____ Zip Code or Country_____

Please cut along this line.

---------- Fold Here ----------

---------- Do Not Tear - Fold Here and Staple ----------

FIRST CLASS
PERMIT NO. 33
MAYNARD, MAS

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN THE UNITED STATES

Postage will be paid by:

digital

VAX CURRENT ENGINEERING
1925 ANDOVER STREET
TEWKSBURY, MASSACHUSETTS 01876