pdp11

THE PDP-11/60 MICROPROGRAMMING TOOLS REFERENCE MANUAL

Order No. AA-C815A-TC

digital

THE PDP-11/60 MICROPROGRAMMING TOOLS REFERENCE MANUAL

Order No. AA-C815A-TC

Edition 1

October 1977

Digital Equipment Corporation, Maynard, Massachusetts, 01754

THE PDP-11/60 MICROPROGRAMMING TOOLS REFERENCE MANUAL

CONTENTS

NOTE: Each of the parts is immediately preceded by a detailed table of contents for the part.

PREFACE

This manual describes the tools that are provided with the Writable Control Store option for the PDP-11/60. The manual gives information about assembling, loading, and debugging microprograms for the 11/60.

The manual is divided into five parts. The first part introduces the microprogramming tools and discusses the syntax notation and other issues that are common to all four parts of the manual.

The next three parts of the manual describe the three tools. Part II describes the MICRO-11/60 assembler; part III, the Microprogram Loader, MLD; and part IV, the Microprogram Debugging Tool, MDT.

The fifth part of the manual contains information on the use of the tools to assemble, load, and debug a microprogram.

Five appendices, which give reference material, are included. Appendix A summarizes the syntax. Appendix B gives the 11/60 Predefinitions. Appendix C describes the dispatch file and a technique for partitioning the Writable Control Store. Appendix D provides a sample microprogram. Appendix E lists the error messages for each of the tools.

Intended Audience

This manual is directed to the experienced assembly-language programmer and to the hardware engineer with some programming experience. The user should be familiar with the basic concepts of the RSX-11M operating system described in Introduction to RSX11M (DEC-11-OM1EA-B-D) and with basic operating procedures described in the RSX-11M Operators's Procedures Manual (DEC-11-OMOGA-B-D).

Related Manual

This manual describes the microprogramming tools and their use. To understand the 11/60 microarchitecture, the following manual is provided:

PDP-11/60 Microprogramming Specification (AA-C814A-TC)

Reading the PDP-11/60 Microprogramming Specification first is suggested.

PART I

INTRODUCTION

Contents

CHAPTER 1

INTRODUCTION

The Writable Control Store is a hardware option that allows users to microprogram the 11/60 for special applications. This manual describes the three tools that are provided with the Writable Control Store option to aid the microprogrammer in writing, loading, and debugging microprograms.

After this introductory part, the next three parts of the manual described the tools, as follows:

Part II - The Micro-11/60 Assembler, which converts source microprograms into loadable object modules.

Part III - The Microprogram Loader -- MLD, which loads the object module into the Writable Control Store.

Part IV - The Microprogram Debugging Tool -- MDT, which allows the microprogrammer to examine and breakpoint microprograms running in the Writable Control Store.

Then, the last part of the manual describes the use of the three tools:

Part V - Microprogramming Tools User's Guide, which describes how to invoke and execute the tools.

Each chapter in the manual has a characteristic form. It consists of a sequence of sections that begin with an introduction to a feature, followed by a rigorous definition, followed by explanations and examples. The subsections that provide this information are given in the following list:

Sub-section	Meaning
Syntax	Defines the structure of the feature.
Interpretation	Gives a succinct, but complete, statement of the meaning of the feature, followed by a detailed discussion of aspects of the feature.
Restrictions	Provides any restrictions on the feature that are not stated in the syntax.
Defaults	Supplies the assumptions made for any cases that are identified as optional within the syntax.

The manual is organized in this way so that it can be read initially as a tutorial and then used conveniently for a reference.

1.1 THE SYNTAX NOTATION

The constructs of the languages used by the WCS tools are defined in a syntax notation. A syntactic rule defines a syntactic name in terms of a string of syntactic terms. The syntactic terms can be terminals (such as: keywords, separators, etc.), which are displayed in upper case, or other syntactic names, which are displayed in lower case. An example of a terminal is given in Section 1.1.

Syntactic rules are displayed in boxes. The box is divided into a left-side and a right-side by a vertical line. On the left-side, the syntactic name being defined is given and, on the right-side, the string that defines the name is given. For example, consider the following syntactic rule:

op-code	octal-digit

In the above rule, the syntactic name op-code is defined to be an octal-digit.

1.1.1 Concatenation

A concatenation is a sequence of two or more definitions strings, written one after another. An example of a concatenation in a syntactic rule is:

field-value-definition	field-value-name ::= value

The above syntactic rule states that a field-value-definition consists of a field-value-name followed by the characters "::=" followed by a value.

Another example of a concatenation, which includes a syntactic terminal and a syntactic name, is the following:

title-line	.TITLE title-string

The above rule states that a title-string consists of the keyword .TITLE followed by a title-string.

1.1.2 Disjunction

A disjunction is a string definition that permits the choice from a set of possible definitions. A disjunction is written within curly braces, with each possibility separated from the others either by being on a separate line or by a vertical bar character.

An example of a disjunction in which each choice is written on a separate line is:

field-setting	field-name / { field-value-name / value }

The above rules states that a field-setting consists of a field-name followed by a "/" character followed by either a field-value-name or a value.

An example of a disjunction in which the choices are separated by vertical bar characters is:

octal-digit	{ 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 }

The above rule states that an octal-digit can be either the character 0, or the character 1, or the character 2, and so on, to the character 7.

1.1.3 Replication

A replication is a string definition that can be repeated a specified number of times. Replication is indicated in the syntax by enclosing the string definition in curly braces with a subscript and superscript. The subscript indicates the minimum number of repetitions and superscript indicates the maximum number of repetitions. An example of replication in a syntactic rule is:

toc-string	{ radix-50-char }$^{64}_{1}$

The above syntactic rule states that a toc-string consists of from 1 to 64 radix-50-chars.

If the replications are separated by some character, then that character is given at the point of the curly brace as shown in the following syntactic rule:

input-spec	$\left\{ \text{input-file} \right\}_{1}^{n}$,

The above syntactic rule states that an input spec consists of one or more input-files separated by commas. So, for example, an input-spec can be any of the following:

```
file1

file1,file2

file1,file2,file3

...
```

The superscript n indicates that any number of replications can be given.

1.1.4 Omission

Omission is indicated by the use of the subscript 0 and the superscript 1, indicating that from 0 to 1 replications are possible. An example of omission is given in the following syntactic rule:

constraint	mask $\left\{ \text{[low-address : high-address]} \right\}_{0}^{1}$

The above rule states that a constraint consists of a mask, optionally followed by the string " [low-address : high-address] ".

1.2 EXAMPLES

Two kinds of examples are used in this manual, namely: abstract and concrete. An abstract example is written with non-mnemonic names (e.g. A, B, ALPHA, BETA, etc) and is used to illustrate a feature when the scope of the feature is such that using a concrete example would be distracting. A concrete example is an actual piece of a microprogram. When a concrete example is used, the context of that example is usually referenced. The manual contains three complete microprograms and concrete examples are drawn from these microprograms.

PART II

THE MICORPROGRAM ASSEMBLER: MICRO-11/60

Contents

CHAPTER 2

INTRODUCTION TO MICRO-11/60

The MICRO-11/60 assembler converts microprograms written in its source language to absolute object code. The source language of MICRO-11/60 allows the symbolic definition of fields and macros and the use of these names in specifying the actions to be performed by the microprogram.

The MICRO-11/60 assembler performs two logical functions: translation and address selection. In translating names within a microinstruction to the appropriate set of bits, the assembler also performs valuable syntax and error checking. In assigning addresses, the assembler aids the programmer in laying out branches and allocating storage in an effective manner.

2.1 TRANSLATION

To construct an object microprogram, the assembler interprets a source microprogram written in a language that defines and uses names to set the appropriate bits in the microwords of the program. Names, called field-names, are defined to identify a sequence of bits within the microword. For example, bits 47 through 44 can be associated with the field-name ALU by the following field-definition:

```
.FIELD ALU ::= <47:44>
```

Names, called field-value-names, then can be defined to represent some or all of the possible field values for the field. Field-value-names are specified following a field definition by a series of name and value pairs, connected by the characters "::=". For example:

```
.FIELD  ALU ::=  <47:44>
     NOT-A  ::= 00
     A-PLUS-B-PLUS-PS[C]   ::=  01
     NOT-A-AND-B  ::= 02
     ZERO  ::=  03
     A-PLUS-B-PLUS-D[C]   ::=  04
     A-PLUS-NOT-B-PLUS-D[C]   ::= 05
     A-XOR-B ::= 06
     A-AND-NOT-B  ::= 07
     DIVIDE  ::=  10
     A-PLUS-B  ::= 11
     B  ::= 12
     A-AND-B ::= 13
     A-PLUS-B-PLUS-1  ::= 14
     A-MINUS-B ::= 15
     A-IOR-B  ::=  16
     A  ::=  17
```

Then to set bits 47 through 44 to the value 10, the microprogrammer can write the field-setting:

```
ALU/DIVIDE
```

Since microprogrammers think in terms of symbolic names rather than bits, the above notation is considerably more convenient for the writer and understandable for the reader than the equivalent:

```
<47:44>/11
```

In addition to this basic ability to refer to fields and their values symbolically, macros can be defined to produce a notation in which the functions of the microword, not the specific field-settings, are given. For example, to use the shift tree, a multiplexer selection for each stage of the shift tree must be specified. Consider the following macro:

```
.MACRO D-RIGHT-14 ::= AEN/CMUX,AMUX/RIGHT-8,BMUX/RIGHT-4,
                      ASEL/RIGHT-2
```

The function performed by this macro is the shift of the D register to the right 14 places. To accomplish this, four field settings are required; however, once this macro is defined, the microprogrammer can simply write:

```
D-RIGHT-14
```

The above macro-call within a microinstruction is equivalent to setting the four fields shown in the macro definition, but is, again, more convenient and readable.

2.1.1 The 11/60 Predefinitions

The MICRO-11/60 assembler is a special version of the general assembler MICRO-11. MICRO-11/60 has been tailored for the needs of the 11/60 microprogrammer by a series of predefinitions, which define the fields of the 11/60 microword and which provide a set of macros that specify the logical functions performed in executing an 11/60 microprogram.

For most applications, the 11/60 microprogrammer need not write any additional field or macro definitions, but can work entirely in terms of the predefinitions provided. These predefinitions have been used, within DIGITAL, for several large microprograms and, in the course of use, have been refined several times.

The philosophy of the predefinitions is described in Section 5.3 and a complete listing of the predefinitions, written in MICRO-11/60 language, is given in Appendix B. The method for incorporating the predefinitions in the microprogram is described in Chapter 14.

2.2 ADDRESS ASSIGNMENT

The 11/60 uses a chained sequencing method of addressing, as described in the "11/60 Microprogramming Specification". Each microinstruction contains the address of another microinstruction. The field that contains this address is called the Micro Pointer Field (UPF). If the microinstruction specifies an unconditional branch, then control passes to the microinstruction whose address is found in the UPF field. If the microinstruction specifies a conditional branch, then control passes to the microinstrction whose address is formed by OR-ing the output of the Branch Micro Test Multiplexer (BUT MUX) with the UPF field.

The assembler, in the absence of any direction from the microprogrammer, assigns unconditional branch addresses and, with some help, assigns conditional branch targets. The following sections describe the address space, the algorithm used by the assembler in assigning addresses and the ways in which the microprogrammer can reserve and specify addresses.

2.2.1 The Address Space

The PDP-11/60 Writable Control Store consists of two pages. The first page occupies addresses 6000 through 6777 and the second page occupies addresses 7000 through 7777. The first 200 locations of the first page, that is 6000 through 6200, are reserved for the resident section of the Writable Control Store. The resident section is described in the "PDP-11/60 Microprogramming Specification".

2.2.2 Address Assignment Algorithm

The assembler selects addresses for assignment from an available address pool that is formed by considering all the addresses that lie between the bounds specified either by the predefinitions-file (6200:7777) or, if the predefinitions-file is not included, by the bounds given by the .BOUNDS keyword in the user-machine-definition. The assembler chooses the lowest address in the available address pool, assigns it to the current microinstruction, and removes the address from the available address pool.

The assembler only uses addresses from the available address pool that are on the current page. The current page can only be changed by an explicit address assignment from the microprogrammer. Initially, the assembler establishes the current page as the page that contains the first address assigned. Suppose, for example, the address space starts at 6200 and ends at 7777. The assembler chooses 6200 for the address of the first microinstruction, and, in this way, establishes the first page of the WCS as the current page. The assembler continues assigning addresses until 6777. If the microprogrammer does not change the page by assigning an address on the second page either before or at that point, then the assembler reports an addressing error for every subsequent microinstruction and assigns an address that has already been assigned to a previous microinstruction.

2.2.3 Address Reservation

The microprogrammer can reserve a set of addresses and, in this way, remove them from the available address pool. Section 6.2 of this manual describes the mechanism for reserving and using reserved addresses, called the target assignment construct. The target assignment construct is used to lay out the branch targets for a conditional branch.

2.2.4 Address Specification

The microprogrammer can specify addresses explicitly, by simply preceding the microinstruction by an address. As noted earlier, such an explicit assignment is necessary to change the current page.

However, if an address is not available when it is specified, then an error is reported and another address selected for the microinstruction. To determine whether an address will be available, the microprogrammer must consider the number of microinstructions that precede the microinstruction, the bounds of the microprogram, and the addresses reserved by the target assignment construct. Further, any changes to the microprogram can affect the availability of a particular address. If, for example, the microprogrammer adds several instructions before the given microinstruction or moves a branch-definition to an earlier point in the program, its specified address may become unavailable in the next assembly.

2.3 ERROR DETECTION AND CORRECTION

MICRO-11/60 detects the errors described in Appendix E. Errors are caused either by invalid input or by system failures.

Some errors are more devastating than others. A system failure usually causes the processing to cease and, in such a case, no useful results are obtained. Some user input errors render the remainder of the assembly useless. Some user input errors simply render the object module invalid. However, although the resulting object module cannot be loaded and executed, the results of the assembly can be examined for other problems.

The devastating errors are usually the exception. Typically, in the course of the assembly, MICRO-11/60 detects a few trivial errors that can all be corrected before the next assembly. Moreover, MICRO-11/60 tries to continue processing and produce a useful result in all cases. If the microprogrammer specifies an impossible action, the assembler tries to counter with a possible action in the hope that the result will be useful. For example, if the user specifies an address for a microinstruction and that address has already either been reserved or used, then the assembler reports the error and assigns the next available address.

Most of the error messages are fairly self-explanatory. One of the most frequently encountered errors is Number 39 -- Syntax Error. If a syntax error is encountered in a line, usually the entire line is discarded, even though some useful information occurs before the syntactically incorrect item. When the line is discarded, sometimes other errors propagate from its absence and so the user should take into account the affect of the absence of a line with a syntax error, when studying the error messages produced as a result of an assembly.

2.4 PRESENTATION

The discussion of the MICRO-11/60 assembler is organized to begin with the smallest unit, the program element, and to proceed through the structure and parts of the microprogram to some complete examples, as discussed in the following paragraphs.

The MICRO-11/60 assembler is a line-oriented processor, which accepts a sequence of input lines written in MICRO-11/60 source language and produces an object module that can be loaded into the 11/60 WCS. The elements of the source language are keywords, names, numbers, and separators. These program elements are described in Chapter 3.

The elements of the MICRO-11/60 source language are combined to form the processing-unit that MICRO-11/60 assembles to form an object module. The object module contains the microwords that, when loaded in the Writable Control Store, define the processing that is performed. The processing-unit is described in Chapter 4.

The MICRO-11/60 processing unit consists of two parts, namely Definitions and Actions. The first part contains the definitions that associate symbolic names with fields, values, field-value pairs and groups of field-value pairs, so that the actions part of the microprogram can be written as a sequence of logical functions. The second part contains the actions that are performed as a result of executing the microprogram. Definitions are described in Chapter 5 and actions in Chapter 6 of this part of the manual.

Two complete microprograms are given in Chapter 7 to illustrate the use of the source language in writing microprograms.

CHAPTER 3

PROGRAM ELEMENTS

A MICRO-11/60 microprogram is made up of a sequence of elements. These elements are keywords, names, values, and separators.

The microprogrammer writes his program in terms of these program elements. He uses spaces, tabs, and blank lines to arrange the program in a clear and readable format and he uses comments to describe the working of the program. When MICRO-11/60 interprets his program, it uses these spaces or comments only to separate the elements of the program. The assembler then combines the language elements to form language constructs and interprets these constructs to produce the desired microprogram.

As an example, consider the following program excerpt, taken from the microprogram in Section 7.1.

```
!
! START OF LOOP TO CHECK EVERY POINT AGAINST THRESHOLD
!
SRCHLP:
.BEGIN=0[6240:6241]
```

The first three lines are comments. The elements of the excerpt are as follows:

Element	Type
SRCHLP	name
:	separator
.BEGIN	keyword
=	separator
0	value
[	separator
6240	value
:	separator
6241	value
]	separator

This chapter explains the elements of which a MICRO-11/60 microprogram is built. Keywords, names, values, and separators are presented. Then, the program line is considered.

3.1 KEYWORDS

The keywords of MICRO-11/60 are given in the following list. The purpose of each keyword is summarized here and treated in more detail in later sections.

Keyword	Purpose
.TITLE	Provide microprogram identification.
.IDENT	Provide microprogram version number.
.BOUNDS	Delimit the address space to be used for the microprogram.
.RADIX	Change the default number base that is assumed for values written without an explicit radix.
.TOC	Make a table of contents entry.
.NLIST	Discontinue listing of assembled program on the listing file.
.LIST	Resume listing of the assembled program on the listing file.
.FIELD	Define a field name and its associated field values.
.MACRO	Define a macro.
.CODE	Indicate the beginning of the code portion of the microprogram.
.BEGIN	Identify and initiate a target assignment construct for conditional branching.
.CASE	Define a conditional branch target.
.ENDB	Indicate the end of a target assignment construct.
.END	Indicate the end of the microprogram.

All MICRO-11/60 keywords begin with the character period (.). A keyword must be given exactly as it is shown in the above list; otherwise, the assembler is unable to recognize the keyword and discards the line on which it appears as syntactically incorrect.

3.2 NAMES

A name in MICRO-11/60 can be composed of from 1 to 32 characters from the set of the characters given in Section 3.2.1, the first of which must be an alphabetic. Five types of names are distinguished by the MICRO-11/60 assembler:

Type	Description
Field-name	A name that identifies a set of bits within the microword.
Field-value-name	A name that represents a particular value for a given field.
Macro-name	A name that identifies a macro.
Formal	A name that is used within a macro-body to indicate a formal parameter.
Label	A name that is associated with a specific microinstruction address.

A name must only be unique within its type. For example, a field-name must be unique from all other field-names but can be the same as a field-value-name, macro-name, formal, or label.

The rules for forming a MICRO-11/60 name are given in the following syntax. Some valid MICRO-11/60 names are:

```
ALPHA

A123

B

THISISASYMBOLNAME

THIS_IS__A_SYMBOL_NAME

A
```

3.2.1 Syntax

name	alphabetic { name-char }$_{0}^{31}$
name-char	{ radix-50-char | _ | % | [|] }
alphabetic	{ A | B | ... | Z }
radix-50-char	{ alphabetic | digit | $ | . }
digit	{ 0 | 1 | ... | 9 }

3.2.2 Interpretation

A name begins with an alphabetic character and continues until a separator is encountered. Separators are described in Section 3.4.

3.3 VALUES

A MICRO-11/60 value consists of a sequence of one or more digits. The digits are interpreted according to the implicit radix, which is assumed initially to be 8 and which can be reset by a .RADIX keyword.

3.3.1 Syntax

value	{ digit }$_{1}^{n}$

3.3.2 Restrictions And Defaults

The maximum value for any number is 2**16 - 1. If a value greater than the maximum value is given, it is truncated; however, no error message is printed. Signed numbers are not accepted.

The implicit radix is assumed to be 8, but each occurence of a .RADIX keyword changes the implicit radix.

3.3.3 Interpretation

A value is interpreted according to the implicit radix and represented in the number of bits specified by the context. If the value cannot be represented in that number of bits, then the value is truncated to the required number of bits.

3.4 SEPARATORS AND DELIMITERS

Some characters have special meaning to MICRO-11/60 as separators or delimiters. The following list summarizes the separators and delimiters, and gives, for each, its special meaning.

Separator		Meaning
.	(period)	Used to indicate a keyword.
@	(at)	Used to indicate a formal parameter in the macro-body of a macro definition.
=	(equals)	Used to initiate a constraint-string.
::= or :=		Used to associate a meaning with a name for field definitions, macro definitions, and field settings.
/	(slash)	Used to separate a field-name from its value in a microinstruction.
,	(prime)	Used to indicate concatenation.
"	(quote)	Used to begin and end a quoted string.
:	(colon)	Used to terminate a label and an address.
*	(asterisk)	Used within a constraint string.
,	(comma)	Used to separate field-settings in a microinstruction or macro-body.
;	(semicolon)	Used to terminate a microinstruction.
!	(exclamation point)	Used to begin a comment.
(CR)	(carriage return)	Used to end a program line.

3.5 THE PROGRAM LINE

MICRO-11/60 is a line-oriented processor. It assumes that each program line contains a complete and coherent piece of microprogram.

A program line can consist of from 1 to 120 characters. If the number of characters on a line is greater than 120 but less than 124, then any error messages associated with the line are lost. If the number of characters on a line exceeds 123, then the rest of the listing is lost.

Some of the MICRO-11/60 constructs must be expressed on a single line, namely: the text following the keywords .TITLE, .IDENT, .BOUNDS, .ENTRY, AND .ENDB. Other constructs are intrinsically multi-lined and each line expresses a particular part of the construct.

The syntax for each construct expresses its representation on program lines. For example, consider the syntax of the field-definition.

field-definition	.FIELD field-name { := / ::= } field-spec { field-value-name { := / ::= } value }n 0

This syntax indicates that a field-definition begins with a line that contains the keyword .FIELD followed by the field-name, followed by either the characters ":::=" or the characters ":=", followed by the field-spec. Following that line, a sequence of lines that define field-value-names can be given.

3.5.1 Comments

Comments can be given on separate lines or at the end of any line. A comment is delimited by the character "!" on the left and the carriage return that ends the line on the right.

The general form of a comment is:

```
! comment-text
```

Comment text is reproduced by the assembler on the output listing, but the assembler does not interpret the comment-text in any way. Comment text can, therefore, appear at any point in the program and can consist of any set of characters.

Some examples of comments are given in the following excerpt:

```
!  ENTRY POINT FOR MATRIX ADDITION
MATADD:
  P1,     CLK-BA,PC-A,                    !INITIATE MEM(PC) READ:
  P2-T,   A-PLUS-B,CSPB(TWO),             !INCREMENT PC.
  P3,     WR(AB,L,A),DATI,
  NEXT,   J,MAT1
```

The use of comments increases the readability of a microprogram. Chapter 7 of this manual contains two microprograms that are well documented by the use of comments.

3.5.2 Spacing

Spaces can be inserted between any of the units of the microprogram. The characters BLANK and TAB can be used to insert space within the program line and the character CARRIAGE RETURN can be used to irsert space between program lines.

3.5.2.1 Inter-Line Spacing - As an example of the use of blanks and tabs for inter-line spacing, consider the following field definition. First, without spacing, it looks like:

```
.FIELD SWITCH ::= <22>
OFF::=0
ON::=1
```

Then, after the addition of some spaces, it looks like:

```
.FIELD  SWITCH  ::=  <22>
   OFF  ::=  0
   ON   ::=  1
```

The field-value-names OFF and ON are started at the first tab to indicate their logical dependence on the line beginning with '.FIELD'. Blanks are used to line up the characters "::=" and the values for ease of reading.

3.5.2.2 Intra-Line Spacing - Blank lines are used to separate logical sections of the microprogram. If, for example, the microprogram contains several field-definitions, then the readability of the program is improved by separating each field-definition from the one that follows by a blank line. For example:

```
.FIELD  SWITCH  ::=  <22>
   OFF  ::=  0
   ON   ::=  1

.FIELD  AFIELD  ::=  <30:20>
   ALPHA  ::=  0
   BETA   ::=  1
   GAMMA  ::=  4
```

CHAPTER 4

PROGRAM STRUCTURE

This section describes the structure of an 11/60 microprogram. First, the basic processing unit is described. Then, the identification part of the microprogram is considered. Finally, the keyword lines that can appear at any point within the microprogram are given.

4.1 THE PROCESSING UNIT

The processing-unit of the MICRO-11/60 assembler is a microprogram. A microprogram consists of the two logical parts: definitions and actions. In the definition part, the 11/60 predefinitions, any program identification, user field definitions, and user macro definitions are specified. In the action part, the dispatch-file and the user microinstructions, written in terms of the names defined in the definition part, determine the processing that is performed when the microprogram is executed.

As an example of a complete microprogram, consider the following:

```
.TITLE   REGEX
.IDENT   /REGEX1/

.CASE 0 OF DISPCH
EXCHANGE:
         P2-T,  SR_A,R3-A,                 !SAVE R3
         NEXT,  J/EXCH2;

EXCH2:
         P2-T,  D_A,R2-A,                  !MOVE R2 TO R3
         P3,    WR(AB,L,B),R3-B,
         NEXT,  J/EXCH3;

EXCH3:
         P2-T,  D_SR,                      !MOVE SAVED R3 TO R2
         P3,    WR(AB,L,B),R2-B,
         NEXT,  BUT(SUBRB),PAGE(0),
                J/BRA05;

.END
```

The above microprogram exchanges registers R2 and R3. The program uses only predefined field and macro names and, therefore, a user-definition part is not present. The action part of the program contains the identification lines, which provide title, version, and bounds information. The microinstructions determine the processing in the microprogram. This microprogram contains three microinstructions. The first microinstruction, labelled EXCHANGE, saves the contents of R3 in the shift register (SR). The second microinstruction, labelled EXCH2, moves the contents of R2 to R3 in both A and B scratchpads. The last microinstruction, EXCH3, moves the saved contents of R3 to R2 in both scratchpads and returns to the base machine so that the next PDP-11 instruction can be processed.

4.1.1 Syntax

processing-unit	predefinitions { user-definitions } 1 / 0 dispatch-part action-part

4.1.2 Interpretation

The processing-unit consists of a sequence of lines that identify the microprogram and the constructs to be used in it, followed by a sequence of microinstructions that make up the program. The assembler uses the definitional part of the program to label and interpret the action part of the program. The assembler produces an object module that can be loaded into the Writable Control Store and executed.

4.2 IDENTIFICATION PART

The identification-part of a microprogram can appear in either the user-definition or action-part. It contains the .TITLE and .IDENT keywords. The .TITLE keyword associates a name with the program and the .IDENT keyword designates the current version number of the program.

As an example of an identification-part, consider the following:

```
.TITLE MATRIX PACKAGE
.IDENT /MPV3A/
```

The .TITLE keyword associates the name MATRIX with the microprogram and the .IDENT keyword indicates that the current version is MPV3A.

4.2.1 Syntax

identification-part	.TITLE title-string .IDENT / ident-string /
title-string ident-string	{ radix-50-char } 64 1

4.2.2 Interpretation

The identification-part of a microprogram is interpreted as follows:

The .TITLE line is used to associate an identifying title with the microprogram. The first six characters of the title-string, or, if a blank occurs in the first six characters, the characters preceding the blank are used as the title in the page heading on each page of the output listing and in the object module produced as a result of the assembly.

The .IDENT line is used to associate a version number with the microprogram. The first six characters of the ident-string or the characters preceding the first blank are used as the version number in the object module produced as a result of the assembly.

Although the .TITLE and .IDENT keywords are normally given only once, as the first two lines of the microprogram, they can be given at any point in the definition part or action part of the microprogram and can be repeated any number of times. In such a case, the last title-string or ident-string encountered is used.

4.2.3 Defaults

If the .TITLE keyword is not given, then a title consisting of 6 blanks is assumed.

If the .IDENT keyword is not given, then an ident-string consisting of 6 blanks is assumed.

4.2.4 Guidelines

The title-string and ident-string are truncated after six characters or at the first blank or tab, whichever comes first. These strings therefore, should be chosen so that the part left after truncation is both unique and meaningful. The ident-string should be changed each time the microprogram is updated, so that different versions of the program are easily distinguishable.

4.3 TOC-LINES

A toc-line is used to identify a logical segment of the microprogram, just as a heading is used to identify a logical segment of a document. The assembler collects the toc-lines and prints them at the beginning of the output listing and, in this way, produces a table of contents for the listing.

The following program excerpt illustrates the use of toc-lines:

```
.TITLE MATRIX PACKAGE
.IDENT /MPLL27/
.ENTRY MATPAK ::= 1
.TOC  *MATRIX PACKAGE
.TOC  *  INITIALIZATION
...
.TOC  *  OPERATION DISPATCH
...
.TOC  *    MATRIX MULTIPLY
...
.TOC  *    MATRIX ADD
...
.TOC  *    MATRIX INVERT
...
.TOC  *  FINALIZATION
...
```

In the output listing, each line is numbered starting from 0001. Thus, the table of contents provides a quick reference to the appropriate line in the microprogram. The output listing for the above excerpt has the following form:

```
TABLE OF CONTENTS

   4  -- *MATRIX PACKAGE
   5  -- *  INITIALIZATION
  22  -- *  OPERATION DISPATCH
  30  -- *    MATRIX MULTIPLY
  79  -- *    MATRIX ADD
 125  -- *    MATRIX INVERT
 220  -- *  FINALIZATION



   1  .TITLE MATRIX PACKAGE
   2  .IDENT /MPLL27/
   3  .ENTRY MATPACK ::= 1
   4  .TOC  *MATRIX PACKAGE
   5  .TOC  *  INITIALIZATION
...
  22  .TOC  *  OPERATION DISPATCH
...
  30  .TOC  *    MATRIX MULTIPLY
...
  79  .TOC  *    MATRIX ADD
...
 125  .TOC  *    MATRIX INVERT
...
 220  .TOC  *  FINALIZATION
```

Observe that an interesting table of contents is constructed if the toc-lines are indented to indicate subordination.

4.3.1 Syntax

toc-line	.TOC toc-string
toc-string	{ name-char } 64 / 1

4.3.2 Interpretation

For each toc-line, the assembler creates a line of the form:

line-number -- toc-string

The assembler prints the created table of contents line at the beginning of the output listing. The toc-string is printed exactly as it appears in the input, including any leading spaces or tabs.

4.4 RADIX LINES

A radix-line is used to change the implicit radix. The implicit radix is the radix that is assumed for a value that appears in the program. The value '22', for example, is interpreted according to the implicit radix and, therefore, can be interpreted as an octal 22 at one point in the program and as a decimal 22 at another point.

As an example of the use of radix-lines, consider the following microprogram excerpt:

```
.TITLE  ABC
.IDENT  /ABCV1/
.FIELD  FIELD1 ::= <10:5>
        VAL1 ::= 20          ! OCTAL 20
.RADIX 10
.FIELD2 ::= <22:10>
        Q1 ::=  12           ! DECIMAL 12
.CODE
0022:
E1:
        FIELD1/VAL1,         ! OCTAL 20
.END
```

The comments in the above microprogram indicate the radix according to which the value on the same line is interpreted. The implicit radix is assumed initially to be 8. Thus, the value of VAL1 is assumed to be an octal 20. When FIELD1 is set to VAL, the implicit radix is 10, but the value of VAL1 was established to be an octal 20 in the definition part of the program and, therefore, an octal 20 is assigned to that field.

4.4.1 Syntax

radix-line	.RADIX radix
radix	{ 2 | 8 | 10 }

4.4.2 Interpretation

A radix-line changes the implicit radix to the radix specified in the line. All values are interpreted according to the implicit radix currently in effect. An implicit radix is in effect until another radix-line is encountered.

4.4.3 Defaults

The implicit radix is assumed initially to be 8.

4.4.4 Discussion

Only those syntactic units that are identified as values in the syntax are interpreted according to the implicit radix. Other numbers, such as addresses and bit specifiers, are not interpreted according to the implicit radix. An address, for example, is always interpreted as an octal-value and is so identified in the syntax. Similarly, a bit specifier is always interpreted as a decimal number.

4.5 LIST KEYWORDS

The list keywords are used in pairs to suppress part of the output listing produced as a result of assembling a microprogram. The .NLIST keyword directs the assembler to suppress listing until a .LIST keyword is encountered. The .LIST keyword directs the assembler to resume listing.

Both files supplied by DIGITAL, the predefinitions file and the dispatch file, contain list keywords that prevent the listing of contents of these files as part of each microprogram assembly. The predefinitions file output, obtained separately without the use of list keywords, is reproduced in Appendix B; the dispatch file is reproduced in Appendix C.

4.5.1 Syntax

list-keywords	{ .NLIST .LIST }

4.5.2 Interpretation

As part of the assembly process, MICRO-11/60 creates a listing file, as described in Section 14.4. If the assembler encounters a .NLIST keyword, it stops writing the listing file. If the assembler encounters a .LIST keyword, it resumes writing the listing file.

The .NLIST and .LIST keywords scope the material that is to be suppressed. The list-keywords essentially change the listing mode of the assembler. If the assembler encounters a .LIST when it is in listing-mode, it effectively ignores that keyword. Similarly, if the assembler encounters a .NLIST when it is in suppress-listing-mode, it ignores the keyword.

4.5.3 Defaults

The default mode is listing-mode.

CHAPTER 5

DEFINITIONS

The definitions part of a MICRO-11/60 microprogram identifies the microprogram and specifies the meanings of all names that are used. The definitions part consists of two units: the predefinitions supplied by DIGITAL and any user-definitions supplied by the programmer. The programmer can define names for fields in the microword and can define macros that set one or more of these fields. The choice of meaningful names and the specification of macros that perform logical functions enhance the readability of the microprogram.

user-definitions	{ field-definition, macro-definition } n, 0

The user-definition part of the microprogram is optional. The MICRO-11/60 assembler obtains from the predefinitions file knowledge of the name of each field, its position within the microword, its length, default setting, truncation mode, and associated field-value-names. Similarly, a set of macros to perform the logical functions associated with microprogramming the 11/60 is also predefined. Most microprograms are written exclusively in terms of these predefined fields and macros and, therefore, do not contain any field or macro definitions.

Occasionally, however, the programmer wants to define a new field or a new set of macros for a special application. In such a case, the definition part of the microprogram contains the field or macro definition. In processing the definition part of the microprogram, the assembler assimilates the information in these definitions and adds that information to the body of predefined information for the duration of the assembly.

This section describes the definition-part of a microprogram. First, field definitions are described. Then, macro definitions are given. Finally, predefinitions are discussed.

5.1 FIELD DEFINITIONS

A field-definition is used to assign a mnemonic name to a set of bits within the microword and, further, to associate with that field a set of names that suggest the meanings of the values that the field can have. For example, consider the following definition of the Bus ENable field, taken from the 11/60 predefinitions:

```
.FIELD BEN ::= <43:42>
     BSPLO ::= 0
     BSPHI ::= 1
     CSP ::= 2
     BASCON ::= 3
```

The field BEN is predefined to occupy bits 43 and 42 of the microword and to have associated with it four field-value-names, namely: BSPLO, BSPHI, CSP, BASCON. The value of the BEN field controls the source that is enabled onto the BUS BIN, as described in the "11/60 Microprogramming Specification". The definition of field-value-names allows the programmer to write the following field-setting in a microinstruction:

```
BEN/BASCON
```

The above field-setting assigns the value 3 to bits 43 through 42 and, further, describes the intent of that assignment, namely: to enable the BASe CONstant source onto BUS BIN.

As another example, consider the following definition of the EMIT field, taken from the 11/60 predefinitions:

```
.FIELD EMIT ::= <47:44>'<41:30>
```

The field EMIT is predefined to occupy bits 47 through 44 and bits 41 through 30 of the microword.

5.1.1 Syntax

field-definition	.FIELD field-name { := / ::= } field-spec { field-value-definition }$_{0}^{n}$
field-spec	bit-spec { , default }$_{0}^{1}$
bit-spec	{ bit-range }$_{1}^{n}$ '
bit-range	< left-bit { : right-bit }$_{0}^{1}$ >
default	value
left-bit right-bit	{ decimal-digit }$_{1}^{2}$
field-value-definition	field-value-name { := / ::= } value
field-value-name field-name	name

5.1.2 Interpretation

A field-definition is interpreted as follows:

The field-spec is evaluated and the designated bits are associated with the field-name. If a default value is given, then that default value becomes part of the default initialization pattern.

The field-value-definitions are then processed. Each field-value-name and its assigned value, truncated, if necessary, according to the truncation mode, are associated with the field-name. Once specified in this way, a field-value-name can be used as part of a field-value-name/field-value pair to provide a value for a field.

5.1.3 Restrictions

The maximum field size that can be specified is 16 bits. Therefore, the bit spec must not specify a concatenation of bits such that the length of the field consisting of those bits exceeds 16.

The maximum number of bit-ranges that can be given in a bit-spec is 16. The limit on bit-ranges derives from the field size limit.

Left-bit and right-bit in a bit-range must satisfy the following inequality:

```
47  >=  left-bit  >=  right-bit  >=  0
```

However, within a sequence of bit-ranges, the relationship among the bit-ranges is unspecified. For example, the bit-spec "<12:10>'<30:26>'<6:5>" is perfectly valid.

5.1.4 Defaults

If right-bit is omitted in a bit-spec, then the left-bit and right-bit are assumed to be the same and a field size of 1 is assumed.

5.1.5 Semantics

The following sections describe the detailed semantics of the field-definition. First, the field-spec is discussed with special attention being given to the topics of non-contiguous and overlapping bit fields. Then, the default initialization pattern is described.

5.1.5.1 Field-Specs - The field-spec gives the information necessary to define the field. It specifies the bits occupied, and, optionally, any default value. A field can be specified to be either a contiguous-bit field or a non-contiguous-bit field.

5.1.5.2 Contiguous-Bit Fields - A contiguous-bit field can be expressed by a single bit spec, as follows:

```
.FIELD field-name ::= <left-bit:right-bit>
```

The field is defined to occupy the contiguous set of bits starting with left-bit and proceeding through right-bit. The length of a contiguous bit field is calculated as follows:

```
field-length = left-bit - right-bit + 1
```

An example of a contiguous-bit field is given by the following definition of the ALU field from the 11/60 predefinitions:

```
.FIELD  ALU ::= <47:44>
```

The field ALU is defined to occupy bits 47 through 44 of the microword. The length of the ALU field is 4 bits.

5.1.5.3 Non-Contiguous-Bit Fields - A non-contiguous-bit field can be expressed by concatenating two or more bit-specs, as follows:

```
.FIELD  field-name  ::=  <l1:r1>'<l2:r2>'...'<ln:rn>
```

Such a field occupies left-bit1 (l1) through right-bit1 (r1), followed by left-bit2 (l2) through right-bit2 (r2), and so on. The length of a non-contiguous-bit field is calculated by adding the lengths of the contiguous components.

```
field-length  =  (l1-r1+1)+(l2-r2+1)+...+(ln-rn+1)
```

An example of a non-contiguous-bit field is given by the following definition of the EMIT field from the 11/60 predefinitions:

```
.FIELD  EMIT  ::= <47:44>'<41:30>
```

The field EMIT is defined to occupy bits 47 through 44 followed by bits 41 through 30 of the microword. The length of the EMIT field is 16 bits. The bits occupied by the EMIT field are indicated in the following diagram by X.

```
4      4         3         2         1
7654321098765432109876543210987654321098765432 10
```

If the field-setting 'EMIT/65432' is given in a microinstruction, then the bits of the EMIT field are set as shown in the following diagram:

```
4      4         3         2         1
7654321098765432109876543210987654321098765432 10
```

```
0110  101100011010
```

5.1.5.4 Overlapping Fields - The MICRO-11/60 assembler allows the definition of overlapping fields and the redefinition of fields and subfields. This flexibility permits, for example, the different structures of the 11/60 microword to be expressed.

For example, the definitions of the ALU and EMIT fields in the 11/60 definition, as seen in the previous section, define overlapping fields. The ALU field is indicated by A and the EMIT field by E in the following diagram:

```
4      4         3         2         1
7654321098765432109876543210987654321098765432 10
```

```
AAAA
EEEE  EEEEEEEEEEEE
```

When a microword is executed, bit steering within the word determines the meaning of fields within the word. The assembler, however, does not attempt to determine if a microword is either complete or consistent. However, if the programmer attempts to set the same bit more than once in a microword, the assembler reports an error. Consider, for example, a microinstruction that, by mistake, sets both the ALU and EMIT fields, as follows:

```
EMIT/65432,ALU/17
```

The EMIT field-setting is processed first and, as a result, the bits are set as shown in the previous section. Then the ALU field setting is processed and bits 47 through 44. An error is reported. However, if the inconsistent field settings do not involve overlapping fields, the assembler is unaware of any problem.

5.1.5.5 Default Initialization Pattern - The pattern that is used to initialize each word in the microprogram before the explicit field-settings are processed is called the default initialization pattern. This pattern is constructed from the defaults specified for field-definitions.

The construction of the default initialization pattern starts with a word that consists of 48 zeros. When field-definitions are processed, any default values are set in the default initialization pattern.

Since the MICRO-11/60 assembler reads the 11/60 predefinitions first, the default initialization pattern is the pattern that exists after processing the defaults in the predefinitions. The only predefined field that has a non-zero default is the UBF field, which is defined as follows:

```
.FIELD UBF ::= <13,9>,30
```

Therefore, the default initialization pattern after considering the predefinition is:

```
4      4         3         2         1
765432109876543210987654321098765432109876543210
┌──────────────────────────────────────────────┐
│000000000000000000000000000000000011000000000000│
└──────────────────────────────────────────────┘
```

If the programmer adds a field-definition with default value, then this default is logically ORed with the default pattern and the result of that operation becomes the new default pattern. Suppose, for example, the programmer adds the following:

```
.FIELD  ALPHA ::= <2:0>,2
.FIELD BETA   ::= <47:40>,25
```

Then the default initialization pattern becomes:

```
4      4         3         2         1
765432109876543210987654321098765432109876543210
┌──────────────────────────────────────────────┐
│000101010000000000000000000000000011000000000010│
└──────────────────────────────────────────────┘
```

The default pattern, therefore, is the word formed by the logical OR of all defaults given. If two fields overlap, only one field should be assigned a default value.

If the default value requires more bits for its representation than are present in the associated field, the default value is truncated and no error message is reported.

5.1.5.6 Oversize Field Values - When a value that requires more bits than are present in a field is assigned to that field, then the high order bits are truncated. For example, suppose the following field-setting is given by mistake:

```
ALU/32
```

As previously noted, the ALU field is 4 bits long. However, the value 32, interpreted according to the implicit radix 8, requires 5 bits for its representation. The assembler truncates the high order 1 and assigns the value 12 to the ALU Field and no error is reported.

5.2 MACRO DEFINITIONS

A macro-definition is used to obtain a convenient and readable notation for a commonly performed operation. For example, consider the following macro definition, taken from the 11/60 predefinitions:

```
.MACRO  DIVIDE ::= ALU/DIVIDE
```

This definition allows the programmer to write the string DIVIDE within a microinstruction instead of the more lengthy string ALU/DIVIDE.

Often, macro definitions are written that combine the setting of several related fields. For example, consider the following:

```
.MACRO  D-RIGHT-14  ::=  AEN/CMUX,AMUX/RIGHT-8,BMUX/RIGHT-4,
                         ASEL/RIGHT-2
```

The programmer can write D-RIGHT-14 within a microinstruction to shift D to the right by 14. The assembler replaces the macro-call 'D-RIGHT-14' by the macro body and the fields to accomplish that shift are set appropriately.

5.2.1 Syntax

macro-definition	.MACRO macro-name { ({ formal }n,1) }1,0 { := / ::= } macro-body
macro-body	{ macro-body-part }n,1
macro-body-part	{ field-name / value-spec / macro-call }
value-spec	{ field-value-name | value | @ formal }
macro-call	macro-name { ({ actual }n,1) }1,0
actual	{ name | @ formal }
macro-name, formal, field-name	name

5.2.2 Interpretation

A macro-definition is interpreted as follows:

> The macro-body is associated with the macro-name so that when a macro-call is encountered, the macro-body, with any formals replaced by actuals, replaces the macro-call.

A macro-call is interpreted as follows:

> The macro-body associated with the macro-name is copied, and the formal parameters in the macro-body are replaced by the actual-parameters in the macro-call, the i'th formal being replaced by the i'th actual. Any excess actual parameters are discarded.

5.2.3 Restrictions

Macros must not be defined to be recursive. That is, the definition of a macro must not contain a call on itself or a call on another macro that ultimately results in a call on itself.

The substitution of an actual for a formal must result in a correct syntactic unit. For example, the actual that replaces a formal in a field-identifier must be a field-name.

5.2.4 Defaults

If an actual is not given for each formal, then sufficient actual parameters with the value 0 to provide for the defined formals are assumed to follow the explicitly given actuals.

5.2.5 Semantics

The detailed semantics of macro-definition and use are considered in the following sections. First, macro expansion is described for the simplest case, in which the macro-definition has no parameters. Then, parameters are considered. Finally, the nesting of macros is described.

5.2.5.1 Macro Expansion - The replacement of a macro-call by the macro-body associated with that macro is called macro expansion. When the assembler encounters a macro-call within a microinstruction, it replaces the macro-call by the macro-body and then processes the macro-body. For example, consider the following macro-definition:

```
.MACRO ALPHA ::= AFLD/10,BFLD/20
```

Suppose that the macro is called within a microinstruction as follows:

```
L1:
     CFLD/10,ALPHA,DFLD/5
```

The assembler first processes the field-setting CFLD/10. Then, when it encounters the macro-call ALPHA, it replaces the call by the macro-body AFLD/10,BFLD/20, processes the field-setting AFLD/10, then the field-setting BFLD/20, and finally the fiel -setting DFLD/5.

The text of the macro-body is not interpreted until it is expanded and therefore, it is understood in the context of the point at which it is expanded. The implicit radix is an example of the context of interpretation. Suppose the macro ALPHA is defined when the implicit radix is 8 and called first when the radix is 10 and again when the radix is 8 as follows:

```
.RADIX 8
.MACRO ALPHA ::= AFLD/10,BFLD/20
...
.CODE
.RADIX 10
L2:
     ALPHA;
...
.RADIX 8
L3:
     ALPHA;
```

Because the macro-body is not interpreted when it is defined, the implicit radix at that point is irrelevant. For microinstruction L2, ALPHA is expanded and interpreted when the implicit radix is 10 and therefore AFLD is set to a decimal 10 and BFLD to a decimal 20. For microinstruction L3, ALPHA is expanded and interpreted when the implicit radix is 8 and AFLD and BFLD are set to octal 10 and octal 20 respectively.

5.2.5.2 Parameters - The simplest case of a macro-definition is the case in which no parameters are defined. In that case, the macro-call simply consists of the macro-name. However, when a macro is defined with parameters, it is possible to specify a more general and powerful substitution.

The formal parameters of a macro definition are identified within parentheses following the macro-name and then indicated within the text of the macro-body by the character '@'. As an example of a macro-definition with parameters, consider the following:

```
.MACRO BETA(X,Y) ::= AFLD/@X,BFLD/@Y
```

The formal parameters of the macro definition are X and Y. The position of X and Y within the parentheses in the macro-definition is important. When a call is made on the macro, the actual parameters in the call are associated with the formal parameters positionally. That is, the first actual is associated with the first formal, the second actual with the second formal, and so on. Consider the following macro-call:

```
BETA(C,D)
```

The actual parameters of the call are C and D. The expansion of the macro-call is:

```
AFLD/C,BFLD/D
```

The first actual C is associated with the first formal X and the second actual D with the second formal Y.

If more actuals than formals are given, then the extra actuals are discarded. If fewer actuals than formals are given, then the missing actuals are assumed to be zero. If a null actual is specified, then that actual is assumed to be zero. Some examples follow to illustrate these cases:

Macro-Call	Macro Expansion
BETA(C,D,E)	AFLD/C,BFLD/D
BETA(D,E)	AFLD/D,BFLD/E
BETA(F)	AFLD/F,BFLD/0
BETA(,G)	AFLD/0,BFLD/G
BETA()	AFLD/0,BFLD/0
BETA	AFLD/0,BFLD/0

A formal parameter is only defined within the macro-body associated with the macro-definition. Further, within the macro-body, the formal must be preceded by the character '@'. Consider the following microprogram excerpt:

```
.FIELD F1 ::= <24:22>
     A ::= 1
     B ::= 2
.MACRO GAMMA(A,B,C) ::= F1/A,F2/@A,F3/@B,F4/@C
.CODE
L1:
     GAMMA(G,H,I);
...
.END
```

The expansion of the macro GAMMA in microinstruction L1 produces the following string:

```
F1/A,F2/G,F3/H,F4/I
```

The symbol A is interpreted first as a field-value-name. Then, when it is preceded by an '@', it is interpreted as the first formal parameter.

5.2.5.3 Nested Macros - The macro-body of a macro definition can contain calls on other macros. As an example, consider the following macro-definition from the 11/60 predefinitions:

```
.MACRO ASPLO[00]_D   ::= ASP(R00),WR(A,L,A)
.MACRO ASP(XX)        ::= ASEL/@XX,RIF/@XX
.MACRO WR(AB,HL,ADDR) ::= MOD/CLKSP,WRSP/@AB,HILO/@HL,WRSEL/@ADDR
```

The macro ASPLO[00]_D is equivalent to the following string:

```
ASEL/R00,RIF/R00,MOD/CLKSP/WRSP/A,HILO/L,WRSEL/A
```

As another example, consider the following set of macro definitions:

```
.MACRO ALPHA(A)     ::= FLD/@A,BETA(@A,B)
.MACRO BETA(X,Y)    ::= XYZ/@X,GAMMA(Q,R,S)
.MACRO GAMMA(X,Y,Z)::= AFLD/@X,BFLD/@Y,CFLD/@Z
```

The macro-call ALPHA(AC) is expanded as follows:

```
FLD/AC,BETA(AC,B)

       XYZ/AC,GAMMA(Q,R,S)

              AFLD/Q,BFLD/R,CFLD/S
```

Thus, the final string is:

```
FLD/AC,XYZ/AC,AFLD/Q,BFLD/R,CFLD/S
```

5.3 PREDEFINITIONS

The 11/60 predefinitions give a set of field and macro definitions that are sufficient for most 11/60 microprograms. The complete set of predefinitions, in MICRO-11/60 source, is given in Appendix B of this manual. The predefinitions-file is part of the input to a normal MICRO-11/60 assembly, as described in Chapter 14.

5.3.1 Field Predefinitions

The field predefinitions specify all the field names for the 11/60 microword. In addition, for each field-name, a set of field-value-names is defined. These names are selected to provide convenient mnemonics for the value of the field. For example, consider the following predefinition for the ALU field:

```
.FIELD  ALU ::=  <47:44>
        NOT-A  ::= 00
        A-PLUS-B-PLUS-PS C   ::=  01
        NOT-A-AND-B  ::= 02
        ZERO  ::=  03
        A-PLUS-B-PLUS-D C   ::=  04
        A-PLUS-NOT-B-PLUS-D C   ::= 05
        A-XOR-B ::= 06
        A-AND-NOT-B  ::= 07
        DIVIDE  ::=  10
        A-PLUS-B  ::= 11
        B  ::= 12
        A-AND-B ::= 13
        A-PLUS-B-PLUS-1  ::= 14
        A-MINUS-B ::= 15
        A-IOR-B  ::=  16
        A  ::=  17
```

Note that the field-value-names correspond closely to the verbal definition given for the field value in Table 2-1 of the "PDP-11/60 Microprogramming Specification".

When several choices for a field-value-name are possible, the predefinitions provide all the names as a convenience for the microprogrammer, who can then use the name that seems most logical to him. As an example, consider the following predefinition:

```
.FIELD WRSP ::= <16:15>
     NOP::=0              ! NO ASP/BSP REWRITE
     WR-A::=1             ! WRITE ASP ONLY, ON P3 120-150 NS.
       A::=1
       ASP::=1
     WR-B::=2             ! WRITE BSP ONLY, ON P3 120-150 NS.
       B::=2
       BSP::=2
     WR-A-AND-B::=3       ! WRITE BOTH ON P3
       AB::=3
       BA::=3
       ABSP::=3
       BASP::=3
       BOTH::=3
```

To set the WRSP field so that both scratchpads are written, the microprogrammer can write any of the following field-settings:

```
WRSP/WR-A-AND-B     WRSP/AB         WRSP/BA
WRSP/ABSP           WRSP/BASP       WRSP/BOTH
```

5.3.2 Macro Predefinitions

The macro predefinitions provide a language for microprogramming the 11/60. A macro is predefined for each logical function that the microprogrammer wants to perform.

Macros are defined to supply a convenient name for a field-setting. For example:

```
.MACRO CLK-SR ::=  CLKSR/YES
```

Macros are predefined to set the several associated fields that must be specified to perform a logical function. For example, to write a scratch pad, the MOD field must be 0; the WRSP must be set to 1 to write A, 2 to write B, or 3 to write both; the WRSEL field must specify the address to be used; and HILO must specify the section of the scratchpad. To do this, the following macro is predefined:

```
.MACRO WR(AB,HL,ADDR) ::=   MOD/CLKSP,
                            WRSP/@AB,
                            HILO,@HL,
                            WRSEL/@ADDR
```

A call on this macro sets the four fields necessary. For example:

```
WR(AB,L,A)
```

This macro-call sets the MOD field to 0, the WRSP field to AB to write both scratchpads, the HILO field to L to indicate the low section, and the WRSEL field to specify the A address.

CHAPTER 6

ACTIONS

The microcode-part contains the actions of the microprogram. The microcode-part of the program consists of two units: the dispatch-file supplied by DIGITAL and the action-part supplied by the programmer. These actions determine the processing that is performed when the microprogram is executed.

The action-part begins with the dispatch-file, which contains the .CODE keyword, continues with the user-actions, and ends with the keyword .END. The dispatch-file is given in Appendix C; it provides a dispatch table to be used as an entry point mechanism. The user-actions are given in the following syntax.

user-actions	{ action-item } n, 1 .END
action-item	{ microinstruction branch-definition case-microinstruction end-definition }

The following sections describe the microinstruction, the target-assignment construct and the entry point mechanism. The branch-definition, case-microinstruction, and end-definition are parts of the target-assignment construct. The entry point mechanism is an application of the target assignment construct.

6.1 MICROINSTRUCTIONS

The microinstruction is the basic unit of the microprogram. It contains the information necessary to set the bits in the microword. As an example of a microinstruction, consider the following:

```
ALPHA:
     P1,    CLK-BA,PC-A,
     P2-T,  A-PLUS-B,CLK-D,CSPB(TWO),
     P3,    WR(AB,L,A),DATI,
     NEXT,  J/BETA;
```

As described in the previous section on "Definitions", this microinstruction is made up of predefined macros. When these macros are expanded, the instruction has the following form:

```
ALPHA:
          CLKBA/YES,
          AEN/ASPLO,ASEL/R07,RIF/R07,
          ALU/A-PLUS-B,CLKD/YES,BEN/BASCON,BSEL/TWO
          MOD/CLKSP,WRSP/AB,HILO/L,WRSEL/A,BEGIN/YES,
                SELECT/BUS,BUSCODE/DATI,
          J/BETA;
```

The expanded microinstruction consists of predefined field-name/ field-value-name pairs. The field-name defines the position within the microword and the field-value-name gives the value to be inserted in that field. The assembler initializes the microword to the default value and then fills in the fields as indicated to form the following microword:

```
4      4         3         2         1
765432109876543210987654321098765432109876543210
┌──────────────────────────────────────────────┐
│100111111001111    1 1 0110  00110     001001110│
└──────────────────────────────────────────────┘
 ALU  BSEL    CLKD          MOD    J
     BEN       CLKBA        WRSP
         AEN      BEGIN     WRSEL
           ASEL       SELECT HILO
             RIF        BUSCODE
```

6.1.1 Syntax

microinstruction	{ address : }1,0 { label : }1,0 { instruction-part }n,0 ;
instruction-part	{ field-name / field-value macro-call }
field-value	{ field-value-name | value }
macro-call	macro-name { ({ actual }n,1) }1,0
field-name field-value-name macro-name actual label	name
address	octal-value

6.1.2 Interpretation

A microinstruction is interpreted as follows:

If an address is specified and the address is both valid and available, then it is assigned to the microinstruction. If an address is not specified, then the assembler selects the first available address within the specified bounds on the current page from the available address pool and assigns that address to the microinstruction. The address assigned either explicitly or by the assembler is removed from the available address pool.

If a label is specified, the label is associated with the assigned address.

The microword is initialized to the default pattern, calculated by the logical ORing together of all the field defaults, as described in Section 5.1.5.5.

The lines of the microinstruction are processed from left to right. In this processing, any macros encountered are expanded and field-name/field-value pairs evaluated. The specified values are set into the specified fields of the microword. If a microinstruction line ends with the ',' delimiter, then another microinstruction line is processed as part of the current microinstruction. However, if the microinstruction line ends with either the ';' delimiter or a blank, then the current microinstruction is assumed to be complete and the next microinstruction line is assumed to start a new microinstruction.

6.1.3 Restrictions

The address specified for a microinstruction must lie within the range 6200 through 7777.

The field-setting that gives the jump address for the next microinstruction 'J/N' must be the last field-setting in the microinstruction.

A line within a microinstruction can contain one or more instruction-parts. The comma separator indicates that more instruction-parts for the microinstruction follow. The semicolon separator indicates that the line terminates the microinstruction.

A field-value-name specified in an instruction-part must be one of the field-value-names defined for the field-name in a field-setting of a .FIELD definition.

The maximum field size that can be specified is 16 bits.

6.1.4 Defaults

If a line does not end with a separator, then the separator semicolon, which indicates the end of the microinstruction, is assumed.

If a macro-call contains fewer actual parameters than the number of formal parameters specified in the macro definition of the macro-name, then as many additional actual parameters as necessary, with the value 0, are assumed to follow the given actuals in the macro-call. If a macro-call contains more actual parameters than the number of formals specified, the extra actuals are discarded.

6.2 TARGET ASSIGNMENT

The MICRO-11/60 assembler provides a construct for specifying the targets of a conditional branch, namely: the target assignment construct. Conditional branching is accomplished in 11/60 microprograms by combining the output of the Branch Micro Test Multiplexer (BUT MUX) with the contents of the Microbranch field (UBF) by a logical OR operation.

The target assignment construct allows the programmer to specify a base address (the contents of the UBF field) and the offset (output of the BUT MUX) for each target associated with that base address. As an example of the use of the target assignment, consider the following microprogram excerpt, which expresses a four-way conditional branch.

```
ALPHA:
.BEGIN=00

A:
     NEXT, BUT(D14-00-EQ-0 D15),    ! BRANCH ON
          J/A0;                     !   POS, NEG, ZERO, NEGZERO

.CASE 0 OF ALPHA                    ! POSITIVE DIFFERENCE
A0:                                 !  NO ACTION
     NEXT, J/B;

.CASE 1 OF ALPHA                    ! NEGATIVE DIFFERENCE
A1:                                 !  HIT COUNTER
     P2-T, D_A-PLUS-B,
          R3-A,CSPB(ONE),
     P3,  WR(AB,L,A),
     NEXT, J/B;

.CASE 2 OF ALPHA                    ! ZERO DIFFERENCE
A2:                                 !  NO ACTION
     NEXT, J/B;

.CASE 3 OF ALPHA                    ! NEGATIVE ZERO
A3:                                 !  ERROR
     NEXT, J/ERR;

.ENDB ALPHA
```

In this excerpt, the four targets of the conditional branch are A0, A1, A2, and A3. The conditional branch instruction, labelled A, branches to the base address A0. The target-assignment construct begins with the line containing the .BEGIN construct, which establishes the name ALPHA, defines the offsets associated with that name, and allocates the target addresses. The lines containing the .CASE keyword then associate the possible targets of the conditional branch with one another and with the offset information. The target-assignment construct ends with the line containing the .ENDB keyword.

The information in the target-assignment construct allows the assembler to assign to each of the microinstructions designated as a branch target the address that is formed by OR-ing the base address with the offset indicated by the case number. For example, if the assembler selects the address 6140 for the base address, then the other targets are assigned as follows:

Target	Case Number	Offset	Address
A0	0	00	6140
A1	1	01	6141
A2	2	10	6142
A3	3	11	6143

This address assignment accomplishes the desired conditional branch for microinstruction A.

6.2.1 Syntax

branch-definition	branch-label : .BEGIN = constraint
constraint	mask { [low-address : high-address] }$^{1}_{0}$
mask	{ 0 | 1 | * }$^{16}_{1}$
case-microinstruction	.CASE case-number OF branch-label microinstruction
end-definition	.ENDB branch-label
low-address high-address	octal-integer
case-number	decimal-integer
branch-label	name

6.2.2 Interpretation

The interpretation of a target-assignment construct starts with the branch-definition. The constraint string is examined to determine the number of possible targets and the requirements on the base-address. A set of targets satisfying the mask is selected from the available addresses on the current page. The number of characters in the mask specifies the number of bits that are constrained in the address.

The assembler selects a base address that has 0's in the positions indicated by 0, 1's in the positions indicated by 1, and either 0 or 1 in the positions indicated by *. CASE 0 uses the base address. The remaining cases systematically use bit positions indicated by 0 in the mask. If an address-range is specified, the set of targets is selected from the given range and allocated. The set of reserved addresses and the constraint string are associated with the branch-label.

A case-microinstruction for a given branch-label is assigned an address by using the case-number to determine from the mask the appropriate offset and then OR-ing that offset with the base- address. The processing of the case-microinstruction following address assignment is exactly the same as that of an ordinary microinstruction.

The interpretation of a target-assignment construct ends with the end-definition. If any of the addresses in the set of addresses associated with the branch-label has not been allocated as the result of interpreting a case-microinstruction, then the address is returned to the available address pool when the end-definition is processed.

6.2.3 Restrictions

A case-microinstruction for a given branch-label must be given only within that branch-label's scope, which is delimited by the branch-definition and end-definition for that label.

The number of 0's, k, in a mask must lie in the range:

$$1 <= k <= 7$$

The case-number must lie within the range of values specified by the constraint given in the branch-definition for the associated branch-label. That is, the case-number, n, must lie in the range:

```
0 <= n <= 2**k - 1
```

The low-address must specify an address that is a legal base address for the given constraint. The high-address must specify an address that is greater than or equal to the highest possible constraint address.

An address must not be specified for a case-microinstruction.

The base address must be defined. That is, a case-microinstruction for case-number 0 must always be given.

6.2.4 Defaults

If an end-definition is not given for a branch-label, then the scope of the branch-label is assumed to extend to the end of the microprogram.

6.2.5 Semantics

The detailed semantics of the constraint, the address-range, target-assignment scope and ease-microinstruction are discussed in the following sections.

6.2.5.1 Mask - The mask specifies the set of possible bits that can be combined with a base-address by an OR operation to form the targets of a conditional branch. The number of 0's, k, in the mask determines the total number of possible branch targets, 2**k, and the position of the 0's within the mask determines the set of possible addresses that can be used for the targets.

For example, consider the following branch-definition:

```
BETA:
.BEGIN=010
```

The mask is '010'. The mask contains two 0's; consequently, four targets are possible. The mask contains a 1 in the second bit position; consequently, any address within the program bounds on the current page that ends with 2 is a candidate for the base address. If the current page begins with 6000 and ends with 6777 and the program bounds are specified to be 6200 through 7777, then the following addresses are all potential base addresses:

```
6202  6212  6222  6232  6242  6252  6262  6272
6302  ...
...
6702  6712  6722  6732  6742  6752  6762  6772
```

The assembler, in choosing a base address, determines which of the above addresses are not yet allocated and of those addresses, which can be OR-ed with the possible offsets to produce addresses that are also unallocated. That is, in order to select the address 6242 as the base address, the following target addresses must all be free:

Base Address	Offset	Target Address
6242	010	6242
6242	011	6243
6242	110	6246
6242	111	6247

If the assembler cannot find a set of addresses to satisfy the target assignment construct, then an error is reported and the assembly continues, but a valid load module cannot be produced.

Each character in a mask has a specific meaning as follows:

Character	Meaning
0	The base address should have a 0 in this position.
1	The base address should have a 1 in this position.
*	The base address can have a 0 or 1 in this position.

The asterisk (*) character allows the programmer to indicate bits that are known to always be 0 in the BUT MUX output for this branch and, in this way, to allow the assembler more freedom in its choice of addresses. If, in the example just given, an asterisk rather than a 1 is given in the branch-definition, then the number of potential base addresses doubles, as follows:

```
6200  6210  6220  6230  6240  6250  6262  6270
6202  6212  6222  6232  6242  6252  6262  6272

6300  ...
6302  ...

...
...

6700  6710  6720  6730  6740  6750  6760  6770
6702  6712  6722  6732  6742  6752  6762  6772
```

6.2.5.2 The Address-Range - The address-range is used to specify the range of addresses from which the targets of the target-assignment construct are to be chosen. If the programmer wants to absolutely assign the targets, then he can specify the base-address as low-address and the target that is formed by OR-ing a string in which each 0 of the mask is replaced by a 1 with the base-address as high-address. As an example of absolute assignment, consider the following branch-definition.

```
GAMMA:
.BEGIN=00101[6205,6237]
```

The base-address associated with GAMMA is 6105 and the targets are assigned as follows:

Case	Offset	Address
0	00101	6205
1	00111	6207
2	01101	6215
3	01111	6217
4	10101	6225
5	10111	6227
6	11101	6235
7	11111	6237

If any of these addresses is not free when the branch-definition given above is processed, then an error is reported and the assembly continues, but the resulting load module cannot be used.

6.2.5.3 The Scope Of The Target Assignment Construct - The scope of the target assignment construct starts with the branch-definition and ends either with the end-definition for the branch-label or, if an end-definition is not given, with the end of the microprogram.

A case-microinstruction for a given branch-label is only valid within the scope of its definition. For example:

```
DELTA:
.BEGIN=0010

.CASE 3 OF DELTA
D3:

.CASE 0 OF DELTA
D0:

.ENDB DELTA

...

.CASE 2 OF DELTA
D2:
```

The case-microinstructions for D3 and D0 lie within the scope of DELTA in the above example and are, therefore, interpreted correctly. The case-microinstruction for D2, however, lies outside the scope of DELTA and therefore, its branch-label is undefined.

6.2.5.4 Case-Microinstructions - Within the scope of a target assignment construct, case-microinstructions can be given in any order. Further, cases that are not used can be omitted. However the zeroth case, which corresponds to the base-address, must always be given.

When the branch-definition that starts the scope of the target assignment construct is processed, all the possible targets, as determined from the constraint, are reserved. If at the end of the scope, some of the reserved targets have not been allocated by a case-microinstruction, then they are returned to the general address pool and, consequently, may be allocated later for a microinstruction that has no relationship with the target assignment construct.

6.2.6 Discussion

The use of the target-assignment construct in connection with two fundamental types of conditional branching is discussed in the following two sections.

6.2.6.1 Looping - The repetition of a sequence of microinstructions based on a counter, is called looping. Looping is a special case of conditional branching that occurs commonly in microprogramming. The following microprogram excerpt, which multiplies I times J, illustrates the use of the target-assignment construct for looping. This excerpt assumes that the counter is loaded with -J.

```
IJLOOP:
.BEGIN=0
MAT11:
        P2-T,   CLK-SR,WCSB[0]-B,B,     !  SR <-- 0
        NEXT,   BUT(COUNT-IS-377),
                J/MAT12;
.CASE 0 OF IJLOOP
MAT12:
        P2-T,   A-PLUS-B,WCSB[0]-B,
                SR,CLK-SR,              ! SR <-- SR + I
        NEXT,   BUT(COUNT-IS-377),      ! LOOP ADDINT I FOR J TIMES
                J/MAT12;
.CASE 1 OF IJLOOP
MAT13:          ...                      ! CONTINUE PROCESSING
```

The target-assignment construct is used to specify the targets MAT12 and MAT13. The instruction MAT12 is repeated J times. After the J'th execution of that instruction, the output of the BUT MUX is 1 and control passes to the instruction MAT13.

The microprogram from which this excerpt is taken is reproduced in full in Section 7.1.

6.2.6.2 Switching - The choice of one of a set of targets is called switching. Sometimes, control separates for a single calculation and then returns to a common point, as illustrated in the example of Section 7.2. Sometimes, control separates to perform totally different processing, as illustrated in the following example:

```
DECODE:
.BEGIN=000

DISPATCH:
        NEXT,   BUT(IR 5-3 ),
                J/BRANCH

.CASE 0 OF DECODE
BRANCH:
        NEXT,   J/MATRIX_ADD;           !  ENTRY FOR XFC 0

.CASE 1 OF DECODE
BRANCH1:
        NEXT,   J/MATRIX_MULT;          ! ENTRY FOR XFC 1

.CASE 2 OF DECODE
BRANCH2:
        NEXT,   J/MATRIX_INVERT;        !  ENTRY FOR XFC 2

.CASE 3 OF DECODE
BRANCH3:
        NEXT:   J/MATRIX_DIAG;          !  ENTRY FOR XFC 3

.CASE 4 OF DECODE
BRANCH4:
        NEXT,   J/LINK;                 !  ENTRY FOR XFC 4
```

The target-assignment construct in the above example is used to specify the switch points for five separate microprograms.

6.2.7 Guidelines

In order to satisfy a target-assignment construct, the assembler must be able to select a set of addresses that have a given relationship to one another. Therefore, the placement of the branch-definitions is important. They should be placed so that the necessary addresses are available to the assembler; that is, close to the beginning of the page to which they apply.

6.3 THE ENTRY POINT MECHANISM

To branch to a microprogram within the Writable Control Store, the programmer uses an XFC instruction in the main memory program. The XFC instruction can be used to branch to any one of eight possible microprograms. The programmer designates an entry point within the microprogram by the use of the DISPCH target assignment construct, as follows:

```
Instruction     Entry Point

XFC 0           .CASE 0 OF DISPCH
XFC 1           .CASE 1 OF DISPCH
..
XFC 7           .CASE 7 OF DISPCH
```

The branch-definition for the DISPCH target assignment construct is contained in the dispatch-file. Therefore, if an entry point is specified in a microprogram, the assembly input file list must include the dispatch-file before any files containing user-actions.

For example, suppose a program has two entry points, ENTA and ENTB. The programmer wants to issue an XFC 0 to enter at ENTA and an XFC 1 to enter at ENTB. He writes his user-actions as follows:

```
.CASE 0 OF DISPCH
ENTA:     ...
          ...
.CASE 1 OF DISPCH
ENTB:
...
.END
```

The dispatch-file is described in Appendix C.

CHAPTER 7

EXAMPLES

This chapter gives two complete microprograms to illustrate the use of the MICRO-11/60 assembler and the 11/60 predefinition language.

The first example is a threshold check program and the second example is a matrix addition program. These applications were chosen because the associated microprograms are short. Typically, microprograms are longer and more general than the programs reproduced here. However, these examples are complete microprograms; they illustrate the form and content of 11/60 microprograms.

7.1 EXAMPLE 1 - THRESHOLD CHECK

The example given in this section performs a threshold check. The microprogram accepts a list of positive values and produces, as its result, a count of the number of values in that list that exceed a specified threshold. The threshold check microprogram locates its inputs and outputs by the use of registers.

The threshold check program is an example of a well-documented microprogram. The inputs and outputs of the program are given in the comments that precede the program. The logical sections of the program are separated by comments that describe the purpose of the section. The actions performed by each microinstruction are described by trailing comments on the microinstruction line.

Two target assignment constructs are used in the threshold check microprogram, namely: SRCHP and THCMP. SRCHP illustrates the use of the target assignment construct for looping and THCMP illustrates its use for switching.

The construct SRCHP is used to control the loop that processes each element in the input list. As long as more elements remain in that list, control returns to CASE 0 of that construct. When the input list is exausted, control passes to CASE 1, which returns to the base machine.

The construct THCMP is used in the comparison of the value of each item in the input list with the threshold. The value of the input item determines to which of the four possible cases control passes.

```
.TITLE QUANTIZE
.IDENT /Q3/
.TOC      DATA POINTS THRESHHOLD CHECK

!
!         INPUT TO THRESHHOLD INSTRUCTION.
!
!         R0 - BASE ADDRESS OF THE LIST OF POINTS.
!              ALL POINTS ARE POSITIVE NUMBERS.
!         R1 - NUMBER OF POINTS IN SAMPLE.
!         R2 - THRESHHOLD VALUE.
!
!         OUTPUT AT TERMINATION OF THRESHHOLD INSTRUCTION.
!
!         R0 - USED AS WORK REGISTER.
!         R1 - USED AS WORK REGISTER.
!         R2 - THRESHHOLD VALUE.
!         R3 - NUMBER OF POINTS OVER THRESHHOLD.

SRCHLP:
.BEGIN=0[6240:6241]                    !TARGETS FOR LOOP TO SEARCH.
THCMP:
.BEGIN=00[6244:6247]                   !COMPARISON TARGETS WITH
                                       !THRESHHOLD.

.CASE 0 OF DISPCH                      !ENTRY POINT FOR XFC 0
XFCTHC:
  P2-T,    CLK-D,ZERO,
  P3,      WR(AB,L,A),R3-A,            !INITIALIZE COUNT OF POINTS TO
                                       !ZERO.
  NEXT,    J/XFCT1;

XFCT1:
  P2-T,    CLK-D,NOT-A,R1-A,
  P3,      WR(AB,L,A),                 !R1 <-- -R1 (COMPLEMENT OF
                                       !LENGTH).
  NEXT,    J/XFCT2;

XFCT2:
  P2,      LOAD-COUNTER,R1-A,          !LOOP COUNTER <--   OF POINTS.
  NEXT,    J/XFCT3;
```

```
!
!         START OF LOOP TO CHECK EVERY POINT AGAINST THRESHHOLD.
!


.CASE 0 OF SRCHLP
XFCT3:
  P1,     CLK-BA,R0-[A],DATI,              !INITIATE FETCH OF (R0)+
  P2-T,   CLK-D,A-PLUS-B,CSPB(TWO),
  P3,     WR(AB,L,A),                      !R0 <-- POINTS TO NEXT DATA
                                           !ITEM.
  NEXT,   J/XFCT4;

XFCT4:
  P3,     WR-CSP,CSPB(MD),                 !DATA ARRIVES FROM MEMORY.
  NEXT,   J/XFCT5;

XFCT5:
  P2-T,   CLK-D,A-MINUS-B,
          CSPB(MD),R2-A,                   !D <-- THRESHHOLD - DATA.
  NEXT,   J/XFCT6;

XFCT6:
  NEXT,   BUT(D14-00-EQ-0D15),             !BRANCH ON EQUAL,GREATER,LESS.
          J/XFCT7;

!
!         POSITIVE DIFFERENCE - THRESHHOLD > DATA POINT - NO OPERATION
!


.CASE 0 OF THCMP
XFCT7:
  NEXT,   BUT(COUNT-IS-377),               !LOOP BACK IF MORE NUMBERS
          J/XFCT1;                         !ELSE EXIT TO XFCT11

!
!         NEGATIVE DIFFERENCE - DATA POINT > THRESHHOLD - HIT THE
!         COUNTER.
!


.CASE 1 OF THCMP
XFCT8:
  P2-T,   CLK-D,A-PLUS-B,
          R3-[A],CSPB(ONE),
  P3,     WR(AB,L,A),                      !INCREMENT COUNTER BY ONE.
  NEXT,   BUT(COUNT-IS-377),               !LOOP BACK IF MORE NUMBERS
          J/XFCT1;                         !ELSE EXIT TO XFCT11
```

```
!
!         ZERO DIFFERENCE - DATA NOT OVER THRESHHOLD - NO OPERATION.
!


.CASE 2 OF THCMP
XFCT9:
  NEXT,   BUT(COUNT-IS-377),              !LOOP BACK IF MORE NUMBERS.
          J/XFCT1;                        !ELSE EXIT TO XFCT11

!
!         IMPOSSIBLE DIFFERENCE - ERROR RETURN TO ERROR ROUTINE.
!


.CASE 3 OF THCMP
XFC10:
  NEXT,   PAGE(0),BUT(SUBRB),             !EXIT TO ERROR POSITION.
          J/0000;


.CASE 1 OF SRCHLP
XFCT11:
  NEXT,   PAGE(0),BUT(SUBRB),             !END OF INSTRUCTION.
          J/BRA05;                        !RETURN FOR NEXT MACRO
                                          !INSTRUCTION.
.END
```

7.2 EXAMPLE 2 - MATRIX ADDITION

The matrix addition example adds the elements of the matrix M to the elements of the matrix N, on an element by element basis. That is, it performs the following computation:

$$M(m,n) = M(m,n) + N(m,n)$$

The number of elements in each row is given by I and the number of elements in each column by J.

Unlike the threshold check microprogram in which the input information is passed through the registers, the matrix addition example contains the input information in its calling sequence, as shown in the comments at the beginning of the program.

The matrix addition example uses two target assignment constructs. Both constructs are used for looping. The first, IJLOOP, is used to calculate the total number of elements; that is, to multiply I times J. The second, MNLOOP, is used to loop through each element of the matrix.

```
.TITLE  MATRIX ADDITION
.IDENT  /MP1A/

!
!                ***************************
!                *        XFC  CODE        *
!                ***************************
!       PC -->   *           M             *
!                ***************************
!                *           N             *
!                ***************************
!                *    I      *      J       *
!                ***************************
!

.TOC      START OF MATRIX ADDITION EXAMPLE.

IJLOOP:
.BEGIN=0
MNLOOP:
.BEGIN=0

.CASE 2 OF DISPCH                          !ENTRY POINT FOR XFC 2
MATADD:
  P1,     CLK-BA,PC-A,                     !INITIATE MEM(PC) READ:
  P2-T,   A-PLUS-B,CSPB(TWO),              !INCREMENT PC.
  P3,     WR(AB,L,A),DATI,
  NEXT,   J/MAT1;

MAT1:
  P3,     CSPB[MD]BUSDIN,                  !MD<--MEMORY - M (START OF M)
  NEXT,   J/MAT2;
```

```
MAT2:
  P1,      CLK-BA,PC-A,                    !INITIATE MEM(PC) READ.
  P2-T,    A-PLUS-B,CLK-D,CSPB(TWO),       !INCREMENT PC.
  P3,      WR(AB,L,A),DATI,
  NEXT,    J/MAT3;

MAT3:
  P2-T,    B,CSPB(MD),CLK-D,               !WORK1<-- M
  P3,      WR(A,H,A),R[SRC]-A,
           CSPB[MD]BUSDIN,                 !MD<-- MEMORY - N (START OF N)
  NEXT,    J/MAT4;

MAT4:
  P1,      CLK-BA,PC-A,                    !INITIATE MEM(PC) READ
  P2-T,    A-PLUS-B,CLK-D,CSPB(TWO),       !INCREMENT PC.
  P3,      WR(AB,L,A),DATI,
  NEXT,    J/MAT5;

MAT5:
  P2-T,    B,CSPB(MD),CLK-D,
  P3,      WR(A,H,A),R[DST]-A,             !WORK2<-- N
           CSPB[MD]BUSDIN,                 !MD<-- MEMORY - I:J
  NEXT,    J/MAT6;

!
! CALCULATE I X J AND LOAD IN COUNTER.
!

MAT6:
  P2-T,    CSPB(MD),CLK-D,CLK-SR,B,        !D <-- I:J , SR <-- I:J
           D[C]0,                          !D(C)<--0
  NEXT,    J/MAT7;

MAT7:
  P2-T,    CLK-D,D-RIGHT-8,D[C]D[C],A,     !D(C) <-- 0.
  P3,      WR(B,H,B),WCSB[0]-B,            !WORK3 <-- 0:I
  NEXT,    J/MAT8;

MAT8:
  P2-T,    CLK-D,SR,A,D[C]D[C],            !D <-- SR  (I:J)
  NEXT,    J/MAT9;

MAT9:
  P2-T,    D-SIGNEXT,NOT-A,CLK-D,          !D <-- COMPLEMENT OF J
  P3,      WR(B,H,B),WCSB[1]-B,            !WCSB(1) <-- COMPLEMENT OF J
  NEXT,    J/MAT10;

MAT10:
  P2,      WCSB[1]-B,
           LOAD-COUNTER,                   !COUNTER GET -J
  NEXT,    J/MAT11;

MAT11:
  P2-T,    CLK-SR,WCSB[0]-B,B,             !SR <-- I FROM WORK3
  NEXT,    BUT(COUNT-IS-377),              !CHECK IF = 1 TARGETS MAT12,13
           J/MAT12;
```

```
.CASE 0 OF IJLOOP
MAT12:
  P2-T,    A-PLUS-B,WCSB[0]-B,
           SR,CLK-SR,                    !SR <-- SR + I
  NEXT,    BUT(COUNT-IS-377),            !LOOP ADDINT I FOR J TIMES.
           J/MAT12;


.CASE 1 OF IJLOOP
MAT13:
  P2-T,    CLK-D,SR,NOT-A,
  P3,      WR(B,H,B),WCSB[1]-B,          !WCSB(1) GETS -I*J
  NEXT,    J/MAT14;

MAT14:
  P2,      WCSB[1]-B,
           LOAD-COUNTER,                 !COUNTER GETS -(I*J)
  NEXT,    J/MAT15;

!
! LOOP TO ADD EACH TERM OF MATRIX M AND N
!


.CASE 0 OF MNLOOP
MAT15:
  P1,      CLK-BA,R[SRC]-A,              !INITIATE MEM(M) READ.
           DATI,                         !WORK1 CONTAINS N
  NEXT,    J/MAT16;

MAT16:
  P3,      CSPB[MD]BUSDIN,               !MD <-- MEM(M)
  NEXT,    J/MAT17;

MAT17:
  P1,      CLK-BA,R[DST]-A,              !INITIATE MEM(N) READ.
  P2-T,    A-PLUS-B,CSPB(TWO),           !INCREMENT N
  P3,      WR(A,H,A),DATI,
  NEXT,    J/MAT18;

MAT18:
  P2-T,    CLK-SR,B,CSPB(MD),            !SR <-- MEM(M)
  P3,      CSPB[MD]BUSDIN,               !MD <-- MEM(N)
  NEXT,    J/MAT19;

MAT19:
  P2-T,    CLK-D,A-PLUS-B,SR,            !D <-- MEM(M) + MEM(N)
           CSPB(MD),
  NEXT,    J/MAT20;
```

```
MAT20:
  P1,      CLK-BA,R[SRC]-A,              !INITIATE MEM(M) WRITE.
           DATO,
  NEXT,    J/MAT21;

MAT21:
  NEXT,    J/MAT22;                      !DATA IS WRITTEN FROM D TO
                                         !MEMORY.

MAT22:
  P2-T,    A-PLUS-B,R[SRC]-A,
           CSPB(TWO),CLK-D,              !INCREMENT M
  P3,      WR(A,H,A),
  NEXT,    BUT(COUNT-IS-377),            !LOOP UNTIL ALL TERMS HAVE
           J/MAT15;                      !BEEN SUMMED. (TARGETS
                                         !MAT15,23)


.CASE 1 OF MNLOOP
MAT23:
  NEXT,    PAGE(0),BUT(SUBRB),           !RETURN TO BASE MACHINE FOR
           J/BRA05;                      !NEXT INSTRUCTION.

.END
```

PART III

THE MICROPROGRAM LOADER: MLD

Contents

CHAPTER 8

MICROPROGRAM LOADER

This chapter describes MLD, the Microprogram LoaDer. MLD is a program that loads the 11/60 Writable Control Store.

The functions of the loader are described in the following section. Then, the loader input, the microprogram object module, is discussed and illustrated.

8.1 LOADER FUNCTIONS

MLD performs three-functions in loading the Writable Control Store, namely:

- The initialization of the Writable Control Store to a special pattern.
- The loading of the resident section of the Writable Control Store.
- The loading of the set of object modules that make up the microprogram.

The following sections describe each of these activities.

8.1.1 Initialization

Before loading any information, MLD initializes the entire Writable Control Store, starting at location 6000 and continuing through location 7777, to the default initialization pattern. Any programs previously loaded into the Writable Control Store are destroyed by this initialization process.

The default initialization pattern is the following microword:

```
4      4         3         2         1
765432109876543210987654321098765432109876543210

000000000000000011000000000000000000111000000001110
```

After the execution of the loader, any microaddress that is not explicitly loaded contains this pattern.

If control passes to a word that contains the default initialization pattern, then the execution of that word causes a transfer to the resident section address 6016, which begins an error routine. This error routine handles the case in which a wild branch sends control to an illegal address. The user can provide an error routine for this case or can rely on the default handling, which exits to the console as if a halt instruction was encountered.

8.1.2 Loading The Resident Section

After initialization, MLD loads the resident section specified by the programmer. Usually, the standard resident section MICPAK is specified.

The contents of the standard resident section MICPAK supplied by DIGITAL are described in Appendix D of the "PDP-11/60 Microprogramming Specification."

8.1.3 Loading The Microprogram

After loading the resident section, MLD loads the microprograms specified by the user. MLD can load any number of object modules and these object modules can coexist in the Writable Control Store as long as the address space occupied by each module is consistent with the address space occupied by the other modules. The address space occupied by an object module is determined by the bounds given with the .BOUNDS keyword at the time the microprogram is assembled.

In loading the Writable Control Store, MLD constructs the address to be loaded from the lower bound and the offset and then loads the bits associated with that offset into the calculated Writable Control Store location. If an object module uses the same location as used in an object module loaded earlier in the load sequence, then the location is reloaded with the contents given in the later object module.

The fact that MLD permits words of the Writable Control Store to be rewritten is convenient for updating programs. However, the programmer must be careful, in that case, to specify the object modules in the correct order, so that the last word loaded into the given address is the expected one.

8.2 THE MICROPROGRAM OBJECT MODULE

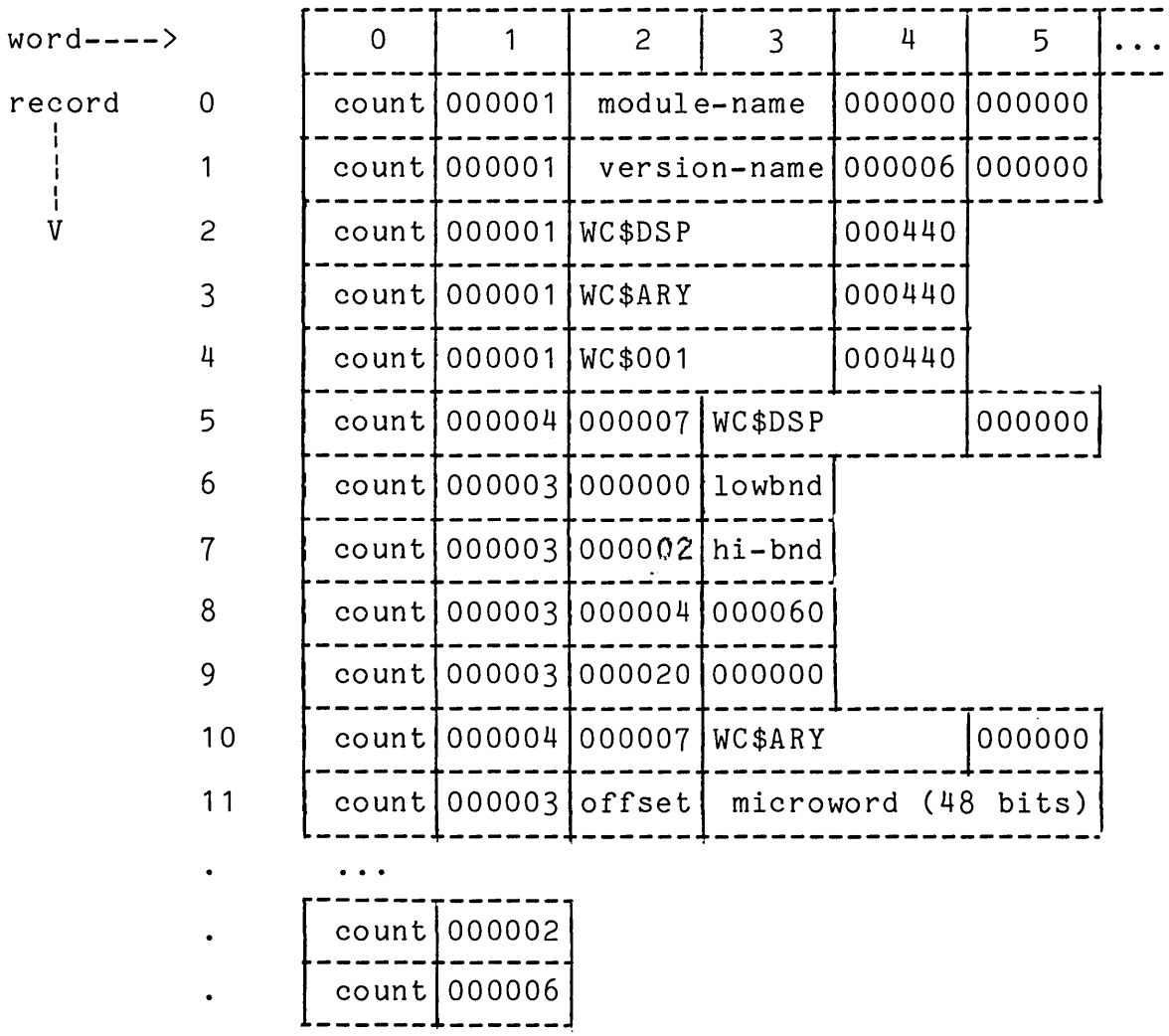
The microprogram object module has a format that is compatible with the standard macro object module produced by the translators operating in the RSX-11M system. However, the microprogram object module is not as general as a macro object module.

8.2.1 Microprogram Object Module Format

An 11/60 microprogram object module is made up of a sequence of records. Five different types of records are used, namely: GSD, RLD, TXT, end-GSD, and end-module. The meaning and format of each of these record types are described in Appendix B of the "RSX-11M Task Builder Reference Manual" (DEC-11-OMTBA-A-D).

The following diagram shows the format of the microprogram object module for the RSX-11M system. The first word is a count of the number of bytes that follow in the record. The second word indicates the type of record. The type of record determines the meaning of the words that follow.

The diagram shows the words on the horizontal axis and the record number on the vertical axis. The data is displayed as it appears in a dump.



The microprogram object module begins with the four GSD (type 1) records shown. The first record contains the program title, as extracted from the title-string supplied with the .TITLE keyword. The second record contains the version identification, given with the .IDENT keyword in the source microprogram. The third and fourth records declare the section names WC$DSP, for the WC$DSP area, and WC$ARY, for the Writable Control Store array storage.

Following these four standard records, a GSD record is given for the microprogram. The partition-name has the form WC$000.

After the GSD records, an RLD (type 4) record for the dispatch area is given. The text (type 3) records following that RLD give the lower and upper bound for the program and the dispatch table.

Following the RLD and TXT records for the section WC$DSP, an RLD record for the section WC$ARY is given. The TXT records following give the offset, relative to the lower bound, at which the microinstruction is to be loaded and the contents of the microinstruction. The offset is given in bytes and since each microinstruction occupies 6 bytes, the microinstruction address is calculated by the following formula:

microinstruction-address = lower-bound + offset/6

The microprogram object module concludes with an end-GSD (type 2) record and an end-module (type 6) record.

The object module for the sample program REGEX given in Section 14.4.9, is as follows:

word---->		0	1	2	3	4	5	...
record	0	000012	000001	REGEX		000000	000000	
¦	1	000012	000001	R1V1		000006	000000	
¦	2	000012	000001	WC$DSP		000040	000000	
V	3	000012	000001	WC$ARY		100040	000000	
	4	000012	000001	WC$000		100040	length	
	5	000012	000004	000007	WC$DSP		000000	
	6	000006	000003	000000	014000			
	7	000006	000003	000002	017377			
	8	000006	000003	000004	000061			
	9	000034	000003	000020	dispatch information ...			
	10	000012	000004	000007	WC$ARY		000000	
	11	000012	000003	000220	170232	004000	030200	
	12	000012	000003	001400	171612	010003	130201	
	13	000012	000003	001406	171012	010003	134003	
	14	000002	000002					
	15	000002	000006					

PART IV

THE MICROPROGRAM DEBUGGING TOOL: MDT

Contents

CHAPTER 9

INTRODUCTION

The MicroDebugging Tool MDT is a stand-alone program that provides an efficient tool for debugging 11/60 microprograms. Using MDT, the programmer can monitor the execution of his microprogram. He can set breakpoints, examine and change data or instructions in main or micro memory, and alter the control of the program.

MDT is intended for debugging microprograms. Usually, the program to be debugged consists of a small main memory program and a microprogram. The main memory program's purpose is to call the microprogram and, in some cases, provide data for the microprogram to manipulate. MDT takes over the machine and controls all I/O vectors and, consequently, all the interrupts. Therefore, the processing that can be done by the main memory program is limited. It cannot, for example, perform any input or output unless the programmer makes special provisions for handling I/O.

Because MDT is used to debug microprograms, it must save the state of the machine. The following section describes the saving of the machine state. After that discussion, an MDT session is considered and some general remarks are made on the MDT command line.

9.1 THE MACHINE STATE

Whenever a breakpoint occurs, MDT saves the state of the machine so that it can restore the state before continuing the execution of the program under test. The machine state is saved in a table called the microstate table.

9.1.1 The Microstate Table

The microstate table has two logical sections. The first section contains the datapath registers and the second section contains the PDP-11 registers.

In the following table, the name of each register, the number of bits occupied by that register, and a brief description of the meaning of the register are given.

TABLE 9-1
Microstate Table

Datapath Registers		
Register-name	Bits	Description
$D	16	D register
$DC	1	D(C) register
$SR	16	Shift register
$An	16	A scratchpad registers n is an octal integer starting with 0 and continuing through 37
$Bn	16	B scratchpad registers n is an octal integer starting with 0 and continuing through 37
$Cn	16	C scratchpad n is an octal integer starting with 17 and continuing through 0
$BA	16	Bus address register
$IR	16	Instruction register
$CNT	6	Count register
$FLAG	8	Flag register
$UCON	17	UCON control register
$RES	4	Residual control register
$RET	16	Return register
PDP-11 Registers		
Register-name	Bits	Description
$n	16	General registers n is an integer starting with 0 and continuing through 7
$PSW	16	Program Status Word
$PC	16	Program Counter
$ACn	32	Floating point registers n is an integer starting with 0 and continuing through 5
$RLn	16	Relocation registers n is an integer starting with 0 and continuing through 7
$FPS	16	Floating point status word
$FEC	16	Floating point error code
$FEA	16	Floating point error address

9.1.2 Restoring The Machine State

The type of breakpoint determines the amount of the machine that must be restored before execution of the program under test can resume. At a main memory breakpoint, MDT restores the PDP-11 registers from the microstate table. At a microprogram breakpoint, MDT restores the datapath registers, as well as the PDP-11 registers.

Three datapath registers cannot be preserved by MDT. These registers are $UCON, $RES, and $RET. The contents of the microstate table for these three registers is the default value used by MDT when restoring the state of the machine.

If the programmer wants to change the default value for any of these unpreserved registers, he can do so by the open-bits command. Since MDT never writes into the unpreserved locations, the default value is not changed. Then, each time that execution of the program is resumed, that default value is copied into the register in question.

9.2 A DEBUGGING SESSION

A debugging session consists of a number of interactions between MDT and the programmer, in which the programmer observes, and sometimes changes, the characteristics of the program under test.

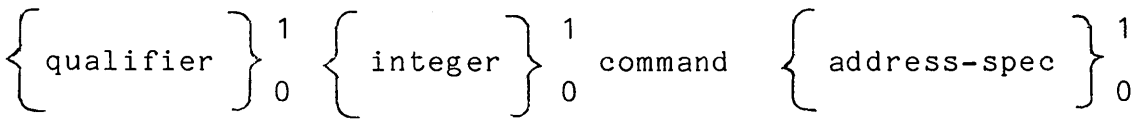
MDT prompts for a command by typing:

```
MDT>
```

In response to this prompt, the programmer types a command line. The following section considers some general properties of the command line.

9.3 THE COMMAND LINE

An MDT command line can contain four possible items, as shown in the following diagram:



The command must always be present in a command line, but the presence, or absence, of the other three items depends on the command. At most, two, out of the three optional items, are required for any command.

The permissable syntax and the meaning of that syntax are given for each type of command in the following chapters. Some general remarks about the form of the command, however, can be made here.

The following sections discuss the four items of the command line, starting with the command and continuing with the address-spec, the qualifier, and the integer.

9.3.1 Commands

The command mnemonics recognized by MDT are summarized in the following list. For each command mnemonic, a brief description is given.

Command	Description
O	Used to open, examine, and possibly change, the contents of bits in main memory or the Writable Control Store.
OB	Used to open, examine, and possibly change, a byte in main memory.
OC	Used to open, examine, and possibly change, a character in main memory.
B	Used to set a breakpoint; or, with the qualifier '-', to delete a breakpoint; or, with the qualifier '?', to list the current breakpoints.
P	Used to proceed after a breakpoint has halted execution.
D	Used to add an item to a display list; or, with the qualifier '-', to delete an item from the display list; or, with the qualifier '?', to list the current display items on the display list.
G	Used to start the execution of the program under test.
R	Used to reset all breakpoints and clear the display list.
L	Used to reload the microprograms into the Writable Control Store.

The first three commands (O, OB, and OC) are described in Chapter 10, "The Open Commands". The next two commands (B and P) are given in Chapter 11, "The Breakpoint Commands". The next command (D) is described in Chapter 12, "The Display Commands". The last three commands (G, R, and L) are described in Chapter 13, "The Control Commands".

9.3.2 The Address-Spec

An address-spec in an MDT command locates a sequence of bits in memory. The form of the address-spec is:

$$\text{address} \left\{ \text{bit-range} \right\}_{0}^{1} \left\{ \text{, relocation-register} \right\}_{0}^{1}$$

The address part of an address-spec can designate a main memory location, a micro memory location, or a register, depending on the first character of the address as indicated in the following list:

Address	Meaning
#n	A main memory address. The address n must lie in the range 0 <= n <= 17777.
n	A micro memory address. The address n must lie in the range 6000 <= n <= 7777 for a microinstruction or in the range 0 <n n <n 5777 for a data (local store) item.
$x	A register. The register named x can be any of the names recognized by MDT, as indicated in the table of Section 9.1.1.

The bit range indicates the bits within the designated location in memory. A bit-range is expressed in the following way:

< left-bit : right-bit >

If a bit-range is not given, then the full word is assumed, namely:

<15:0>	For a main memory address
<47:0>	For a micro memory instruction address
<15:0>	For a micro memory data address
<n:0>	For a register, where n is the value of bits, from the table in Section 9.1.1, minus 1.

The relocation-register can be any one of the eight relocation-registers, namely $RL0, $RL1, ... $RL7. A relocation-register can only be used with a main memory address. If a relocation-register is present in an address-spec, the contents of that register is added to the specified address to determine the actual address.

The form of the address-spec that is permitted depends on the command. The most general address-spec is allowed in an open-bits command. Other commands, however, are more restrictive. For example, the open-byte and open-character commands only accept main memory byte addresses and the go-command only accepts a word address. The syntax of each command indicates the form of the address-spec that can be given.

9.3.3 The Qualifier

The qualifier changes the meaning of the command mnemonic. Two qualifiers can be used in MDT, as indicated in the following list:

Qualifier	Meaning
-	Negate the meaning of the breakpoint or display mnemonic, so that instead of setting a breakpoint or display, a breakpoint or display is deleted.
?	List the breakpoints or displays currently set.

A qualifier can only be used with the breakpoint command (B) or the display command (D).

9.3.4 Integer Field

The integer field is used as an alternative to the address-spec. When a breakpoint or display is set, it is associated with an identification number. This identification number can be explicitly assigned by the programmer. More often, however, it is assigned by MDT. To delete a breakpoint or display, the programmer can give either the identification number or the address-spec. Some programmers find the identification number more convenient because it usually involves less typing.

9.3.5 Examples

Some examples of MDT command lines are given in the following list. For each command line, a brief description is given.

Command Line	Description
O#46000 <5:2>	Open bits 5 through 2 of main memory address 46000.
B6200	Set a breakpoint at micro memory address 6200.
?D	List the display items currently on the display list.
-2D	Delete display item with identification number 2.
G#47010	Start execution at main memory address 47010.

The following chapters describe the MDT commands in detail.

CHAPTER 10

OPEN COMMANDS

Three open-commands are provided to examine, and possibly change, the contents of locations within main memory or the Writable Control Store (micro memory). The open-commands are given in the following syntax:

open-command	{ open-bits-command open-byte-command open-character-command }

The open-bits command is the most general open-command. It can be used to look at a specified number of bits in main memory, in micro memory, or in the machine registers. The other two open-commands are special purpose commands that can be used to access either a byte or a character in main memory.

The following sections describe each of the open-commands.

10.1 THE OPEN-BITS COMMAND

The open-bits command can be used to open a main memory address, a microinstruction address, or a machine register. The programmer types the character "O" followed by an address-spec to open and display the contents of an address. The address-spec identifies the type of address, the address, and, optionally, the bits within the word addressed. For example:

```
MDT>O#2034
```

The character "#" identifies the address 2034 as a main memory address. Since no bit-range is given, MDT assumes the full word is to be displayed. MDT repeats the address-spec and then types the contents of the bits specified in the address-spec, as follows:

```
#2034<15:0>  14332    (pause)
```

The (pause) on the above line indicates that MDT is waiting for the programmer's instructions after typing the address-spec and value. The programmer can at this point enter a new value. For example, to change bit 3 to a 0, the programmer types the new value 14322 as follows:

```
#2034<15:0>  14332  14322  (separator)
```

The separator in the above line tells MDT whether the programmer wants to open another address or to return to MDT command level for another command. If the programmer types carriage return (cr) for the separator, then MDT considers the command finished and prompts for another command. The sequence, in this case, is:

```
MDT>O#2034
#2034<15:0>  14332  14333
MDT>
```

If the programmer types a line feed (lf), then MDT continues the current command by opening the next consecutive address. The sequence is:

```
MDT>O#2034  (cr)
#2034<15:0>  14332  14333  (lf)
#2036<15:0>  1010  (lf)
#2040<15:0>  23322  (cr)
MDT>
```

10.1.1 Syntax

open-bits-command	0 address-spec { address-spec value { new-value }1 term }n 0 0 address-spec value { new-value }1 0
address-spec	{ macro-address-spec micro-address-spec register-address-spec }
macro-address-spec	macro-address { bit-range }1 { , relocation-register }1 0 0
micro-address-spec	micro-address { bit-range }1 field-indicator 0
register-address-spec	register-name { bit-range }1 0
bit-range	< left-bit { : right-bit }1 > 0
left-bit right-bit	decimal-number
field-indicator	< field-name >
term	{ (lf) | ^ | @ }

The valid register-names are given in the microstate table in Section 9.1.1 and the valid field-names are given in Section 10.1.4.2.

10.1.2 Interpretation

MDT interprets an open-bits-command in the following way:

The address-spec is used to determine the bits to be opened. If the address begins with the character "#", it is interpreted as a main memory address. If the address begins with a "$", it is interpreted as a register-address. Otherwise, it is interpreted as a micro-address.

If a bit-range is specified, then those bits within the specified address are designated. If a bit-range is not specified, then the bit-range is constructed from the highest and lowest bit of the address, as follows:

Address	Default Bit-Range
macro-address	<15:0>
micro-address	
instruction	<47:0>
data	<15:0>
register-name	register dependent (see table in Section 9.1.1.)

If a field-indicator is given in a micro-address-spec, then the bit-range associated with the field-name in the table of Section 10.1.4.2 is used to designate the bits to be opened.

The designated bits are displayed as an octal number, in which leading zeros are suppressed and right-bit is the zeroth bit.

If the programmer types a new-value, that value is interpreted as an octal number with leading zeros suppressed, converted to a bit sequence of the proper length, and used to replace the designated bits.

If a separator is given, then MDT opens an address according to the separator used, as indicated in the following list:

Separator	Action
(lf)	Open the next consecutive address: The next address is determined for a macro address by adding 2 to the current address; for a micro address by adding 1 to the current address; for a register by taking the next consecutive register in the table of Section 9.1.1.
^	Open the previous location. The previous macro address is determined by subtracting 2 from the current address; for a micro address by subtracting 1 from the current address; for a register address by taking the entry just before the current register in the register name table of Section 9.1.1.
@	Open the address indicated in the memory address currently open (indirect addressing). This address is formed for a macro address by taking the 16 bit contents of the current address; for a micro address by taking bits <8:0> of the contents of the current address; for a register address, by taking the contents of the register and interpreting it as a macro-address.

If the register contains more than 16 bits, the extra bits are truncated. If the register contains less than 16 bits, leading 0's are supplied.

A carriage return following a displayed value or a new-value terminates the command.

10.1.3 Restrictions

If a new-value is given, it must fit, after truncating any leading zeros, in the bits designated by the address-spec.

A micro-address for an instruction must lie in the range 6000 through 7777. A micro-address for a data (local store) item must lie in the range 0 through 5777.

A macro-address must be an even address in the range 0 through 177777.

The left-bit and right-bit must lie in the valid bit range for the address type, as follows:

```
macro-address      15 >= left-bit  >  right-bit >= 0

micro-address
   instruction     47 >= left-bit  >  right-bit >= 0
   data            15 >= left-bit  >  right-bit >= 0
```

10.1.4 Defaults

If a right-bit is not given, then the right-bit is assumed to be the same as the left-bit and a single bit field is assumed.

If no bit-range is given, the full word is assumed.

If a new-value is not given, then the value displayed is not changed.

10.1.4.1 The Macro-Address-Spec - The macro-address-spec describes the bits to be opened in a main memory location. A main memory word consists of 16 bits, as indicated in the following diagram:

```
                1    1
                5432109876543210
address         ----------------
                |              |
                ----------------
```

Suppose the main memory address 10234 contains the following bits:

```
                1    1
                5432109876543210
                ----------------
10234           |1011011101111011|
                ----------------
```

To examine the entire word, the programmer types the character "#" to indicate a macro-address and the main memory address in response to the MDT prompt, as follows:

```
MDT>O#10234
#10234<15:0> 133573
```

To examine a set of bits within the word, he types the character "#", followed by the address, followed by a bit-range.

```
MDT>O#10234<8:5>
#10234<8:5>  13
```

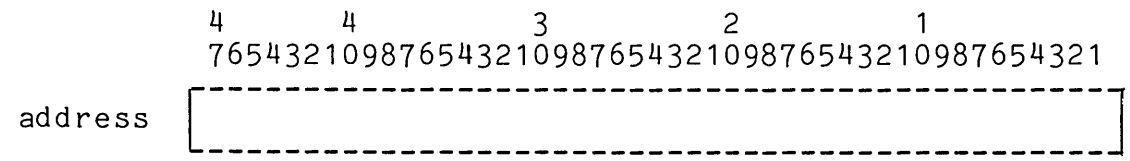
To change a set of bits within the word, he gives a new-value, as follows:

```
MDT>O#10234<2:0>
#10234<2:0> 3 0
```

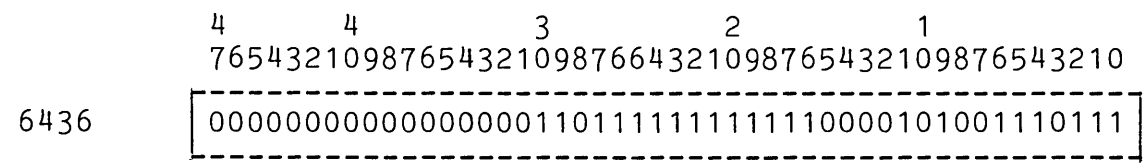
After the above interaction the contents of main memory location 10234 contains the following bits:

```
                1    1          0
                5432109876543210
                ----------------
10234           |1011011101111000|
                ----------------
```

10.1.4.2 The Micro-Address-Spec - The micro-address-spec describes the bits to be opened in the Writable Control Store. A Writable Control Store location consists of 48 bits, as indicated in the following diagram:



Suppose the Writable Control Store location 6436 contains the following bits:



To display the entire word, the programmer types the address without a bit range, as follows:

```
MDT>06436
6436<47:0>  15777605167
```

To display a set of bits within the word, the programmer types the address followed by a bit-range:

```
MDT>06436<12:7>
6436<12:7>  24
```

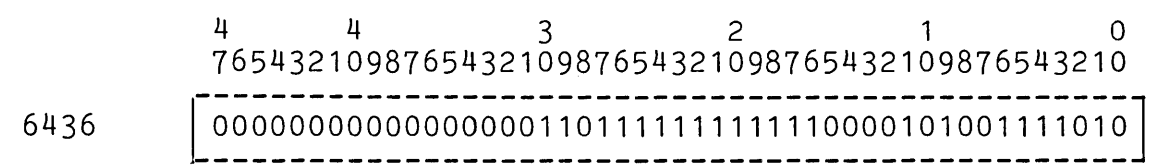
To display a predefined microinstruction field within the word, the programmer types the address followed by a field-indicator:

```
MDT>06436<UPF>
6436<8:0>  163
```

To change a set of bits, he types a new-value, as follows:

```
MDT>06436<UPF>
UPF<8:0> 163  172
```

After the above interaction, location 6436 contains the following bits:



The field-names that can be given in a micro-address-spec are given in the following table. For each field-name, the associated bit-range and a brief description are given.

Field-Name	Bit-Range	Meaning
ALU	<47:44>	ALU function field
BEN	<43:42>	B scratchpad enable field
BSEL	<41:40>	B scratchpad select field
AEN	<39:38>	A scratchpad enable field
ASEL	<37:36>	A scratchpad select field
XMUX	<36>	XMUX select field
CMUX	<37:36>	CMUX select field
RIF	<35:33>	Register immediate field
COUT	<32:30>	Carry select field
WHEN	<29>	Time to clock select field
CLKD	<28>	Clock D register
CLKSR	<27>	Clock SR register
CLKBA	<26>	Clock BA register
SCC	<25>	Set condition codes
BGB	<24>	BGB field
BUSBOX	<23>	Bus or Box field
BMUX	<24>	BMUX select field
BC	<22:20>	BC field
AMUX	<22:20>	AMUX select field
WRCSP	<19>	Write C scratchpad from DIN
HILO	<18>	High/Low select field
WRSEL	<17>	A/B write select field
WRSP	<16:15>	Rewrite select field
MOD	<14>	Field select
CLKCOUNT	<16>	Clock count register field
CLKRES	<18>	Clock RES register field
UBF	<13:9>	UBF field
UPF	<8:0>	Next address field

10.1.4.3 The Register-Address-Spec - The register-address-spec describes the bits to be opened either in a datapath register or a PDP-11 register. The register-names recognized by MDT are given in the Microstate Table in Section 9.1.1.

10.1.4.4 New-Values - The programmer can make a temporary change in a program by supplying a new-value in an open-command. The new-value replaces the bits designated in the address-spec. However, if the program is reloaded, the old values are restored.

New-values can be entered without leading zeros. MDT supplies sufficient leading zeros to fill out the new value to the number of bits specified by the address-spec and then replaces those bits within the address. If the programmer gives a new-value that contains more bits than given by the address-spec, then the value is truncated to fit the bits and no error is reported.

10.1.4.5 Line Terminators - The line terminator (term) communicates to MDT whether or not the programmer wants to open another address, which is related to the current address. If the programmer types the carriage return line terminator, then MDT considers the open-command complete and prompts for the next command. However, if the programmer types a line feed, circumflex (^), or "@", then MDT opens another address.

To open the next consecutive address, the programmer types the line feed line terminator, as follows:

```
MDT>6240
6240<47:0> 334564  (lf)
6241<47:0> 110200004007  (lf)
6242<47:0> 104 (cr)
```

To open the previous address in memory, the programmer types the circumflex terminator, as follows:

```
MDT>10234<10:8>
#10234<10:8>  2  ^
#10232<10:8>  0  ^
#10230<10:8>  1  (cr)
```

Observe that the bits designated in the bit range are used for each address opened.

To open the indirect address, the programmer types the "@" line terminator, as follows:

```
MDT>6240
6240<47:0>  334564  @
6564<47:0>  4457030344  @
6344<47:0>  202000  (cr)

MDT>#10236
10236<15:0>  4430  @
4430<15:0>   4436  @
4436<15:0>   10234  (cr)
```

The line terminators can be mixed in an open-command sequence. Consider the case in which the programmer has a linked list of word pairs. The first word of a pair contains the value and the second word contains the link. To examine the list, he begins by opening the first word of the first word pair, then the second word, then using the link in that word, the next pair in the list, as follows:

```
MDT>#4000
#4000<15:0> 0  (lf)
#4002<15:0> 4016  @
#4016<15:0> 3  (lf)
#4020<15:0> 4110  @
#4110<15:0> 2  (lf)
  ...
```

When the address-spec gives a register address, then the line feed and circumflex line terminators can be used to display the other registers in the microstate table. If a line feed terminator is used, then the next register in the microstate table is opened and if a circumflex is used, then the previous register in the microstate table is opened. An example of the use of the line feed terminator with a register address is as follows:

```
MDT>$BA
$BA<15:0>      12      (lf)
$IR<15:0>       3      (lf)
$CNT<5:0>       2      (lf)
$FLAG<7:0>      0
```

10.2 THE OPEN-BYTE COMMAND

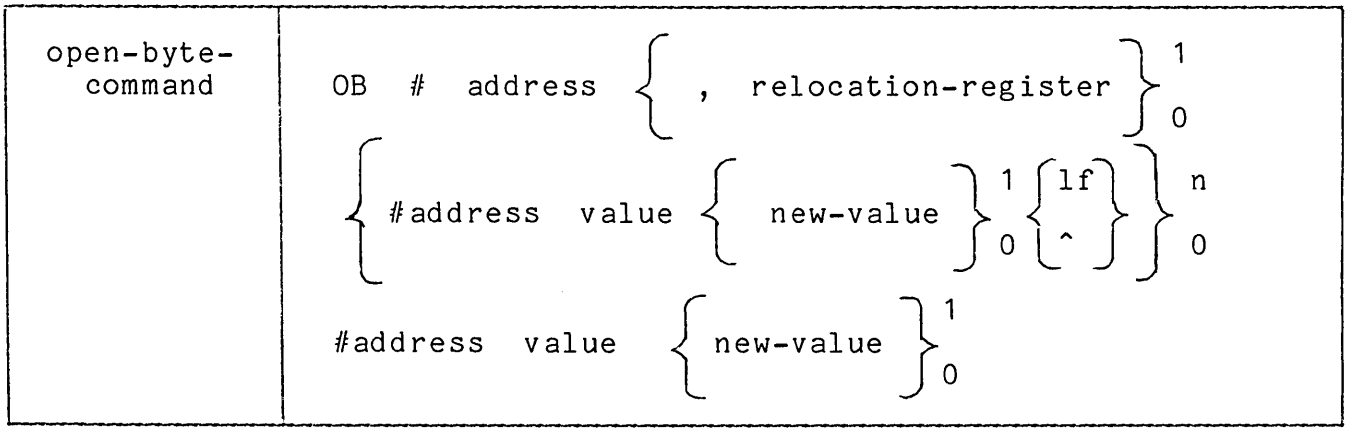
The open-byte command is used to open a byte of main memory. Suppose, for example, that the programmer wants to examine main memory locations 7002 through 7012 on a byte-by-byte basis and change the first zero byte to the value 33. He opens the first location with an open-byte-command and then, to locate and change the first byte with value 0, uses the line feed line terminator to examine the following bytes, as follows:

```
MDT>OB#7002
#7002  302  (lf)
#7003  111  (lf)
#7004  123  (lf)
#7005  0  33  (cr)
```

The open-byte command can be used to open an odd address. For example:

```
MDT>OB#3345
#3345  22
```

10.2.1 Syntax



10.2.2 Interpretation

MDT interprets the open-byte-command in the following way:

The address locates the main memory byte to be opened. The contents of that byte are displayed as an octal number, with leading zeros suppressed.

If a new-value is given, that value is interpreted as an octal number with leading zeros suppressed and is used to replace the contents of the designated byte in memory.

If a line feed is used to terminate a line, then the next consecutive byte in memory is displayed. If a circumflex is used to terminate a line, then the previous byte is opened. If a carriage return is used, then the command is terminated.

10.3 THE OPEN-CHARACTER COMMAND

The open-character command is used to open a character in main memory. The open-character command accepts a byte address and displays the contents of that byte as an ASCII character.

As an example of the use of the open-character-command, consider a table of names in main memory. Each name consists of a sequence of ASCII characters terminated by the special symbol ":". Once the programmer gets a pointer to the beginning of a string in the table, he can use the open-character-command to examine the characters in the name, as follows:

```
MDT>OC#12340
#12340  A  (lf)
#12341  L  (lf)
#12342  P  (lf)
#12343  H  (lf)
#12344  A  (lf)
#12345  :  (cr)
```

Suppose he wants to change that entry in the table from ALPHA to ALPH1, he opens the appropriate character and changes it as follows:

```
MDT>OC#12344
#12344  A  1  (cr)
```

10.3.1 Syntax

open-char-command	OC # address { , relocation-register }$_{0}^{1}$ { #address char { new-char }$_{0}^{1}$ { lf / ^ }$_{0}^{n}$ #address char { new-char }$_{0}^{1}$

10.3.2 Interpretation

MDT interprets the open-char-command as follows:

The address locates a byte in main memory to be opened. The contents of that byte are displayed as an ASCII character. If the contents of the byte is not a printable character, a blank space is printed.

If a new-char is given, that character replaces the contents of the designated byte.

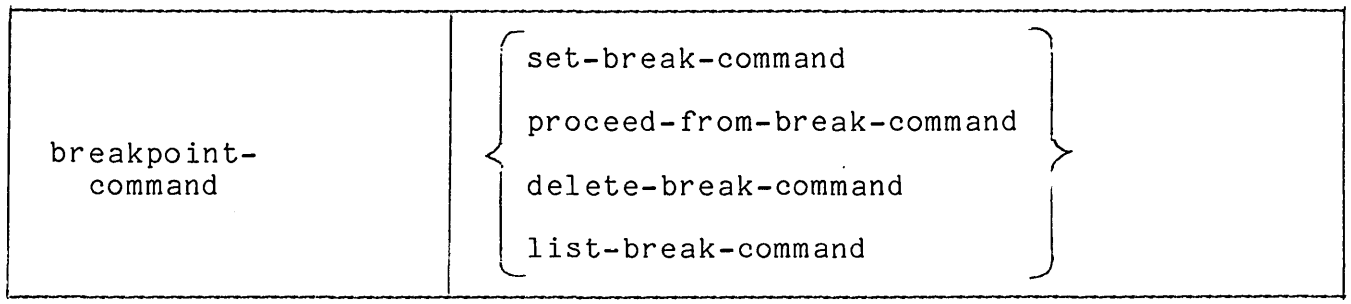
If a line feed is used to terminate a line, then the next consecutive byte in memory is displayed as an ASCII character. If a circumflex is used to terminate a line, then the previous byte is displayed as an ASCII character. If a carriage return is used, the command is terminated.

CHAPTER 11

BREAKPOINT COMMANDS

The breakpoint commands allow the user to set breakpoints in main memory or in the Writable Control Store, to continue execution after a breakpoint occurs, to selectively delete breakpoints, and to list the breakpoints currently set in the program.

The breakpoint commands are given in the following syntax:



Two items of information are maintained for active breakpoints, namely: the break-address and the repeat-count. The break-address is supplied by the set-break-command; it determines the address at which the breakpoint is taken. The repeat-count is supplied by the proceed-from-break-command; it determines the number of times the breakpoint must occur before execution halts at that breakpoint.

MDT keeps this information in the breakpoint list. The breakpoint list contains 16 entries, one for each possible breakpoint. The number of entries in this list limits the number of breakpoints that can be active at any one time. The break-id is the index into the breakpoint list. The break-id can be given in a set-break-command. However, if it is not supplied, MDT assigns the lowest available break-id.

Initially the breakpoint list is empty. Each time a set-break-command is executed, an entry is made in the breakpoint list. Suppose the programmer types the following two set-break-commands:

```
MDT>B#10200
MDT>2B6430
```

In the first set-break-command, no break-id is given and, therefore, MDT assigns the first available break-id, 0, to that breakpoint. In the second set-break-command, the break-id 2 is given. After the execution of these two commands, the breakpoint list looks as follows:

Break-Id	Break-Address	Repeat-Count
0	#10200	0
1	--	-
2	6430	0
3	--	-
...		
15	--	-

When a breakpoint occurs, MDT examines the repeat-count associated with that breakpoint. If the repeat-count is 0, then MDT prints the breakpoint message and halts. Suppose control passes first to the main memory address 10200. Then, the following message is printed:

```
BREAKPOINT NUMBER 0 AT ADDRESS #10200
```

To continue the execution of his program, the programmer types a proceed-from-break-command, which can contain a repeat-count. Suppose the programmer wants to ignore the next four occurrences of the breakpoint at 10200. To do this, he types the following command:

```
MDT>4P
```

MDT updates the repeat-count associated with that breakpoint in the breakpoint list. After the execution of the proceed-from-break-command, the breakpoint list looks as follows:

Break-Id	Break-Address	Repeat-Count
0	#10200	4
1	--	-
2	6430	0
3	--	-
...		
15	--	-

Suppose control passes again to 10200 in main memory and then to 6430 in the Writable Control Store. When control passes to 10200, the repeat-count is decreased by 1, but no message is printed and no halt occurs. When control passes to 6430, a message is printed and execution stops. The programmer can, at that point, print the breakpoint list by typing a list-break-command, as follows:

```
BREAKPOINT NUMBER 2 AT ADDRESS 16430
MDT>?B
```

MDT, in response to this command, types the breakpoint list:

```
BREAKPOINT NUMBER 0 IS SET AT ADDRESS 10200
     THE CURRENT REPEAT COUNT IS 3
BREAKPOINT NUMBER 2 IS SET AT ADDRESS 6430
     THE CURRENT REPEAT COUNT IS 0
```

By examining the breakpoint list, the programmer learns the current breakpoints that are set and the state of the repeat-count associated with each breakpoint. Suppose that, on the basis of this information, the programmer decides to delete the breakpoint at main memory location 10200 and then continue execution. He types the following commands:

```
MDT>-B#10200
MDT>P
```

The breakpoint list then looks as follows:

Break-Id	Break-Address	Repeat-Count
0	--	-
1	--	-
2	6430	0
...	--	-
15	0	0

The following sections discuss each of the breakpoint commands in detail.

11.1 THE SET-BREAK-COMMAND

The set-break-command is used to indicate the address at which a breakpoint is to be taken. When control passes to that address, MDT examines the repeat-count associated with the address in the breakpoint list. If the repeat-count is not zero, it is decremented by 1 and execution continues. If the repeat-count is zero, MDT prints a message identifying the breakpoint plus the contents of any display-items on the display list, and halts.

The display list and the MDT commands that add and delete entries from the display list are described in the next chapter. In this chapter, the display list is assumed to be empty.

As an example of the use of breakpoints, consider a program that dispatches, on a calculated value, to one of three possible paths. The programmer wants to examine some locations the first time control passes through each path. To do this, he sets a breakpoint at the beginning of each path, as follows:

```
MDT>B6500
MDT>B6720
MDT>B6400
```

Then, he starts the execution of his program. Suppose the first time control passes through the dispatch point it is sent to the path that begins with the microinstruction 6720. MDT prints the following message and halts.

```
BREAKPOINT NUMBER 1 AT ADDRESS 6720
```

The programmer can, at this point, open addresses to examine the state of the program, make temporary changes, set or delete breakpoints, or perform any of the other MDT commands.

11.1.1 Syntax

set-break-command	{ break-id } 1 / 0 B break-address
break-id	{ 0 | 1 | 2 | ... | 15 }
break-address	{ #000000 | #00002 | ... | #17776 / 6000 | 6001 | ... | 7777 }

11.1.2 Interpretation

In response to a set-break-command, MDT performs two actions, namely: including the breakpoint in the breakpoint list and altering the contents of the break-address so that MDT can handle the breakpoint. The following sections discuss these two actions.

11.1.2.1 Including The Breakpoint In The Breakpoint List - MDT enters the break-address given in the set-break-command with a repeat-count of 0 in the breakpoint list entry specified by the break-id.

If a break-id is given in the set-break-command, MDT uses that break-id as an index into the breakpoint list. If the entry indicated by the break-id has an associated break-address, indicating that it is a currently active breakpoint, then MDT deletes the old breakpoint address in the table and replaces that address by the break-address given in the set-break-command.

If a break-id is not given, MDT assigns the first available break-id. If no break-id is available, indicating that 16 breakpoints are active, MDT reports an error and rejects the command.

11.1.2.2 Planting The Subroutine Call - To set a breakpoint, MDT alters the contents of the break-address. It removes and saves the current contents and replaces the contents with a call to an MDT subroutine. When control passes to the break-address, the MDT subroutine is executed.

The MDT subroutine examines the repeat-count associated with the break-address in the breakpoint list. If the repeat-count is not zero, the subroutine decrements the repeat-count and execution proceeds. If the repeat-count is zero, the subroutine prints a message identifying the breakpoint, then prints the contents of any display-items on the display list, halts the execution of the program and prompts for the next MDT command.

11.1.3 Restrictions

Observe that, since MDT alters the contents of the break-address, an instruction that is dynamically modified by the program must not be given as a break-address.

Further, since certain operations in the 11/60 (such as reading from memory) take two cycles to complete, an MDT breakpoint should not be placed on the second microword of a microword pair.

11.2 THE PROCEED-FROM-BREAK-COMMAND

The proceed-from-break-command is used to resume program execution after a breakpoint. Execution stops just before the instructon at the break-address is executed. The proceed-from-break-command causes execution to resume at the instruction given in the set-break-command. Execution then continues until another breakpoint is reached or until the program halts for some other reasons.

In some cases, the programmer does not want to stop at every occurrence of a breakpoint. The proceed-from-break-command allows the programmer to give a repeat-count, which directs MDT to pass through n occurrences of the breakpoint but to halt at the n+1th occurrence.

Suppose, for example, the programmer is debugging a program with a loop that executes approximately 200 times. The programmer plants a breakpoint within the loop. He wants the breakpoint to halt execution the first time the loop is executed and again as the loop is about to terminate. The first time the loop is executed, MDT prints the breakpoint message and halts, waiting for a command. The programmer types the following proceed-from-break-command:

```
BREAKPOINT NUMBER 0 AT ADDRESS 6436
MDT> 200P
```

The proceed-from-break-command instructs MDT to ignore the next 200 occurrences of the breakpoint. At the 201st occurrence of the breakpoint, MDT prints the following message and halts:

```
BREAKPOINT NUMBER 0 AT ADDRESS 6436
```

If the programmer wants to halt the 202nd occurrence, he types the following proceed-from-break-command:

```
MDT> P
```

If a repeat-count is not given, it is assumed to be 0. The above command, therefore, instructs MDT to stop at the next, in this case 202nd, occurrence of the breakpoint.

11.2.1 Syntax

proceed-from-break-command	{ repeat-count } 1 0 P
repeat-count	{ 0 | 1 | 2 | ... | 511 }

11.2.2 Interpretation

In response to a proceed-from-break-command, MDT performs two actions, as follows:

MDT enters the repeat-count in the breakpoint list for the breakpoint that has halted execution. If no repeat-count is given in the proceed-from-break-command, MDT enters the repeat-count 0.

MDT causes the execution of the program to resume at the instruction that was contained in the break-address before the set-break-command.

11.3 THE DELETE-BREAK-COMMAND

The delete-break-command is used to remove a breakpoint that was set by a set-break-command.

As an example of the use of the delete-break-command, consider the case in which the programmer has set breakpoints in his program at addresses #10200, #12331, #16050, and #14443.

During the debugging, the programmer finds that control passes through the breakpoint #12331 in the expected way and, therefore, the breakpoint is no longer necessary. The programmer deletes that breakpoint as follows:

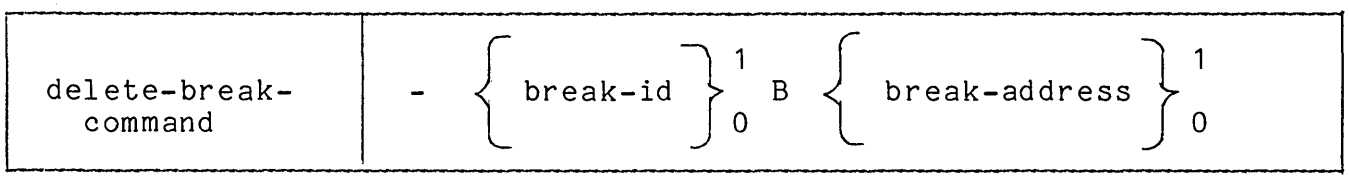
```
MDT>-B#12331
```

Later in the debugging session, the programmer wants to concentrate on a particular section of the program. He dispenses with all the existing breakpoints by the following command:

```
MDT>-B
```

The above command clears all breakpoints currently active in the program. The programmer can then add breakpoints in the area of special interest.

11.3.1 Syntax



11.3.2 Interpretation

In response to a delete-break-command, MDT deletes a breakpoint. That is, it removes the subroutine call placed in the breakpoint location by the set-break-command and restores the original contents of that location.

The form of the delete-break-command determines the breakpoint to be deleted.

If neither a break-id nor break-address is given, then MDT deletes all breakpoints currently set in the breakpoint list.

If only a break-id is given, then MDT deletes the breakpoint for the address associated with that break-id in the breakpoint list. If no address is currently associated with the break-id, then an error is reported and the command is rejected.

If only a break-address is given, then MDT searches the breakpoint list. If the specified break-address is found in the breakpoint list, that breakpoint is deleted. If it is not found, then an error is reported and the command is rejected.

11.3.3 Restriction

Either a break-id or a break-address can be given in a delete-break-command, but not both.

11.4 THE LIST-BREAK-COMMAND

The list-break-command is used to print the breakpoint list, which contains all the breakpoints that are currently active in the program. The programmer types:

```
MDT>?B
```

In response to this list-break-command, MDT types the breakpoint list in the format shown in the following example:

```
BREAKPOINT NUMBER 0 IS SET AT ADDRESS #6436
      THE CURRENT REPEAT COUNT IS 0
BREAKPOINT NUMBER 1 IS SET AT ADDRESS #12231
      THE CURRENT REPEAT COUNT IS 12
BREAKPOINT NUMBER 3 IS SET AT ADDRESS 6557
      THE CURRENT REPEAT COUNT IS 1
```

11.4.1 Syntax

list-break-command	? B

11.4.2 Interpretation

In response to the list-break-command, MDT types the breakpoint list, giving the break-id, break-address, and repeat-count for each active breakpoint in the following format:

```
BREAKPOINT NUMBER  break-id  IS SET AT ADDRESS  break-address
      REPEAT COUNT IS  repeat-count
```

CHAPTER 12

DISPLAY COMMANDS

The display commands allow the user to add address-specs and selectively delete them from the display list. The display list contains the set of address-specs whose contents are printed by MDT each time a breakpoint with a repeat-count of 0 occurs.

The display-commands are given in the following syntax:

display-command	{ set-display-command delete-display-command list-display-command }

The display list is very similar to the breakpoint list. Like the breakpoint list, it contains 16 entries, one for each possible display and the number of entries in the list limits the number of displays that can be active. The display-id is the index into the display list just as the break-id is the index into the breakpoint list and it is assigned either explicitly in a set-display-command or by MDT in a similar fashion.

Each entry in the display list contains the address and bits to be printed. The display list is empty initially. The set-display-command adds an entry to the list and the delete-display-command deletes an entry.

The following sections describe the display commands.

12.1 THE SET-DISPLAY-COMMAND

The set-display-command is used to add an address-spec to the display list. After the execution of a set-display-command, the address-spec given in the command is printed, as part of the display list, each time a breakpoint with repeat-count 0 occurs.

As an example of the use of the set-display-command, consider the case in which the programmer wants to examine, at each breakpoint, the contents of registers 2 and 7, the program status word, and the ALU field of microinstruction 6430. The programmer adds the address-specs that define his display needs to the display list by the following sequence of commands:

```
MDT>D$7
MDT>D$2
MDT>D$PSW
MDT>D6430<ALU>
```

When a breakpoint occurs, the display list is printed as follows:

```
BREAKPOINT NUMBER 0 AT ADDRESS #10010
DISPLAY
$7<15:0>  1012
$2<15:0>  54321
$PSW<15:0>  340
6430<ALU>  12
END OF DISPLAY
```

The address-specs are given in the display output according to their display-id. No display-id was given when the address-specs were added to the display list; consequently, the display-ids were assigned by MDT. The first address-spec entered was assigned the first available display-id 0; the second address-spec was assigned the display-id 1; and so on.

12.1.1 Syntax

set-display-command	{ display-id }1 D address-spec 0
display-id	{ 0 | 1 | 2 | ... | 15 }

12.1.2 Interpretation

In response to a set-display-command, MDT adds the address-spec to the display list entry specified by the display-id.

If a display-id is given, then the address-spec is entered into the display list at that position. If the display-id indicates a display list entry that has an address-spec, indicating that an active display is associated with that display-id, MDT overwrites the address-spec in the display list with the new address-spec from the set-display-command.

If a display-id is not given, MDT assigns the first available display-id. If no display-id is available, then MDT reports an error and rejects the command.

The address-spec is saved in the display list so that it can be used to locate the bits to be printed when the display list is output. The address-spec is described in connection with the open-bits-command in Section 9.1. Adding an address-spec to the display list directs MDT to perform the open-bits-command for that address-spec automatically each time a breakpoint is honored.

12.2 THE DELETE-DISPLAY-COMMAND

The delete-display-command is used to delete an address-spec from the display list. The delete-display-command can be used to clear the display list or to selectively remove address-specs from the display list.

Suppose that, as in Section 12.1, the programmer has added registers 2 and 7, the program status word, and the ALU field of microinstruction 6430 to the display list. After taking a few breakpoints, he finds that he is no longer interested in the contents of register 2. He can delete that display by the following delete-display-command:

```
MDT>-D$2
```

Subsequent breakpoints, then, no longer include register 2 in the display list. At the next breakpoint, the following display list is printed:

```
BREAKPOINT NUMBER 0 AT ADDRESS #1001
DISPLAY
$7<15:0> = 1012
$PSW<15:0> = 343
6430<ALU>  11
END OF DISPLAY
```

To delete all the address-specs, the programmer types the following delete-display-command:

```
MDT>-D
```

At the next breakpoint, the display list is empty and therefore, no display is printed. The MDT simply reports the breakpoint as follows:

```
BREAKPOINT NUMBER 0 AT ADDRESS #1001
```

12.2.1 Syntax

delete-display-command	- { display-id }[1 / 0] D { address-spec }[1 / 0]

12.2.2 Interpretation

MDT interprets a delete-display-command as follows:

If neither a display-id nor an address-spec is given in a delete-display-command, then MDT deletes all address-specs in the display list to return the display list to its initial empty state.

If a display-id is given, then MDT deletes the address-spec associated with that display-id. If no address-spec is associated with the display-id, then an error is reported and the command is rejected.

If an address-spec is given, then MDT searches the display list to find a display-id associated with that address-spec and deletes the address-spec. If MDT does not find the address-spec on the current display list, then an error is reported and the command is rejected.

12.2.3 Restriction

Either a display-id or an address-spec can be given, but not both.

12.3 THE LIST-DISPLAY-COMMAND

The list-display-command is used to print the address-specs that are currently on the display list.

To examine the contents of the display list, the programmer types the following command:

```
MDT>?D
```

MDT responds by typing the display list in the format shown in the following example:

```
DISPLAY LISTING
  NUMBER        ADDRESS    FIELD
     0           $0        <15:0>
     1          #21210     <7:0>
     3           6430      <22:20>
```

This listing indicates that three displays are current, with display-ids 0, 1, and 3.

12.3.1 Syntax

list-display-command	? D

12.3.2 Interpretation

In response to the list-display-command, MDT types the display list, giving the display-id and address-spec for each active display in the following format:

```
DISPLAY LISTING
  NUMBER        ADDRESS    FIELD
  display-id    address    bit-range
```

If a display-id is not currently in use, it is not printed as part of the display listing.

CHAPTER 13

CONTROL COMMANDS

Three control commands are provided in MDT. These commands can be used to start the execution of the program to be tested in the debugging environment, to reload the Writable Control Store, and to reset the state of MDT by deleting all breakpoints and displays.

The control commands are given in the following syntax:

control-command	{ go-command load-command reset-command }

The following sections describe each of the control commands.

13.1 THE GO-COMMAND

The go-command is used to start the execution of the program being tested. The address part of the go-command informs MDT where to start execution. For example:

```
MDT>G#46100
```

This go-command instructs MDT to begin execution at the main memory address 46100. If the relocation register $RL0 contains the base address 46000, then the following go-command also starts execution at main memory address 46000:

```
MDT>G#100,0
```

The go-command can be used to change the control sequence of the execution after a breakpoint. For example, suppose that the program execution halts at a breakpoint on one branch of a target assignment construct and that the programmer wants to test out another branch. He can enter a go-command after the breakpoint as indicated in the following sequence:

```
MDT>B6420
MDT>B6430
MDT>G#46100
 BREAKPOINT NUMBER 0 AT ADDRESS 6420
MDT>P
 BREAKPOINT NUMBER 0 AT ADDRESS 6420
MDT>G6430
```

13.1.1 Syntax

go-command	G transfer-address
transfer-address	{ # macro-address / micro-address } { , relocation-register } 1 / 0

13.1.2 Interpretation

MDT interprets a go-command in the following way:

MDT computes the transfer address. If a relocation-register is present, the contents of that register are added to the address. If the address begins with the character "#", then MDT interprets it as a main memory address; otherwise, MDT interprets the address as a micro memory address.

MDT restores the state of the machine from the microstate table. If the transfer address is to main memory, then the PDP-11 registers are restored. If the transfer address is to micro memory, then the datapath registers, as well as the PDP-11 registers are restored from the microstate table.

MDT starts the execution of the program under test at the transfer address.

13.1.3 Restrictions

The transfer address in the go-command must not be a micro address unless either a micro breakpoint has occurred or the programmer has manually set up the datapath registers in the microstate table by a sequence of open-bits commands.

A relocation-register must not be specified with a micro address.

13.2 THE LOAD COMMAND

The load command is used to restore the Writable Control Store to its initial loaded state. As part of its initialization sequence, MDT copies the contents of the Writable Control Store into an unused portion of main memory. If, after the program has been executing for a time, the programmer wants to restore the contents of the Writable Control Store, he uses the load-command. In response to this command, MDT restores the saved copy of the contents of the Writable Control Store.

This command is useful in the case in which the contents of the Writable Control Store has been altered either intentionally or accidentally while the contents of main memory have not been disturbed. An example of such a case occurs when the programmer has been making modifications to the microprogram in the Writable Control Store by the use of open-commands.

As a simple example, consider the case in which the programmer begins the debugging session by changing the contents of the ALU field of locations 6200, 6201, and 6203. He mistakenly opens 6000 and makes his modifications, as follows:

```
MDT>06000<ALU>
6000<47:44>  10  11  (lf)
6001<47:44>   7   6  (lf)
6002<47:44>  10  11
```

At this point, he realizes his mistake. He can either restore the contents of 6000-6002 by the use of the open-command or he can reload the Writable Control Store to restore it to its initial state. In this case, since no other changes were made to the Writable Control Store, the use of the load-command is clearly simpler. He continues by typing the load-command and then entering the correct changes, as follows:

```
MDT>L
MDT>06200<ALU>
6200<47:44>   2  11  (lf)
6201<47:44>   3   6  (lf)
6202<47:44>   1  11
```

13.2.1 Syntax

load-command	L

13.2.2 Interpretation

MDT interprets a load-command in the following way:

The saved copy of the Writable Control Store, which MDT copied into main memory as part of its initialization sequence, is restored to the Writable Control Store.

13.3 THE RESET COMMAND

The reset-command is used to reset the MDT tables. A reset-command directs MDT to remove all breakpoints and display-items from its tables. The reset command consists of the reset mnemonic, as indicated in the following command-line:

```
MDT>R
```

The above line is equivalent to the following two command-lines:

```
MDT>-B
MDT>-D
```

The reset-command does not supply any additional functionality but is provided in the MDT command language as a convenience.

13.3.1 Syntax

reset-command	R

13.3.2 Interpretation

MDT interprets a reset-command in the following way:

All breakpoints are removed from the program under test and all display-items are removed from the display list.

PART V

MICROPROGRAMMING TOOLS USER'S GUIDE

Contents

CHAPTER 14

USING THE ASSEMBLER

This section describes using the assembler. First, the two methods of assembling a microprogram are considered; then the input to the assembler is described; next, the assembler output is given; and finally, an example of the use of the assembler is presented.

14.1 THE INDIRECT FILE METHOD

The indirect file method of assembling a microprogram assumes that the programmer has written the program entirely in terms of the 11/60 predefinitions. Further, it assumes an object module file and, optionally, a listing file are wanted as a result of the assembly. As an example of the use of the indirect file method, consider the following interaction:

```
>@MIC
>; MIC.CMD \ASSEMBLE WCS MICROPROGRAM
>;
>* ENTER MICROPROGRAM SOURCE FILE SPECIFICATION [S]: LNKLST
>* LIST? [Y/N]:Y
>MIC LNKLST,LNKLST=PREDEF,DSPTCH,LNKLST

VERSION 1= 12-AUG-77


ERRORS DETECTED:      0
NUMBER OF LINES PROCESSED:      2745
```

The system requests the programmer to enter the name of the source file that contains the action-part of the microprogram. The programmer responds with the name of the file, in this case LNKLST. Then, the system asks if the programmer wants a listing file. He responds Y (yes) and the assembly is initiated. The assembly produces the object module file LNKLST.OBJ and the listing file LNKLST.LST. At the end of the assembly, a summary is printed. In this case, the summary shows that no errors were encountered in the assembly and that the number of lines processed was 2745. The lines of the predefinitions and dispatch files are counted in this summary although the contents of these files is not reproduced in the list file, due to the fact that these files contain .NLIST keyword to suppress listing.

14.1.1 Syntax

in-direct-method	>@MIC >* ENTER MICROPROGRAM SOURCE FILE SPECIFICATION[S]: in-spec >* LIST? [Y/N]: { Y / N }
in-spec	{ in-file } n , 1

The text typed by the user is underlined in the above syntax.

14.1.2 Interpretation

The in-spec given in response to the system question "ENTER MICROPROGRAM SOURCE FILE SPECIFICATION[S]:" is assumed to contain the action-part written in terms of the 11/60 predefinitions.

The response given to the question "LIST? [Y/N]" determines whether or not a listing file is created.

An assembly command line is constructed using the name of the input file and the names of the files supplied by Digital.

If the assembler is not installed, the actions necessary to install it are taken.

14.1.3 Restrictions

The maximum number of characters that can be given in an in-spec is 16.

14.1.4 Default

If the extension is omitted for a file-spec in the in-spec, the default extension .MIC is assumed.

14.2 THE DIRECT METHOD

The direct method of assembling a microprogram allows the user to specify the files that make up the program in a general way. He can, for example, extend or replace the 11/60 predefinitions.

Suppose, for example, that the programmer wants to add a few of his own definitions to the 11/60 predefinitions. To do this, he invokes the assembler in the following way for the files LNKLST and MYDEF:

```
>MIC LNKLST,LNKLST=PREDEF,MYDEF,DSPTCH,LNKLST
```

The user-definitions are included in the file MYDEF.

If the MICRO-11/60 assembler is installed, it is invoked by typing its three letter abbreviation, MIC, followed by the assembly-command-line at the command level in response to an operating system prompt. If it is not installed, the RUN command must be used. The assembly-command-line gives the output and input files in the RSX-11M standard notation. The output files are given on the left-hand-side and the input files are given on the right-hand-side of the '=' character in the assembly command line.

14.2.1 Syntax

assembly	>MIC assembly-command-line
assembly-command-line	output-spec = input-spec
output-spec	{ object-file }$_{0}^{1}$ { , list-file }$_{0}^{1}$
input-spec	{ input-file }$_{1}^{n}$
object-file list-file input-file	file-spec { / switch }$_{0}^{2}$
file-spec	{ dev: }$_{0}^{1}$ { [ppn] }$_{0}^{1}$ file-name { .ext }$_{0}^{1}$
switch	{ MX | BT }

The text typed by the user is underlined in the syntax for assembly.

14.2.2 Interpretation

An assembly interaction consists of the invocation of the assembler, followed by an assembly-command-line.

The assembly-command-line informs the assembler of the names of the files to be used for output and input. The position of the file within the assembly-command-line indicates its intended use.

Both the output files are optional. If both output files are omitted, then the assembler validity checks the input and reports any errors on the terminal. If more than one input file is given, the assembler reads the microprogram from the specified input files in the order given in the assembly-command-line. The first input file is read and, if an .END is not encountered before the end of that file is reached, then that file is closed and the second file opened. Processing continues with the second input file, again until either an .END, signifying the end of the microprogram, or an end-of-file, indicating the end of input on that file, is read. Processing continues in this way, moving from file to file, until an .END is encountered. The .END determines the end of the microprogram. If it is read before all the files specified in the assembly-command-line are processed, then the information after the .END is discarded.

The assembler produces the object module on the object-file and that file can be subsequently given as an input file for the microprogram loader MLD . If an object-file is not specified, no object module is produced. The format of the object module is described in Section 8.2.1

The assembler produces the output listing on the list-file. If a list-file is not specified, then no output listing is produced and any errors detected in the assembly are reported at the terminal. The output listing is described in Section 14.4.

The switches can be given following any file in the assembly-command-line. The switch MX directs the assembler to include macro expansions for all input files in the output listing and the switch BT directs the assembler to add a bit map at the end of the output listing. The result of adding the MX switch is described in Section 14.4.6 and the result of the BT switch in Section 14.4.7

14.2.3 Defaults

If the file extension (.ext) is omitted in a file-spec, the following extensions are assumed:

File	Default Extension
input	.MIC
object	.OBJ
list	.LST

14.2.4 Guidelines

Omitting the object-file increases the speed of the assembly. Sometimes several assemblies of a microprogram are necessary before the microprogram is ready to be loaded and tested. In early assemblies, therefore, omitting the object module saves time.

More often, however, the microprogrammer feels that the results of the assembly may be useful for testing and includes the object-file. An assembly can produce a number of errors and still yield an object module that can be executed and, from whose execution, the microprogrammer can obtain information about the validity of the microprogram.

In the absence of a strong conviction about the usefulness of either of the output files, the microprogrammer should include both in the assembly-command-line. The time saved by omitting the object-file is minor compared to the time required to rerun the assembly simply to obtain an object module and when an object-file is produced, a list-file should be produced. If an object-file does not have an associated list-file, then troublesome questions about its contents are apt to arise.

14.3 ASSEMBLER INPUT

The input to the assembler is a microprogram. The microprogram consists of a sequence of lines, written in MICRO-11/60 source and conforming to the syntactic rules of that language. The input can be prepared using any available editor.

This section discusses preparing the input, suggests some formatting rules, and gives an example of assembler input.

14.3.1 Preparing The Input

The first step in preparing the input is writing the microprogram. To write a microprogram, the programmer must be familiar with the internal details of the 11/60 processor, as described in the "11/60 Microprogramming Specification" and with the 11/60 predefinitions, given in Appendix B of this manual.

Representing an algorithm as a microprogram often involves rethinking the logic of the algorithm. The example given in Appendix D illustrates this process, showing first three macro programs for manipulating a linked list and then giving the restructured algorithms for the corresponding microprograms.

Once the logic of an algorithm is determined, the microprogram is written using the 11/60 predefinition language, which defines the fields of the 11/60 microword and provides a macro language that is oriented toward the logical operations performed in a microprogram.

Then, the microprogram source is entered using any available editor. From the assembler's point of view, the microprogram consists of a sequence of lines beginning at the start of the first input file and continuing until an .END is encountered. Within those limits, the microprogram must have the expected structure. The first part must give the definitional information and the second part must give the actions to be performed when the microprogram is executed.

The assembler detects and reports errors, as described in Section 14.4.5. In response to these errors, the microprogrammer edits the input to obtain a valid microprogram. This process continues until either no errors are present or until the microprogrammer is convinced that the messages produced do not affect the validity of the microprogram.

14.3.2 Formatting The Microprogram

Using a standard formatting scheme increases the readability of the microprogram. A standard format for 11/60 microprograms has been developed at DIGITAL and is given here for the information of Writable Control Store users.

If a microprogram has any definitions, it begins with the .TITLE and .IDENT lines and continues with field definitions, followed by macro-definitions, as follows:

```
.TITLE   title
.IDENT   /version/
.FIELD   field-name ::= field-spec
    field-value-name ::= value
    ...
...
.MACRO   macro-name ::=
    instruction-part,...
    ...
```

The action-part of a microprogram consists of a sequence of microinstructions, as follows:

```
microinstruction
...
.END
```

The standard format for a microinstruction is as follows:

```
! comment
...

! comment

address:

label:

     time-state,  instruction-part, ... ! comment
                  instruction-part, ... ! comment
     ...

     NEXT,        instruction-part, ... ! comment
                  J/next-address
```

The rules for formatting a microinstruction are summarized as follows:

1. Precede the microinstruction by any general comments.

2. If the microinstruction has an explicit address, give that address at the left-margin and do not include any other information on that line.

3. If the microinstruction has a label, give that label at the left margin and do not include any other information on that line.

4. Begin the microinstruction with the first time-state. Time-states are given at the first tab position. (column 9).

5. Include as many instruction-parts, separated by commas, as will fit in the columns starting at the second tab (column 17) and continuing to column 38.

6. Place any line-specific comments at the fifth tab (column 41). In order to maintain the microprogram in a bindable form (8 1/2 x 11), do not continue the comment past column 70.

7. If more instruction-parts are specified for a time-state than can fit on a single line, continue at the second tab (column 17) of the next line through column 38.

8. Give the NEXT time-state as the last time-state of the microinstruction and conclude the instruction-part of the NEXT time-state with a branch to the next-address.

9. Separate each microinstruction from the remainder of the microprogram by one or more blank lines.

All the microprograms in this manual are written in the standard format.

14.3.3 A Sample Input Listing

The input listing for the microprogram given in Chapter 4 of the first part of this manual is reproduced here. The register exchange microprogram is chosen because its size, although untypically small for a microprogram, is convenient for inclusion in a manual. A more typical microprogram is given in Appendix D.

```
.TITLE   REGEX
.IDENT   /R1V1/
! REGISTER EXCHANGE PROGRAM


.CASE 0 OF DISPCH
EXCHANGE:
         P2-T,  SR_A,R3-A,                ! SAVE R3
         NEXT,  J/EXCH2;

EXCH2:
         P2-T,  D_A,R2-A,                 ! MOVE R2 TO R3
         P3,    WR(AB,L,B),R3-B,
         NEXT,  J/EXCH3;

EXCH3:
         P2-T,  D_SR,                     ! MOVE SAVED R3 TO R2
         P3,    WR(AB,L,B),R2-B,
         NEXT,  BUT(SUBRB),PAGE(0),
                J/BRA05;
.END
```

The output listing for this sample is given in Section 14.4.9.

14.4 THE OUTPUT LISTING

The output listing of a microprogram corresponds to the input listing, except that the assembler prints some additional information, namely:

- A table of contents, formed by listing each .TOC line with its assigned line number at the beginning of the output listing.
- A line number at the beginning of each line.
- Page headings at the top of each page.
- Microword lines, giving the address and bits for each microinstruction in the microprogram.
- Error messages, if any errors are detected.
- Macro expansions, if requested by the MX switch.
- A bit map, if requested by the BT switch.
- An error summary.

A brief description of each of the above items is given in the following sections.

14.4.1 The Table Of Contents

The table of contents is constructed by collecting the .TOC lines to the beginning of the listing. Judicial placement of .TOC lines within the listing results in a useful table of contents, by which the microprogrammer can quickly reference any logical section of the microprogram.

As the size of a microprogram increases, the value of the table of contents increases. However, the assembler always prints a table of contents page, even when the microprogram does not contain any .TOC lines and the table of contents is, accordingly, empty. Therefore, including some .TOC lines in even the shortest microprogram is advisable.

.TOC lines and the construction of the table of contents are described in detail in Chapter 4, "Program Structure". A good example of the use of .TOC lines to produce a comprehensive table of contents can be found in Appendix B, in which the 11/60 predefinitions are given.

14.4.2 Line Numbers

Each input line is numbered, by the assembler. The line number is a four digit decimal number, which starts at 0001 and continues, in increments of 0001, through 9999. If a microprogram contains more than 9999 lines, then the string '****' is used instead of a line number for every line after 9999.

Since blank lines and comments are assigned line numbers, it is not unusual for a small microprogram to occupy several thousand lines. However, the line limit of 9999 is seldom exceeded. If it is exceeded, the resulting assembly is still valid and the only inconvenience is that the table of contents does not locate the position of .TOC lines that occur after the 9999th line.

14.4.3 Page Headings

The assembler divides the output listing into pages. Each page contains a heading line and 54 lines of the microprogram. The page heading gives the following items of information:

- The program title, as derived from the first six characters of the last .TITLE line.
- The name and version number of the MICRO-11/60 assembler used in assembling the microprogram.
- The date and time of the assembly.
- The page number.

If a .TITLE line is not given in the microprogram, then the title part of the heading is left blank.

An example of a heading line is:

```
LNKLST   MICRO   V00A-1   11:20:02   10-SEP-77          PAGE 2
```

The heading line, as part of an output listing, is given in Section 14.4.9.

14.4.4 The Microword Line

The microword line contains the address and bits of the microword, in the following format:

```
   nnnn  b bbbbbbbb bbbbbbbb bbbbbbbb bbbbbbbb bbbbbbbb bbbbbbbb
           |        |        |        |        |        |      |
           47       39       31       23       15       7      0
```

The address, nnnn, identifies the location to which the microword is assigned. The bits, b, identify the value of the bits in the microword. The bits are displayed as shown above, in groups of eight. The first single bit is not part of the microword and should be ignored. It is present because of a technique used in the predefinition language.

The microword line is illustrated in Section 14.4.9.

14.4.5 Error Messages

If an error is detected in a microprogram line, then an error message is printed by the assembler following that line. Error messages are easy to find within the listing because, instead of a line number, error messages begin with the string '****', followed by the error number. The error message also contains a short description of the error and, if possible, a piece of the input line to show the point at which the error was detected.

If the assembler detects an error in a line, then the information on that line is not fully interpreted in the assembly process. The fact that the information is not interpreted sometimes causes additional errors later. Usually, when the first error is corrected, the other errors disappear. For example, suppose the microprogrammer makes the syntactic error of using a hyphen rather than a colon in a field definition as follows:

```
.FIELD ALPHA ::= <44-40>
```

The assembler detects that error:

```
2345  .FIELD ALPHA ::= <44-40>
****39      SYNTAX ERROR
```

Because the assembler rejects the definition of ALPHA, any uses of the field ALPHA within subsequent microinstructions also produce error messages as follows:

```
4567        ALPHA/1
****24    MICROINSTRUCTION ILLEGAL        ALPHA
```

Appendix E lists all of the assembler error messages.

14.4.6 Macro Expansions

If the microprogrammer specifies the MX switch in the assembly-command-line, then the macros used in each microinstruction are expanded and printed in the output listing.

The expansion of each macro is shown on a separate line. The expansion line begins with the string '+ ' instead of a line number.

As an example of a listing that includes macro expansions, consider the following output listing excerpt produced from the assembly of the matrix addition example given in Section 7.2.

```
2129  6200:
2130  MATADD:
2131    P1,    CLK-BA,PC-A,                        !INITIATE MEM(PC) READ:
 +             N/O
 +             CLKBA/YES
 +             AEN/ASPLO,ASEL/R07,RIF/R07
2132    P2-T,  A-PLUS-B,CSPB(TWO),                 !INCREMENT PC.
 +             WHEN/AT-P2-T
 +             ALU/A-PLUS-B
 +             BEN/BASCON,BSEL/TWO
2133    P3,    WR(AB,L,A),DATI,
 +             N/O
 +             MOD/CLKSP,WRSP/AB,HILO/L,WRSEL/A
 +             BEGIN/YES,SELECT/BUS,BUSCODE/DATI
2134    NEXT,  J/MAT1;
 +             N/O

      6200    0 10011111 10011110 00000101 01100001 10110000 00000000
```

The first macro in the first microinstruction line, 'P1', expands to the string 'N/O'. The second macro 'CLK-BA' expands to the string 'CLKBA/YES'. The third macro 'PC-A' expands to the string 'AEN/ASPLO,ASEL/R07,RIF/R07'. The first macro in the second microinstruction line, 'P2-T', expands to the string 'WHEN/AT-P2-T', and so on.

14.4.7 The Bit Map

If the microprogrammer specifies the BT switch on the assembly-command-line, a bit map is produced at the end of the output listing. The bit map indicates the addresses that are used by the assembly.

The bit map consists of a matrix of binary digits. The digit 1 indicates that an address is used and the digit 0 indicates that an address is not used.

As an example of a bit map, consider the following:

```
6576  1 1 1 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

6636  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

6676  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

6736  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

6776  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

7036  0 0
```

The bit map displays all the addresses within the specified bounds. The above bit map was produced by a microprogram that contains five microinstructions and the bounds 6576:7037. As can be seen from the map, the assembler used locations 6576, 6577, 6600, 6605, and 6607 for that program.

14.4.8 The Summary

At the end of the input listing, MICRO-11/60 summarizes all the errors detected as a result of the assembly and gives the total number of lines processed. The number of lines includes the lines from the predefinitions and dispatch-files, if those files are included as part of the assembly.

14.4.9 A Sample Output Listing

The output listing for the sample input program given in Section 14.3.3 is shown below.

```
            MICRO     V00A-1  10:17:08  06-JUL-77           PAGE 1


TABLE OF CONTENTS




REGEX       MICRO     V00A-1  10:17:08  06-JUL-77           PAGE 2


    1  .TITLE  REGEX
    2  .IDENT  /R1V1/
    3  ! REGISTER EXCHANGE PROGRAM
    4  .CASE 0 OF DISPCH
    5  EXCHANGE:
    6          P2-T,   SR_A,R3-A,              ! SAVE R3
    7          NEXT,   J/EXCH2;

       6030  0 11110000 10011010 00001000 00000000 00110000 10000000

    8
    9  EXCH2:
   10          P2-T,   D_A,R2-A,               ! MOVE R2 TO R3
   11          P3,     WR(AB,L,B),R3-B,
   12          NEXT,   J/EXCH3;

       6200  0 11110011 10001010 00010000 00000011 10110000 10000001

   13
   14  EXCH3:
   15          P2-T,   D_SR,                   ! MOVE SAVED R3 TO R2
   16          P3,     WR(AB,L,B),R2-B,
   17          NEXT,   BUT(SUBRB),PAGE(0),
   18                  J/BRA05;

       6201  0 11110010 00001010 00010000 00000011 10111000 0011

   19 .END

MIC -- ERRORS DETECTED:    0
MIC -- NUMBER OF LINES PROCESSED:    2382
```

CHAPTER 15

USING THE MICROPROGRAM LOADER

Before a program that uses a microprogram can be executed, the 11/60 Writable Control Store must be loaded and enabled. Loading and enabling requires the use of the Microprogram Loader, MLD, and two small stand-alone programs, MSTART and MSTOP. The program MSTART enables the Writable Control Store and MSTOP disables it.

All three of these programs are intended to be privileged programs under RSX-11M to be accessed only by users with that status. The execution of a microprogram in the WCS essentially modifies the machine dynamically, and, therefore, the use of the WCS should be restricted to the programmer who is aware of his responsibility to the other programmers on the system. Until the error-free operation of a microprogram is assured, any testing of that microprogram should be done in a single-user (stand-alone) mode.

To run a program that contains both macro and micro code, several steps are necessary. First, the macro code must be linked and loaded into main memory. Then, the micro code must be loaded into micro memory (the WCS). Finally, the WCS must be enabled. An indirect command file, @MLD, is provided to assist the programmer in loading and enabling the WCS.

The steps necessary to link and load macro memory are not discussed here. Information on loading macro memory can be found in the RSX-11M Task Builder Reference Manual (DEC-11-OMTBA-A-D).

The following sections describe the two methods for invoking the loader and the enabling programs. Then, the inputs and outputs of the loader are discussed. Finally, an example of the use of the loader is given.

15.1 THE INDIRECT FILE METHOD

In the indirect file method of loading the Writable Control Store, the user invokes the loader by typing the loader indirect command file, @MLD, at command level. The system then asks if the WCS is to be enabled and requests the name of the file that contains microprograms to be loaded. An example of an interaction is:

```
>@MLD
>; MLD.CMD\LOAD WCS
>;
>* ENABLE WCS?[Y/N]:Y
>* ENTER MICROPROGRAM OBJECT FILE SPECIFICATION[S]:LNKLST
>MLD WCS=MICPAK,LNKLST
>RUN MSTART                    !MSTOP SHOULD BE RUN WHEN FINISHED
>@<EOF>
```

In response to the system's request for information, the user types 'Y' and the file LNKLST, which contains an object module for that program.

15.1.1 Syntax

in direct-load	>@MLD * ENABLE WCS?[Y/N]: { Y / N } * ENTER MICROPROGRAM OBJECT FILE SPECIFICATION[S]: obj-spec
obj-spec	{ file-spec }n,0

The text typed by the user is underlined in the syntax for load.

15.1.2 Interpretation

A load interaction consists of the invocation of the loader, followed by an indication whether or not to enable the WCS, by the specification of the file for the microprograms to be loaded.

If the user answers 'Y' to the question 'ENABLE WCS?', the program MSTART is executed to enable the WCS after the microprograms are loaded. If the user types 'N' or simply carriage return, then MSTART is not executed. Any other response is an error.

The resident section supplied by DIGITAL, MICPAK, is loaded followed by the specified microprogram object files.

The loader does not report an error message if, in the course of loading a set of object modules, a word in the WCS is reloaded. If the programmer specifies more than one object module, he must be responsible for the address compatibility of the modules. Object modules can be assembled into different parts of the WCS by the use of the .BOUNDS keyword, as described in "Partitioning the WCS" in Appendix C.

The fact that the loader allows addresses to be reloaded can be convenient. It allows the user to patch an existing object module.

15.1.3 Restrictions

The maximum number of characters that can be given in an obj-spec that is part of an @MLD sequence is 16.

15.1.4 Defaults

If any of the file-specs given in response to the request 'ENTER MICROPROGRAM OBJECT FILE SPECIFICATION[S]:' does not contain an extension, the extension '.OBJ' is assumed.

If no file-spec is given, only the resident section is loaded into the WCS.

15.2 THE DIRECT METHOD

The indirect command file @MLD is an aid supplied for the programmer. He can invoke the loader directly by typing MLD, followed by the loader command line, in response to the operating system prompt, as follows:

```
>MLD WCS=MICPAK,LNKLST
```

The loader command line gives the programs to be loaded, which, in this case, are MICPAK, the resident section supplied by DIGITAL, and LNKLST, the user's object module file.

If the user wants to load more than one object module, he can specify the object modules when invoking MLD directly as follows:

```
>MLD WCS=MICPAK,LNKLST,MATPAK
```

Or, if he wants to supply his own resident section, he can do so, as follows:

```
>MLD WCS=MYRES,MYMIC
```

15.2.1 Syntax

direct-load	>MLD { WCS }1 0 = { MICPAK , }1 0 { file-spec }n 1

15.2.2 Interpretation

The loader loads the specified files starting with the first and continuing through the last file into the WCS.

Following the use of the loader, the programmer should run the program MSTART to enable the WCS.

15.2.3 Defaults

The default file extension is .OBJ for files specified in a call on the loader.

15.3 ENABLING THE WCS

The execution of MSTART enables the WCS. The user can then execute a program in the WCS.

When the user is finished with the program that uses the WCS, he should disable the WCS by running the disabling program MSTOP, as follows:

```
>RUN MSTOP
```

MSTART sets a bit in the WHAMI register that permits the use of the WCS; and, MSTOP resets that bit. Therefore, the successful execution of these programs is usually assumed. However, if the user wants to verify that the programs executed properly, he reads the console lights. The successful completion of these programs is indicated by the following pattern in the console lights:

Program	Console Lights
MSTART	000222
MSTOP	000333

Once the WCS is enabled, any XFC instruction executed, whether intentionally or intentionally, causes the execution of microcode. Therefore, the WCS should be disabled as soon as the programmer is finished executing his microprogram.

15.4 LOADER INPUT

The loader input consists of the object module for the resident section and the object modules that make up the microprogram to be executed. The object module is described and illustrated in Section 8.2.1.

When more than one object module is to be loaded into the WCS, the user must ensure that the addresses occupied by the different modules do not conflict. He can cause the object modules to occupy different address spaces by the use of the .BOUNDS keyword, as described in the section on "Partitioning the WCS" in Appendix C.

15.5 LOADER OUTPUT

After executing, the loader prints a message indicating whether or not its execution was successfully completed.

When the loading process is successful, MLD prints the message:

WCS LOAD COMPLETED

The user can assume, if that message is printed and no warnings are issued, that the WCS is properly loaded. He can then proceed to enable the WCS and run his program.

When the loading process is unsuccessful, MLD prints the message:

ABNORMAL PROGRAM TERMINATON

This message is usually preceded by one or more error messages, which indicate the reasons for the failure of the loading process.

15.5.1 Error Messages

The microprogrammer who uses the predefinitions and dispatch files as part of his assembly is likely to encounter only two errors in loading the resulting object module.

BAD RECORD IN OBJECT FILE

The loader reports this error if the format of the object module is not correct. In response to this message, the user should check all the input files to ensure that they contain valid object modules.

The other error is:

WRITE TO WCS FAILED AT MICROINSTRUCTION ADDRESS: micro-address

The loader writes an instruction into a WCS location and then reads back the contents of that location to compare it with the value written. If the value read does agree with the value written, then the above error is reported. In response to this message, the microprogrammer can try again. However, if he receives the same message, he should assume that a hardware problem is likely to exist.

If the microprogrammer changes the bounds for the assembler, as described in connection with the .BOUNDS keyword at the beginning of Appendix B, he may encounter the following error message:

MLD -- WARNING: INVALID 11/60 MICROINSTRUCTION ADDRESS: address

Moreover, this message is issued if the loader receives an address in the range 0000:5777. The loader loads the first 16 bits in the local store address specified and then reports the message given above.

15.6 EXAMPLE

As an example of the procedure used to load and enable the Writable Control Store prior to execution, consider the case in which the program to be run consists of two FORTRAN programs, ANALYZR and PARSE, a MACRO 11 program, INTRFC, and two microprograms LNKLST and MATPAK.

The user begins by building the task for the main memory programs. On a system with OTS in SYSLIB, the task build is:

```
>TKB AWCSYS=ANALYZR,PARSE,INTRFC
```

Then, he creates a microprogram file that contains the object modules for LNKLST and MATPAK loads the WCS, as follows:

```
>@MLD
>; MLD.CMD\LOAD WCS
>;
>* ENABLE WCS?[Y/N]:Y
>* ENTER MICROPROGRAM OBJECT FILE SPECIFICATION[S]:LNKLST,MATPAK
>MLD WCS=MICPAK,LNKLST,MATPAK
>RUN MSTART                !MSTOP SHOULD BE RUN WHEN FINISHED
>@<EOF>
```

Then, he runs his program, as follows:

```
>RUN AWCSYS
```

After the execution of the task is complete, the user disables the WCS by running MSTOP.

.

CHAPTER 16

USING THE DEBUGGER

The MicroDebugging Tool MDT operates as a stand-alone tool for debugging programs that use the Writable Control Store of the 11/60. To use MDT, he invokes the MDT indirect command file. This command file shuts down the operating system and interacts with the programmer to run the program under test in a single-user mode.

16.1 RUNNING MDT

To initiate MDT from RSX-11M, the user enters the debugger indirect command file, @MDT, at command level. The system then requests the name of the microprogram to be loaded into the WCS, runs the RSX-11M shutdown program, and brings up MDT. An example of an interaction is:

```
>@MDT
>; MDT.CMD \LOAD WCS AND START MDT (FROM PRIVILEGED UIC)
>;
>; S H U T S    R S X    D O W N
>;
>* ENTER MICROPROGRAM OBJECT FILE SPECIFICATION [S]: MATADD
>MLD WCS=MDTMIC,MATADD
>RUN MSTART    !ENABLES WCS (SHOULD RUN MSTOP AS SOON AS RSX BACK UP)
>;             (SUGGEST PUTTING MSTOP IN STARTUP.CMD FILE)
>RUN SHUTUP

RSX11M SHUT DOWN PROGRAM

>
ENTER MINUTES TO WAIT BEFORE SHUTDOWN:
0
>
ENTER MINUTES BETWEEN MESSAGES:
0
>
ALL FURTHER LOGINS ARE DISABLED

01-SEP-77 11:53 PLEASE FINISH UP, 0 MINUTES BEFORE SHUTDOWN
; TYPE "RES AT." WHEN SHUTDOWN COMPLETED
>
AT. -- PAUSING.  TO CONTINUE TYPE "RES ...AT."
                                                DMO DB0:
>
>RES AT.
AT. -- CONTINUING
>
>
>* HAVE YOU PROTECTED EVERYTHING YOU WANT PROTECTED? [Y/N]:Y
>BOOT MDT     ! T H I S   T A K E S   R S X   D O W N
MICRO DEBUGGING TOOL. VERSION #1.0

MDT>
```

In response to the system's request for information, the user types the name of the microprogram 'MATADD', the shutdown parameters '0' and '0', 'RES AT', and the answer 'Y' to the question about write protection.

The general form of the interaction is given in the following syntax. In this syntax, the commentary printed by the system is omitted.

16.1.1 Syntax

```
mdt-
  call  >@MDT
        >* ENTER MICROPROGRAM OBJECT FILE SPECIFICATION[S]:obj-spec
        ENTER MINUTES TO WAIT BEFORE SHUTDOWN:
        minutes
        ENTER MINUTES BETWEEN MESSAGES:
        minutes
        AT: -- PAUSING. TO CONTINUE TYPE 'RES ...AT.'
        >RES AT.

        >* HAVE YOU PROTECTED EVERYTHING YOU WANT PROTECTED? { Y } 1
                                                            { N } 0
        MDT>
```

The text typed by the user is underlined in the above syntax.

16.1.2 Interpretation

The resident section for MDT and the obj-spec given by the user in response to the request 'ENTER MICROPROGRAM OBJECT FILE SPECIFICATION[S]:' are loaded into the WCS by MLD. The obj-spec can consist of one or more files, each of which contains one or more object modules. The object modules are loaded by MLD, starting with the first, and continuing until the end of the last object module is reached.

The minutes until shutdown and the minutes between messages given by the user are used in the shutdown procedure. If the system is operating in stand-alone mode, then a response of 0 minutes in both cases is indicated. If the system is operating in multi-user mode, then an appropriate delay before shutdown should be given.

When the shutdown is complete, the system types the message: 'AT. -- PAUSING. TO CONTINUE TYPE 'RES ...AT.'. The response 'RES AT.' directs the system to continue. The user must not type 'RES AT.' until the shutdown is complete. In the example given above, the shutdown is complete when the system types 'DMO DB0:'

Before bringing up MDT, the system asks if all the devices are properly protected. At this point, if the user needs to write protect some devices, he can type 'N' or not respond. If he types 'N', the system repeats the question. When the devices are protected, he types 'Y' and MDT is brought up. MDT identifies itself and prompts for the first command in the debug session.

16.1.3 Restrictions

The maximum number of characters that can be given in an obj-spec that is part of an @MLD sequence is 16.

16.2 THE DEBUG SESSION

Following the prompt from MDT, the debug session begins. The user can enter any number of MDT commands to set breakpoints, to examine and change locations in main or micro memory, and to execute the program under test. The commands that can be given in response to the MDT prompt are described in Chapters 9 through 13 of this manual:

16.2.1 Interrupting Program Execution

The programmer can interrupt the execution of either MDT or the program under test by typing two control-C characters.

If the double control-C is typed during the input of a command line, all the input entered is erased. The programmer can use the double control-C to delete an incorrect command line. For example, suppose the programmer forgets the '#' character in the transfer address of a go-command. If he notices the error before hitting the carriage return that terminates the command, he can type two control-C (^C) characters and start the command again, as follows:

```
MDT>G6612^C^C
G#6612
```

If the double control-C is typed during the execution of the program under test, then the execution of the program is aborted and control is returned to MDT. MDT prompts for another command. The state of the machine is not saved when the program execution is interrupted and, therefore, the contents of the registers are not meaningful. Program execution can be resumed after interruption by the use of a go-command to a main memory address.

An exception to this procedure for interrupting execution occurs when the microprogram is in an infinite loop and not checking for service. In this case, only an INIT signal from the console can interrupt execution. The INIT signal is produced by simultaneously pressing the HALT and START buttons.

16.2.2 Restarting The Debugger

In some cases, it may be necessary to restart the debugger. An entry point $DEBUG is provided for this purpose. If the programmer loads the address associated with this entry point into the switch register and starts the processor, then MDT is restarted with the state of the system as it was before the operation that caused the problem.

16.2.3 Terminating A Session

When the programmer is finished with the debugger, he brings up the operating system as if a system crash occurred as described in the RSX-11M Operator Procedures Manual.

However, since the use of MDT enables the WCS as part of its initialization procedure, the operating system restart should call the program MSTOP to disable the WCS. The inclusion of the call on MSTOP as part of the STARTUP command file is recommended. An example of a system restart with MSTOP as part of the STARTUP file is as follows:

```
MDT>

  RSX-11M V03 BL18   124K   MAPPED
>RED DB0:=SY0:
>MOU DB0:RSX11MBL18
>@[1,2]STARTUP
>RUN [1,54]MSTOP
>INS [1,54] PIP
>INS [1,54] CRF
>INS [1,54] EDI
>INS [1,54] TEC
>INS [1,54] TEC/TASK=...MAK
>INS [1,54] PRT/PAR=SPLPAR/CKP=NO
>INS [1,54] BIGTKB/PAR=GEN
>INS [1,54] BIGMAC/PAR=GEN
>INS [1,54] BRO
>INS [1,54] RMDEMO
>INS [1,54] SHUTUP
>INS [1,54] F4P/INC=5120.
>INS [1,54] HEL
>INS [1,54] BYE
>LOA DK:
>LOA LP:
>* PLEASE ENTER TIME AND DATE (HH:MM MM/DD/YY) [S]: 11:56 9/1/77
>TIM 11:56 9/1/77
>* PLEASE ENTER OPERATOR UIC AS #,# [S]: 2,100
>SET /UIC=[2,100]
>BRO @[2,1]ONTHEAIR,TXT

01-SEP-77 11:56
```

16.3 DEBUGGER ERRORS

MDT examines the command strings it receives from the programmer for validity. If MDT detects an error in a command, it prints a message, rejects the command, and prompts for another command. The programmer can then reenter the command with the appropriate syntax.

The error messages printed by MDT are given in Appendix E. Most messages are self-explanatory. For example, consider the following interaction:

```
MDT>O#46002
#46002<15:0>  32  O#47020
OPEN ROUTINE:  UNKNOWN TERMINATOR
MDT>O#47020
```

In the above interaction, the programmer forgets to type the carriage return to terminate the open-bits-command. MDT responds with the error message shown above. The programmer can then enter the command again.

The least specific MDT error message occurs in the following interaction:

```
MDT>D6016
MDT>D6017
MDT>6060
COMMAND PARSER:  SYNTAX ERROR
MDT>D6060
```

The programmer is entering a sequence of display-items. In the third line, he forgets the D command mnemonic. MDT cannot guess his intent and responds with the message shown above. The programmer, in response to this message, examines the preceeding line, finds his error, and reenters the command.

16.3.1 A Special Error

One error detected by MDT requires special attention. If, in typing a command to MDT, the programmer fills up the 64 character input buffer, MDT rings the bell each time a character is typed and rejects the additional characters. The only way to escape from this error condition is to type either a control-C or control-U character. These control characters delete all the information in the input buffer and reset it to an empty condition.

16.4 AN EXAMPLE

As an example of the use of MDT, in which a main memory program is used to call the microprogram, consider the debugging of the microprogram for matrix addition given in Section 7.2. As the first step in the debugging process, the programmer writes a simple macro program to call the microprogram, as follows:

```
            .TITLE    MATEST
MATA:       .BLKW     100
MATB:       .BLKW     100
START:      .WORD     076700
            .WORD     MATA
            .WORD     MATB
            .BYTE     4
            .BYTE     4
            .HALT
            .END      START
```

The programmer assembles the macro program and links the resulting object module with the MDT object modules, as follows:

```
>TKB MATEST=@MDTBLD,MATEST
```

The load map produced as a result of linking the object modules can be used to locate the object module MATEST in memory. Suppose it begins at 46000.

Next, the programmer invokes MDT by typing the indirect file @MDT.

```
>@MDT
>; MDT.CMD \LOAD WCS AND START MDT (FROM PRIVILEGED UIC)
>;
>; S H U T S   R S X   D O W N
>;
>* ENTER MICROPROGRAM OBJECT FILE SPECIFICATION[S]: MATADD
>SET /MAIN=WCS:7600:200:DEV
>INS [1,1]WCS
>INS MLD/TASK=...MLD
>MLD WCS=MDTMIC,MATADD
>RUN MSTART   !ENABLES WCS (SHOULD RUN MSTOP AS SOON AS RSX BACK UP)
>;            (SUGGEST PUTTING MSTOP IN STARTUP.CMD FILE)
>RUN SHUTUP

RSX11M SHUT DOWN PROGRAM

>
ENTER MINUTES TO WAIT BEFORE SHUTDOWN:
0
ENTER MINUTES BETWEEN MESSAGES:
0
ALL FURTHER LOGINS ARE DISABLED
>
```

```
01-SEP-77 11:56 PLEASE FINISH UP, 0 MINUTES BEFORE SHUTDOWN
: TYPE "RES AT." WHEN SHUTDOWN COMPLETED
>
AT. -- PAUSING. TO CONTINUE TYPE "RES ...AT."
                                                DMO DB0:
>
>RES AT.
AT. -- CONTINUING
>
>
>* HAVE YOU PROTECTED EVERYTHING YOU WANT PROTECTED? [Y/N]:Y
>BOOT MDT     ! T H I S   T A K E S   R S X   D O W N
MICRO DEBUGGING TOOL. VERSION #1.0

MDT>
```

Observe that this dialogue is different from the dialogue given at the beginning of this chapter in that it installs MLD and sets up the WCS I/O page common.

Next, the programmer uses MDT to set some data values into the blocks reserved in the main memory test program, as follows:

```
MDT>O#46000
#46000<15:0>  0  1  (lf)
#46002<15:0>  0  2  (lf)
#46004<15:0>  0  3  (lf)
#46006<15:0>  0  4
MDT>O#46100
#46200<15:0>  0  5  (lf)
#46202<15:0>  0  6  (lf)
#46204<15:0>  0  7  (lf)
#46206<15:0>  0  10
```

Then, he sets breakpoints at the beginning and end of the microprogram and starts the execution of the program, as follows:

```
MDT>B6200
MDT>B6224
MDT>G#47200
```

When the program execution halts at the breakpoint at the end of the program, the programmer can use the open-bits command to check the results of the matrix addition, as follows:

```
MDT>O#46000
#46000<15:0>  6   (lf)
#46002<15:0>  10  (lf)
#46004<15:0>  12  (lf)
#46006<15:0>  14
```

The programmer can then change the dimensions of the array and try again, as follows:

```
MDT>OB#46406
#46406<7:0>  4  6   (lf)
#46407<7:0>  4  6
```

APPENDIXES

Contents

APPENDIX A

SYNTACTIC SUMMARY OF SOURCE AND COMMAND LANGUAGES

This appendix contains a syntactic summary for the source and command languages used by each of the microprogramming tools. The MICRO-11/60 source is summarized first. Next, the MDT commands are listed. Then, the command language syntax is given for MICRO-11/60, MLD, and MDT.

The syntax is presented here in a concise format for quick reference. Some of the tutorial metasyntactic names used in the syntax within the manual are omitted so that the syntax can be given in a minimum amount of space.

All the syntactic terms used in the syntax sections of the manual and in this appendix are listed in the index. If the programmer wishes more information on a syntactic term, he can obtain the page in the manual that discusses that term by looking in the index.

A.1 MICRO-11/60 SOURCE SYNTAX

A.1.1 Processing-Unit

processing-unit	{ predefinitions-file user-machine-definitions } { .TITLE title-string .IDENT / ident-string / .RADIX radix .TOC toc-string field-definition macro-definition } n 0 { dispatch-file .CODE } { .TITLE title-string .IDENT / ident-string / .RADIX radix .TOC toc-string microinstruction branch-definition case-microinstruction end-definition } n 1 .END
user-machine-definition	.WIDTH 49R .BOUNDS [lower-bound : upper-bound] .ADDRESS J::=<8:0> .OBJECT <47:32>'<31:16>'<15:0>

A.1.2 Field-Definition

field-definition	.FIELD field-name { := / ::= } field-spec { field-value-name { := / ::= } value }n 0
field-spec	{ < left-bit { : right-bit }1 0 > }n ' 1 { , default }1 0

Examples:

```
.FIELD ALPHA ::= <40:30>

.FIELD BETA  ::= <10:5>'<20:17>,22

.FIELD GAMMA ::= <26>
   ON  ::= 0
   OFF ::= 1

.FIELD DELTA ::= <33:22>
  D1  ::=  0
  D2  ::=  2
  D3  ::=  4
  D4  ::=  6
```

A.1.3 Macro Definition

macro-definition	.MACRO macro-name { ({ formal }n,) }1 0 { := / ::= } { macro-body-part }n, 1
macro-body-part	{ field-name / { field-value-name / value / @ formal } } { macro-name { ({ actual }n,) }1 0 }

Examples:

```
.MACRO ALPHA ::=  A/B,C/D

.MACRO BETA(B1,B2)  ::=  A/@B1,C/@B2,D/@B1

.MACRO GAMMA(X,Y,Z)  ::=  BETA(@X,CT7),ALPHA,C42/20
```

A.1.4 Microinstruction

micro- instruction	{ address }$^{1}_{0}$ { label }$^{1}_{0}$ { field-name / { value / field-value-name } }^{n} , ; { macro-name { ({ actual }$^{n}_{1}$,) }$^{1}_{0}$ }$_{1}$

Examples:

```
     6200:
ALPHA:    A/B;

BETA:     A/B,C/D,E/F,
          G/H;

GAMMA:    MC1(X,Y),
          A/B,
          C,   D/E;

          MC2,
          X/2;

     6412:
          A/B1,
          J/GAMMA;
```

A.1.5 Target Assignment Construct

target-assignment	branch-definition { case-microinstruction }$_{1}^{n}$ { end-definition }$_{0}^{1}$
branch-definition	branch-label: .BEGIN = { 0 / 1 / * }$_{1}^{n}$ { address-range }$_{0}^{1}$
case-micro-instruction	.CASE case-number OF branch-label microinstruction
end-definition	.ENDB branch-label

Example:

```
ALPHA:
.BEGIN=0
BETA:
.BEGIN=10*10[6240:6277]

.CASE 0 OF ALPHA
A0:  A/B,
     J/B1;

.CASE 1 OF ALPHA
A1:  C/D,
     J/B1;

.ENDB ALPHA

.CASE 0 OF BETA
01:  A/B,C/D,
     J/G1;

.CASE 2 OF BETA
02:  X/4
     J/G2
```

A.1.6 MICRO-11/60 Elements

title-string ident-string toc-string	{ radix-50-char } 64 1
radix-50-char	{ alphabetic | digit | $ | . }
alphabetic	{ A | B | ... | Z }
digit	{ 0 | 1 | ... | 9 }
address-range	[low-address : high-address]
label field-name field-value-name macro-name formal actual entry-label	alphabetic { name-char } 31 0
op-code	octal-digit
lower-bound upper-bound low-address high-address address default-value value	octal-integer
left-bit right-bit	decimal-integer
name-char	{ radix-50-char | _ | % | [|] }

A.2 MDT COMMAND SYNTAX

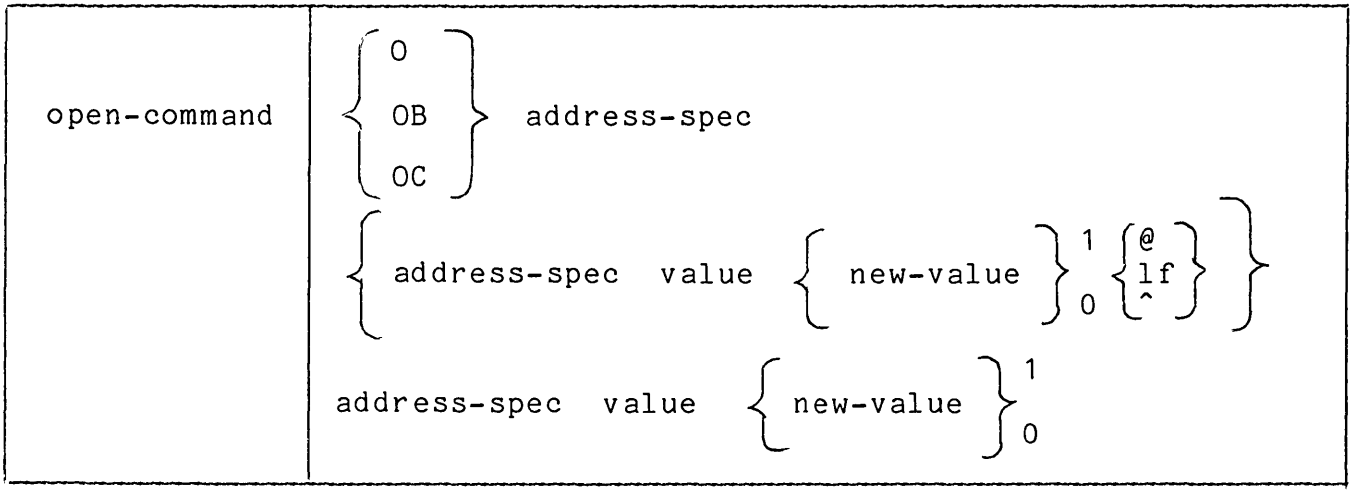
A.2.1 MDT-Session

mdt-session MDT>	{ open-command | breakpoint-command | display-command | control-command } n 1

Example:

```
MDT>O#46000                          Open main memory address
                                     46000
#46000<15:0>  0  2  (lf)             Set values
#46002<15:0>  0  4  (lf)
#46004<15:0>  0  6
MDT>O#46100                          Open memory address 46100
#46100<15:0>  0  1  (lf)             Set values
#46102<15:0>  0  3  (lf)
#46104<15:0>  0  5
MDT>B6200                            Set micro memory
                                     breakpoint at 6200
MDT>G#46200                          Start execution at main
                                     memory address 46200
BREAKPOINT NUMBER 0 AT ADDRESS 6200
MDT>O$0                              Examine register 0
$0<15:0>  12
MDT>P                                Proceed from breakpoint
BREAKPOINT NUMBER 0 AT ADDRESS 6200
MDT>B#46210                          Set main memory
                                     breakpoint at 46210
MDT>P                                Proceed from breakpoint
BREAKPOINT NUMBER 1 AT ADDRESS 46210
MDT>O#46000                          Examine main memory
                                     address 46000-46004
#46000<15:0>  3  (lf)
#46002<15:0>  7  (lf)
#46004<15:0>  13
```

A.2.2 Open-Command



Restrictions:

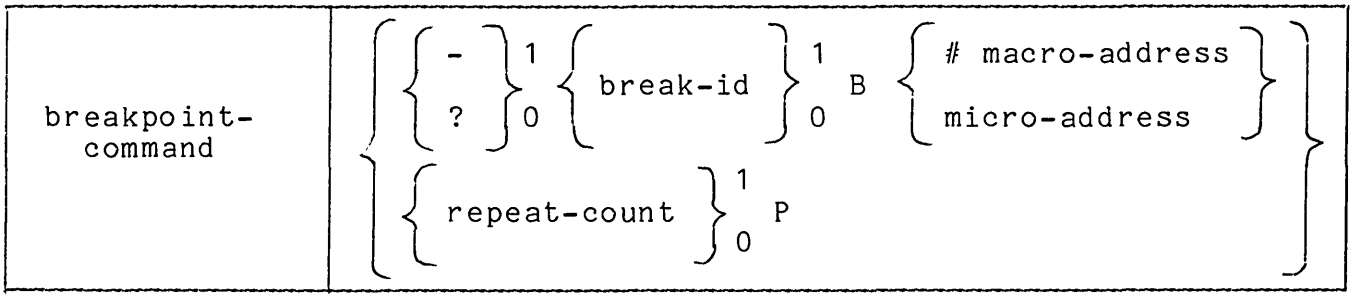
The address-spec used with an OB or OC command must be a main memory byte address.

The terminator @ must not be used with an address-spec that is a register or that has a bit range or with the commands OB or OC.

Examples:

```
MDT>06200<40:38>                Open bits 40 through 38 of
                                microaddress 6200
6200<40:38>  0  1  (lf)         Look at same bits of next address
6201<40:38>  1

MDT>O#46000                     Open main memory address 46000
#46000<15:0>  46300  @          Look at the address pointed to
#46300<15:0>  46520  @          Again
#46520<15:0>  0      ^          Look at the previous word
#46516<15:0>  2234

MDT>OB#46701                    Open the main memory byte location
                                46701
#46701<7:0>  71

MDT>OC#46703                    Open the character at main memory
                                byte location 46703
#46703<7:0>  G

MDT>O$PSW                       Open the register $PSW
$PSW<15:0>   223  ^             Look at previous register in
                                microstate table
$7<15:0>  0  123  ^             Again
$6<15:0>  7
```

A.2.3 Breakpoint-Command



Restrictions:

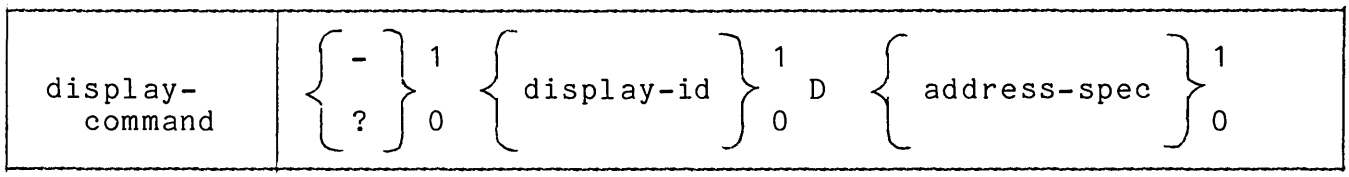
If the qualifier '-' is given, then either a break-id or a break-address can be given, but not both.

If the qualifier '?' is given, then neither a break-id nor break-address can be given.

Examples:

MDT>B#46000	Set breakpoint at main memory address 46000.
MDT>4B6400	Set a breakpoint with id 4 at micro address 6400.
MDT>?B	List the breakpoints that are set.
MDT>-B#46000	Delete the breakpoint at main memory address 46000.
MDT>-4B	Delete the breakpoint with id 4.
MDT>2P	Proceed from the current breakpoint and pass through 2 breakpoints at that address before halting.
MDT>-B	Delete all breakpoints.

A.2.4 Display-Command



Restrictions:

If the qualifier '-' is given, then either a display-id or an address-spec can be given, but not both.

If the qualifier '?' is given, then neither a display-id nor an address-spec can be given.

If no qualifier is given, then the address-spec must be given.

Examples:

MDT>D$PSW	Add $PSW to the display list.
MDT>D6400<ALU>	Add the ALU field of microinstruction 6400 to the display list.
MDT>?D	Print out the display list.
MDT>-D$PSW	Delete $PSW from the display list.
MDT>-1D	Delete the display-item with id 1 from the display list.
MDT>-D	Clear the display list by deleting all displays.

A.2.5 Control Command

control-command	{ G { # macro-address / micro-address } / L / R }

Examples:

G#46000	Start execution at main memory 46000.
L	Reload the Writable Control Store.
R	Reset the debugger by deleting all breakpoints and displays.

A.2.6 MDT Elements

address-spec	{ # macro-address { bit-range }1 0 { , relocation-reg }1 0 micro-address { bit-range <field-name> }1 0 $ register-name { bit-range }1 0 }
macro-address	{ 0 | 2 | 4 | ... | 17776 }
micro-address	{ 6000 | 6001 | ... | 7777 }
bit-range	< left-bit { : right-bit }1 0 >
value new-value	octal-integer
break display-id repeat-count left-bit right-bit	decimal-integer

A.3 COMMAND LANGUAGE SYNTAX

A.3.1 MICRO-11/60 Command Syntax

in-direct-method	>@MIC >* ENTER MICROPROGRAM SOURCE FILE SPECIFICATION[S]: in-spec >* LIST? [Y/N]: { Y / N }
direct-assembly	>MIC assembly-command-line
assembly-command-line	output-spec = in-spec
output-spec	{ object-file }$^{1}_{0}$ { , list-file }$^{1}_{0}$
in-spec	{ in-file }$^{n}_{1}$
object-file list-file in-file	file-spec { / switch }$^{2}_{0}$
file-spec	{ dev: }$^{1}_{0}$ { [ppn] }$^{1}_{0}$ file-name { . .ext }$^{1}_{0}$
switch	{ MX | BT }

A.3.2 MLD Command Syntax

IN-DIRECT-LOAD	>@MLD * ENABLE WCS?[Y/N]: { Y / N } * ENTER MICROPROGRAM OBJECT FILE SPECIFICATION[S]: obj-spec
obj-spec	{ file-spec }$_{1}^{n}$
direct-load	>MLD WCS= { MICPAK , }$_{0}^{1}$ { file-spec }$_{1}^{n}$

Example:

```
>@MLD
>; MLD.CMD\LOAD WCS
>;
>* ENABLE WCS[Y/N]:Y
>* ENTER MICROPROGRAM OBJECT FILE SPECIFICATION[S]:MPROG.OBJ
>MLD WCS=MICPAK,MPROG
>RUN MSTART                    !MSTOP SHOULD BE RUN WHEN FINISHED
>@<EOF>

>MLD WCS=MICPAK,LNKLST,MATPAK
```

A.3.3 MDT Command Syntax

mdt- call	>@MDT >* ENTER MICROPROGRAM OBJECT FILE SPECIFICATION[S]:file-spec ENTER MINUTES TO WAIT BEFORE SHUTDOWN: minutes ENTER MINUTES BETWEEN MESSAGES: minutes AT: -- PAUSING. TO CONTINUE TYPE 'RES ...AT.' >RES AT. >* HAVE YOU PROTECTED EVERYTHING YOU WANT PROTECTED? { Y } 1 { N } 0 MDT>

Example:

```
>@MDT
>; MDT.CMD \LOAD WCS AND START MDT (FROM PRIVILEGED UIC)
>;
>; S H U T S   R S X   D O W N
>;
>* ENTER MICROPROGRAM OBJECT FILE SPECIFICATION[S]: MATADD
>SET /MAIN=WCS:7600:200:DEV
>INS [1,1]WCS
>INS MLD/TASK=...MLD
>MLD WCS=MICPAK,MDTMIC,SMKMIC
>RUN MSTART !ENABLES WCS (SHOULD RUN MSTOP AS SOON AS RSX BACK
UP)
>;       (SUGGEST PUTTING MSTOP IN STARTUP.CMD FILE)
>RUN SHUTUP
RSX11M SHUT DOWN PROGRAM
>
ENTER MINUTES TO WAIT BEFORE SHUTDOWN:
0
ENTER MINUTES BETWEEN MESSAGES:
0
ALL FURTHER LOGINS ARE DISABLED
>
01-SEP-77 11:56 PLEASE FINISH UP, 0 MINUTES BEFORE SHUTDOWN
: TYPE "RES AT." WHEN SHUTDOWN COMPLETED
>
AT. -- PAUSING. TO CONTINUE TYPE "RES ...AT."
                                                DMO DB0:
>
>RES AT.
AT. -- CONTINUING
>
>
>* HAVE YOU PROTECTED EVERYTHING YOU WANT PROTECTED? [Y/N]:Y
>BOOT MDT ! T H I S   T A K E S   R S X   D O W N
MICRO DEBUGGING TOOL. VERSION #0
MDT>
```

APPENDIX B

THE 11/60 PREDEFINITIONS

The 11/60 predefinitions define the 11/60 architecture, specify the fields of the 11/60 microword and supply a set of field-value-names and macro-names that are useful in writing microprograms for the 11/60 WCS.

The predefinitions are given here as an output listing from MICRO-11/60 assembler. The microprogrammer can, by reading this listing, familiarize himself with all the predefined names that can be used in a microprogram. The table-of-contents at the beginning of this listing is useful for finding a particular field or macro name.

The programmer specifies the predefinitions file as the first input file in an assembly. The MICRO-11/60 assembler, therefore, reads this file first and incorporates all the names defined in this file into its internal tables.

In some advanced applications, the programmer may wish to assemble without the predefinitions file. In such a case, the following lines, which define the 11/60 architecture, must be supplied:

```
.WIDTH    49R
.BOUNDS   [6000:7777]
.OBJECT   <47:32>'<31:16>'<15:0>
.ADDRESS  J ::= <8:0>
```

The .WIDTH keyword specifies the number of bits and the ordering (right-to-left) of bits within the microword. The .BOUNDS keyword defines the legal address limits for the program. The .OBJECT keyword defines the order of the bits within the object module. The .ADDRESS keyword locates the bits within the microword used for the address field.

B.1 PREDEFINITIONS SOURCE LISTING

TABLE OF CONTENTS

```
 633 --         *     REGISTER LOADING
 637 --         *       <LOADRES>-LOAD RESIDUAL CONTROL REGISTER
 647 --         *       <LOADCOUNT>-LOAD COUNTER
 657 --         *   SEQUENCING FIELD
 661 --         *     <UBF>-BUT MICROBRANCH FIELD
 666 --         *       NO BUT
 670 --         *       ACTIVE ONLY
 691 --         *       INACTIVE ONLY
 763 --         *       BOTH ACTIVE AND INACTIVE
 786 --         *     <UPF>-MICRO POINTER FIELD
 812 --         *   MISCELLANEOUS FIELDS
 816 --         *     <NEXT-PAGE>-NEW PAGE ADDRESS LOADED DURING BUT[SUBROUTINE]
 822 --         *     <MULTIPLE>-SELECT CODE FOR BUT[MULTIPLE]
 837 --         *     EMIT FIELD - IMMEDIATE DATA FROM MICROWORD
 864 --         *     RETURN ADDRESS - FOR MICROSUBROUTINE CALLS
 870 --         *     UCON SELECTION AND CONTROL FIELDS
 873 --         *         SELECTION
 892 --         *         CONTROL (ALSO TMS ROUTINES)
 967 --         *     LOCAL STORE FIELDS
 990 --         * MACRO DEFINITIONS
 993 --         *   PRIMITIVE OPERATIONS
 996 --         *     TIMING
1032 --         *     WRITING THE A AND B SCRATCH PADS
1059 --         *     ASP AND BSP PHYSICAL REGISTER ADDRESSES
1091 --         *     ASP AND BSP BASE MACHINE FUNCTIONAL REGISTER ADDRESSES
1165 --         *     ASP AND BSP INDIRECT REGISTER ADDRESSES
1191 --         *     ASP, BSP INDIRECT ADDRESSING
1206 --         *     WRITING THE C SCRATCH PAD
1215 --         *     CSP IMPLIED ADDRESSING
1228 --         *     CSP DIRECT ADDRESSING
1243 --         *     SHIFT TREE SPECIFICATION
1247 --         *       ENABLED ONTO BUS A
1310 --         *       FIRST TWO LEVELS ONLY [AMUX,
1320 --         *     ALU FUNCTIONS
1341 --         *     COUT GENERATION
1356 --         *     CLOCKS
1360 --         *       BASIC REGISTER CLOCKS [D, SR, BA, CC]
1375 --         *       REDEFINED FROM SP REWRITE FIELD [RES, COUNTER]
1383 --         *       RES REGISTER CONTROL VALUES [FROM EMIT]
```

```
1405 --         *       CC CONTROL  [FROM EMIT]
1414 --         *     BUS CONTROL MACROS
1434 --         *     UCON CONTROL MACROS
1442 --         *       PROCESSOR UCON CONTROL SETUP
1464 --         *       CACHE/KT UCON CONTROL
1498 --         *       I/O UCON CONTROL
1503 --         *         BUS CONTROL
1524 --         *         CONSOLE I-O
1551 --         *         REMOTE CONSOLE INTERFACE
1562 --         *     MICROBRANCH FIELD MACROS
1580 --         *     MISCELLANEOUS
1582 --         *       OTHER SOURCES ENABLED FOR A-BUS
1588 --         *       PAGING, RETURN REGISTER
1605 --         *   ADVANCED OPERATIONS
1609 --         *     DATA INTO CSP, AT P3 ONLY
1661 --         *     DATA INTO ASP, BSP, AT P2-T * P3
1873 --         *     D AND SR <- (BUS-A FCN BUS-B), AT P2-T OR P3-T
1916 --         *     D[C] GETS SET
1936 --         *     D-REGISTER <- [BBUS = ABUS], BITWISE, AT P2-T OR P3-T
1973 --         *     D-REGISTER <- D-REGISTER THRU SHIFT-TREE
2007 --         *     D <- WHATEVER'S LEFT, AT P2-T OR P3-T
2055 --         *     SR <- DATA, AT P2 T OR P3 T
2087 --         *     RES-REG OPERATION MACROS
2096 --         *     BASE MACHINE COUNTER
2104 --         *     ENABLE ON BUS-A/B ONLY
2130 --         *     LOADING BA REGISTER
2143 --         *     D AND SR TOGETHER
2151 --         *     UCON FUNCTIONS
2155 --         *       PROCESSOR UCON FUNCTIONS
2191 --         *       CACHE/KT UCON FUNCTIONS
2237 --         *       I-O UCON FUNCTIONS
2266 --         *       CONSOLE UCON FUNCTIONS
2290 --         *       DBUF UCON FUNCTIONS
2300 --         *       MULTIPLE UCON FUNCTIONS
2311 --         *       WCS FUNCTIONS
2325 --         *     JAM UPP LOG MACROS
```

```
!
!
!
!
!
!                    I D E N T I F I C A T I O N
!                    ---------------------------
!
!
!       PRODUCT CODE:
!
!       PRODUCT NAME:   PDP 11/60 DEFINITION FILE
!
!       MAINTAINER:     11/60 ENGINEERING
!
!       AUTHOR:         11/60 ENGINEERING
!
!       DATE CREATED:   18-JANUARY-1977
!
!       LAST REVISION:  18-JANUARY-1977
!
!                       23-MAY-1977
!                       28-JUL-1977
!
!
!
!
!
!
```

```
56
57
58
59  !=====================================================================
60
61  .TOC     * IDENTIFICATION
62  !
63  .TITLE   PDP 11/60 DEFAULT MACROS
64
65  !=====================================================================
66
67  .TOC     * REVISION HISTORY
68
69  .IDENT   /V01/
70
71
72
73
74
75  !=====================================================================
76
77  .TOC     * MICROWORD FIELD DEFINITIONS
78
79  !        NOTE:   THE FOLLOWING ARE THE ASSIGNED RANGES OF THE
80  !                MICROWORD FIELD BIT DEFINITIONS USED IN THIS
81  !                SOURCE LISTING:
82  !
83  !                BITS[NUMBER]        WHERE HELD
84  !                ------------        --------------------------
85  !
86  !                [47:00]             WCS CONTROL STORE
87
88
89  !=====================================================================
```

```
 90
 91  .TOC     * MICROWORD BIT LAYOUT
 92  !        BASE              1-EMIT                                  6-RETURN
 93  !        MACHINE           2-SHIFT-TREE     4-UCON-DATA            7-PAGING
 94  !        CONTROL           3-RESIDUAL-CTL   5-CSP-ADDRESS          8-UCON-CONTROL
 95  !        -------           --------------   -------------          --------------
 96  !U47     ALU3              1-EMITH15        4-UCONL10
 97  !U46     ALU2              1-EMITH14        4-UCON-I/O-SEL         6-RETR11
 98  !U45     ALU1              1-EMITH13        4-UCON-WCS-SEL         6-RETR10
 99  !U44     ALU0              1-EMITH12        4-UCON-KT-SEL          6-RETR09
100  !U43     BEN1
101  !U42     BEN0                               4-UCONL09
102  !U41     BSEL1             1-EMITM11        4-UCONL08              6-RETR08
103  !U40     BSEL0             1-EMITM10        4-UCONL07              6-RETR07
104  !U39     AEN1              1-EMITM09        4-UCONL06              6-RETR06
105  !U38     AEN0              1-EMITM08        4-UCONL05              6-RETR05
106  !U37     ASEL1             1-EMITL07                               6-RETR04
107  !U36     ASEL0             1-EMITL06        4-UCON-PROC-SEL        6-RETR03
108  !U35     RIF2              1-EMITL05        4-UCONM12              6-RETR02
109  !U34     RIF1
110  !U33     RIF0              1-EMITL03        4-UCON-FP-SEL          6-RETR00
111  !U32     COUT2             1-EMITL02        4-UCONH15              7-NEXT-PAGE2
112  !U31     COUT1             1-EMITL01        4-UCONH14              7-NEXT-PAGE1
113  !U30     COUT0             1-EMITL00        4-UCONH13              7-NEXT-PAGE0
114  !U29 *   WHEN
115  !U28 *   CLK-D
116  !U27 *   CLK-SR
117  !U26 *   CLK-BA
118  !U25 *   SET-CC
119  !U24 *   BEGIN
120  !U23     SELECT(=0)        2-BMUX           5-CSPADR3              8-SELECT(=1)
121  !U22     BUSCOD2           2-AMUX2          5-CSPADR2              8-FLPGO
122  !U21     BUSCOD1           2-AMUX1          5-CSPADR1              8-UCON-XFER
123  !U20     BUSCOD0           2-AMUX0          5-CSPADR0              8-UCON-LOAD
124  !U19 *   WRCSP
125  !U18     HI/LO             3-LOAD-RES
126  !U17     WRSEL
127  !U16     WRB               3-LOAD-COUNT
128  !U15     WRA
129  !U14 *   MOD(=0)           3-MOD(=1)
130  !U13 *   UBF4
131  !U12 *   UBF3
132  !U11 *   UBF2
133  !U10 *   UBF1
134  !U09 *   UBF0
135  !U08 *   UPF8
136  !U07 *   UPF7
137  !U06 *   UPF6
138  !U05 *   UPF5
139  !U04 *   UPF4
140  !U03 *   UPF3
141  !U02 *   UPF2
142  !U01 *   UPF1
143  !U00 *   UPF0     ( * = DEDICATED TO THE CORRESPONDING SINGLE FUNCTION )
```

```
144
145
146  !          BASE                              LOCAL          UCON
147  !          MACHINE          9-RES-BITS       STORE          PROCESSOR
148  !          CONTROL          10-MULTIPLE      DEFINITION     CONTROL
149  !          -------          -----------      ----------     ---------
150  !U47       ALU3                              COLZERO15      PS<3:0>-CLK
151  !U46       ALU2             9-HISMUXSELL     COLZERO14
152  !U45       ALU1             9-SRS1-L         COLZERO13
153  !U44       ALU0             9-SRS0-L         COLZERO12
154  !U43       BEN1                              COLZERO11
155  !U42       BEN0                              COLZERO10      UBREAK-CLK
156  !U41       BSEL1            9-GUARD-EN-H     COLZERO09          <NU>
157  !U40       BSEL0                             COLZERO08          <NU>
158  !U39       AEN1                              COLZERO07      SEL-HBMUX1L
159  !U38       AEN0                              COLZERO06      SEL-HBMUX0L
160  !U37       ASEL1                             COLZERO05
161  !U36       ASEL0                             COLZERO04
162  !U35       RIF2                              COLZERO03      FPS<7:4>-CLK
163  !U34       RIF1                              COLZERO02      PS<7:4>-CLK
164  !U33       RIF0                              COLZERO01
165  !U32       COUT2            10-MULT-SEL2     COLZERO00      IR-CLOCK
166  !U31       COUT1            10-MULT-SEL1     COLONE15       PS<15:12>-CLK
167  !U30       COUT0            10-MULT-SEL0     COLONE14       FLAG<8:0>-CLK
168  !U29  *    WHEN                              COLONE13
169  !U28  *    CLK-D                             COLONE12
170  !U27  *    CLK-SR                            COLONE11
171  !U26  *    CLK-BA                            COLONE10
172  !U25  *    SET-CC                            COLONE09
173  !U24  *    BEGIN                             COLONE08
174  !U23       SELECT(=0)                        COLONE07
175  !U22       BUSCOD2                           COLONE06
176  !U21       BUSCOD1                           COLONE05
177  !U20       BUSCOD0                           COLONE04
178  !U19  *    WRCSP                             COLONE03
179  !U18       HI/LO                             COLONE02
180  !U17       WRSEL                             COLONE01
181  !U16       WRB                               COLONE00
182  !U15       WRA                               COLTWO15
183  !U14  *    MOD(=0)                           COLTWO14
184  !U13  *    UBF4                              COLTWO13
185  !U12  *    UBF3                              COLTWO12
186  !U11  *    UBF2                              COLTWO11
187  !U10  *    UBF1                              COLTWO10
188  !U09  *    UBF0                              COLTWO09
189  !U08  *    UPF8                              COLTWO08
190  !U07  *    UPF7                              COLTWO07
191  !U06  *    UPF6                              COLTWO06
192  !U05  *    UPF5                              COLTWO05
193  !U04  *    UPF4                              COLTWO04
194  !U03  *    UPF3                              COLTWO03
195  !U02  *    UPF2                              COLTWO02
196  !U01  *    UPF1                              COLTWO01
197  !U00  *    UPF0                              COLTWO00
198
```

```
199
200 !=========================================================================
201
202 .TOC     * MICROWORD FIELD SPECIFICATION
203
204
205 !-------------------------------------------------------------------------
206
207 .TOC     *   MICROWORD FIELD FORMAT
208
209 .RADIX   8               ! ALL NUMBERS ARE OCTAL, UNLESS OTHERWISE NOTED
210
211 .WIDTH   49R             ! MICROWORD IS 4810 BITS WIDE, BIT <00> IS
212                          !RIGHTMOST BIT. BIT 48 USED FOR NULL FIELD.
213
214 .BOUNDS  [6000:7777]             ! ADDRESSES ARE 12 BITS, ON PAGES 6:7
215
216 .OBJECT  <47:32>'<31:16>'<15:00> ! OUTPUT FORMAT (DEFAULT SPEC)
217
218 !-------------------------------------------------------------------------
219
220 .TOC     *   NULL FIELD/MACRO SPECIFICATION
221
222 .FIELD   N::=<48>
223 .MACRO   NULL::=N/0
224
225
226
227 .TOC     *   ALU AND INTERNAL DATA BUS CONTROL
228
229
230
231 .TOC     *     <ALU>-ALU FUNCTION CONTROL BITS
232 !SPECIFIES ALU FUNCTION CODE AND CINMUX SELECT. ALWAYS IN EFFECT.
233 .FIELD   ALU::=<47:44>
234 !                        ---FUNCTION---     LOG/AR  ALUS<3:0> H  CINMUX L
235 NOT-A::=00               !COMPLEMENT A,          L    0000   -1
236 A-PLUS-B-PLUS-PS[C]::=01 !ADD,                   A    1001   -PS[C]
237 NOT-A-AND-B::=02         !AND,                   L    0010   -PS[C]
238 ZERO::=03                !ZERO,                  L    0011   -PS[C]
239 A-PLUS-B-PLUS-D[C]::=04  !PLUS,                  A    1001   -D[C]
240 A-PLUS-NOT-B-PLUS-D[C]::=05 !PLUS,               A    0110   -D[C]
241 A-XOR-B::=06             !XOR,                   L    0110   -D[C]
242 A-AND-NOT-B::=07         !AND,                   L    0111   -D[C]
243 DIVIDE::=10              !DIVIDE STEP,
244                          !SUB,   IF D[C]H=1      A    0110   -D[C]=-1
245                          !ADD,   IF D[C]H=0      A    1001   -D[C]=-0
246 A-PLUS-B::=11            !PLUS,                  A    1001   -0
247 B::=12                   !SELECT B,              L    1010   -0
248 A-AND-B::=13             !AND,                   L    1011   -0
249 A-PLUS-B-PLUS-1::=14     !PLUS,                  A    1001   -1
250 A-MINUS-B::=15           !MINUS,                 A    0110   -1
251 A-IOR-B::=16             !IOR,                   L    1110   -1
252 A::=17                   !SELECT A,              L    1111   -1
253
```

```
254
255
256  .TOC     *     <BEN>-B-BUS DATA SOURCE
257  !SPECIFIES GATING OF DATA ONTO B-BUS. ALWAYS IN EFFECT.
258  .FIELD   BEN::=<43:42>
259           BSPLO::=0              !DIRECT BSP LOCATIONS 00-17
260           BSPHI::=1              !DIRECT BSP LOCATIONS 20-37
261           CSP::=2                !USE <CSPADDR> [SIC] AS ADDRESS (4 BIT)
262           BASCON::=3             !1 OF 4 BASE CONSTANTS IN CSP17 TO
263                                  !CSP14 (2 BIT)
264
265
266
267  .TOC     *     <BSEL>-B-BUS SOURCE SELECTION CONTROL
268  !SPECIFIES CONTROL OF INDIVIDUAL B-BUS SOURCES. ALWAYS IN EFFECT.
269  .FIELD   BSEL::=<41:40>
270  !THIS FIELD NOT USED WHEN BEN/CSP IS SPECIFIED
271  !CSP17 TO CSP14 IMMEDIATE ADDRESS WHEN BEN/BASCON
272           B17::=0                !
273           B16::=1                !
274           B15::=2                !
275           B14::=3                !
276           ONE::=0                !ONE CONSTANT
277           ZERO::=1               !ZERO CONSTANT
278           MD::=2                 !MEMORY DATA
279           TWO::=3                !TWO CONSTANT
280  !USED IN CONJUNCTION WITH <RIF> FOR SP ADDRESS WHEN
281  !BEN/BSPLO OR BEN/BSPHI
282           DF::=0                 !DESTINATION FIELD
283           SF::=1                 !SOURCE FIELD
284           IMMED0::=2             !DIRECT ADDRESS, LOW BIT=0
285             R00::=2              !FOR JOINT USE W/ RIF FIELD
286             R02::=2              !
287             R04::=2              !
288             R06::=2              !
289             R10::=2              !
290             R12::=2              !
291             R14::=2              !
292             R16::=2              !
293           IMMED1::=3             !DIRECT ADDRESS, LOW BIT=1
294             R01::=3              !FOR JOINT USE W/ RIF FIELD
295             R03::=3              !
296             R05::=3              !
297             R07::=3              !
298             R11::=3              !
299             R13::=3              !
300             R15::=3              !
301             R17::=3              !
302
```

```
303
304  .TOC     *     <AEN>-A-BUS DATA SOURCE
305  !SPECIFIES GATING OF DATA ONTO A-BUS. ALWAYS IN EFFECT.
306  .FIELD   AEN::=<39:38>
307           XMUX::=0                !XMUX=SR OR FLTPT ASSEMBLE
308           CMUX::=1                !SHIFT TREE
309           ASPLO::=2               !DIRECT ASP LOCATIONS 00-17
310           ASPHI::=3               !DIRECT ASP LOCATIONS 20-37
311
312
313
314  .TOC     *     <ASEL>-A-BUS SOURCE SELECTION CONTROL
315  !SPECIFIES CONTROL OF INDIVIDUAL A-BUS SOURCES. ALWAYS IN EFFECT.
316  .FIELD   ASEL0::=<36>
317  !XMUX CONTROL WHEN AEN/XMUX  [USES ASEL0 ONLY]
318           SR::=0                  !SR OUTPUT ONTO BUS-A
319           FLTPT::=1               !FLTPT-ASSEMBLE ONTO BUS-A
320  .FIELD   ASEL::=<37:36>
321  !CMUX CONTROL WHEN AEN/CMUX.  SHIFTS CMUX INPUT APPROPRIATE AMOUNT
322           LEFT-1::=0              !LOW BIT GETS SENDMUX OUTPUT
323           DIRECT::=1              !OUTPUT=INPUT
324           RIGHT-1::=2             !HIGH BIT GETS D[C]
325           RIGHT-2::=3             !HIGH BITS BOTH GET D[C]
326  !USED IN CONJUNCTION WITH <RIF> FOR SP ADDRESS WHEN
327  !AEN/ASPLO OR AEN/ASPHI
328           IMMED0::=0              !DIRECT ADDRESS, LOW BIT=0
329             R00::=0               !FOR JOINT USE W/ RIF FIELD
330             R02::=0               !
331             R04::=0               !
332             R06::=0               !
333             R10::=0               !
334             R12::=0               !
335             R14::=0               !
336             R16::=0               !
337           IMMED1::=1              !DIRECT ADDRESS, LOW BIT=1
338             R01::=1               !FOR JOINT USE W/ RIF FIELD
339             R03::=1               !
340             R05::=1               !
341             R07::=1               !
342             R11::=1               !
343             R13::=1               !
344             R15::=1               !
345             R17::=1               !
346           DF::=2                  !DESTINATION FIELD
347           SF::=3                  !SOURCE FIELD
348
```

```
349
350
351 .TOC     *     <RIF>-ASP, BSP REGISTER IMMEDIATE FIELD
352 !SPECIFIES ADDRESSES WITH ASP, BSP ALONG WITH AEN, ASEL & BEN, BSEL
353 .FIELD   RIF::=<35:33>
354           R00-OR-01::=4           !LOW BIT IS 0/1, SPECIFIED BY
355             R00::=4               !
356             R01::=4               !
357           R02-OR-03::=5           !USING EITHER IMMED0/IMMED1 MODES
358             R02::=5               !
359             R03::=5               !
360           R04-OR-05::=6           !
361             R04::=6               !
362             R05::=6               !
363           R06-OR-07::=7           !
364             R06::=7               !
365             R07::=7               !
366           R10-OR-11::=0           !
367             R10::=0               !
368             R11::=0               !
369           R12-OR-13::=1   !ADDR<3:0>H = -RIF<2>H # RIF<1:0>H # A/BSEL<0>H
370             R12::=1               !
371             R13::=1               !
372           R14-OR-15::=2           !
373             R14::=2               !
374             R15::=2               !
375           R16-OR-17::=3           !
376             R16::=3               !
377             R17::=3               !
378
379
380 .TOC     *     <COUT>-CARRY OUT BIT MUX SELECTION
381 !SPECIFY INPUT TO D[C] REGISTER, LOADED WHEN D REGISTER LOADED.
382 !THIS IS ALWAYS IN EFFECT.
383 .FIELD   COUT::=<32:30>
384           CIN::=0                 !OUTPUT OF CINMUX [SIC]
385           PS[C]::=1               !PS C-BIT
386           ALU00::=2               !ALU OUTPUT BIT 00
387           ALU07::=3               !ALU OUTPUT BIT 07
388           ALU15::=4               !ALU OUTPUT BIT 15
389           COUT07::=5              !BYTE CARRY BIT
390           COUT15::=6              !WORD CARRY BIT
391           D[C]::=7                !PROPOGATE [SAVE] LAST D[C]
392
393 !-------------------------------------------------------------------------
394
```

```
395
396
397  .TOC     *   CLOCKS
398
399
400
401  .TOC     *     <WHEN>-D/SR WHEN TO CLOCK
402  !SPECIFY CLOCK D/SR REGISTERS AT P2 T OR P3 T. ALWAYS IN EFFECT.
403  .FIELD   WHEN::=<29>,0
404           AT-P2-T::=0              !CLOCK D AND/OR SR AT P2 T[100 NS].
405           AT-P3-T::=1              !CLOCK D AND/OR SR AT P3 T[150 NS].
406
407
408
409  .TOC     *     <CLKD>-ENABLE D-REGISTER CLOCKING
410  !ENABLES CLOCKING OF D-REGISTER. ALWAYS IN EFFECT.
411  .FIELD   CLKD::=<28>,0
412           NO::=0                   !NOP
413           YES::=1                  !CLOCK D[C], D-REGISTER AT <WHEN>
414
415
416
417  .TOC     *     <CLKSR>-ENABLE SR-REGISTER CLOCKING
418  !ENABLES CLOCKING OF SR-REGISTER. ALWAYS IN EFFECT.
419  .FIELD   CLKSR::=<27>,0
420           NO::=0                   !NOP
421           YES::=1                  !CLOCK SR-REGISTER AT <WHEN>
422
423
424
425  .TOC     *     <CLKBA>-ENABLE CLOCKING OF BA-REGISTER
426  !ENABLES CLOCKING OF BA-REGISTER AT P1 T[60 NS]. ALWAYS IN EFFECT.
427  .FIELD   CLKBA::=<26>,0
428           NO::=0                   !NOP
429           YES::=1                  !CLOCK BA-REGISTER AT P1 T[60 NS].
430
431
432
433  .TOC     *     <SCC>-ENABLE SETTING OF PS CONDITION CODES
434  !ENABLE CLOCKING OF PS CONDITION CODES AT P2 T[100 NS] OF NEXT UWORD.
435  !D MUST BE CLOCKED AT P2 T OR EARLIER OF PREVIOUS MICROWORD.
436  !THIS IS ALWAYS IN EFFECT.
437  .FIELD   SCC::=<25>,0
438           NO::=0                   !NOP
439           YES::=1                  !ENABLE CLOCKING IN NEXT UWORD
440
441  !-----------------------------------------------------------------------
442
```

```
443
444
445  .TOC     *   BUS/UCON & CSP-ADDRESS & SHIFT-TREE CONTROL
446
447
448
449  .TOC     *     BUS/UCON CONTROL
450
451
452  .TOC     *       <BEGIN>-BEGIN BUS/UCON OPERATION
453  !INITIATE BUS XOR UCON OPERATION. ALWAYS IN EFFECT.
454  .FIELD   BEGIN::=<24>,0
455           NO::=0                    !NOP FOR BUS AND UCON OPERATIONS
456           YES::=1                   !BEGIN OPERATION SPECIFIED
457
458
459
460  .TOC     *       <SELECT>-SELECT BUS OR UCON
461  !SELECT BUS XOR UCON. ONLY USED IF BEGIN/YES.
462  .FIELD   SELECT::=<23>
463           BUS::=0                   !SELECT BUS
464           UCON::=1                  !SELECT UCON
465
466
467
468  .TOC     *       BUS CONTROL
469
470
471  .TOC     *         <BUSCODE>-BUS CODE ACTION FIELD
472  !BUS ACTION CODES. ONLY USED IF BEGIN/YES & SELECT/BUS.
473  .FIELD   BUSCODE::=<22:20>
474           DATI-CLKIR::=0            !DATA IN, LOAD IR
475           DATI-NOINT::=1            !DATA IN, NO INTERNAL ADDRESS
476           DATO::=2                  !DATA OUT
477           DATIB::=3                 !DATA IN, ALLOW: ODD ADDRESS
478           DATIB[P]::=3              !DATA IN, ALLOW: ODD ADDRESS,
479                                     !FORCE TO PAUSE.
480           DATIP::=4                 !DATA IN, NO CACHE, LOCK BUS
481           DATOB::=5                 !DATA OUT, ALLOW: ODD ADDRESS
482           DATI::=6                  !DATA IN
483           DATI[P]::=6               !DATA IN, ALLOW: FORCE TO PAUSE
484           INVALIDATE::=7            !INVALIDATE CACHE  LOCATION FUNCTION
485
486
487
488  .TOC     *       UCON CONTROL
489
490
491
492  .TOC     *         <FLPGO>-START HOT FLOATING POINT
493  !INITIATES HOT FLOATING POINT FUNCTION.  ONLY USED IF BEGIN/YES
494  !AND SELECT/UCON.
495  .FIELD   FLPGO::=<22>
496           NO::=0                    !NOP
497           YES::=1                   !YELL GO
498
```

```
499
500
501  .TOC     *         <UCON-XFER>-UCON OPERATION
502  !EXECUTE A UCON FUNCTION. ONLY USED IF BEGIN/YES & SELECT/UCON.
503  .FIELD   UCON-XFER::=<21>
504           NO::=0                    !NOP
505           YES::=1                   !START UCON OPERATION
506
507
508
509  .TOC     *         <UCON-LOAD>-LOAD UCON REGISTER
510  !LOAD UCON CONTROL REGISTER. ONLY USED IF BEGIN/YES & SELECT/UCON.
511  .FIELD   UCON-LOAD::=<20>
512           NO::=0                    !NOP
513           YES::=1                   !LOAD UCON CONTROL REGISTER
514
515
516
517  .TOC     *     CSP ADDRESS SPECIFICATION
518
519
520  .TOC     *       <CSPADDR>-CSP IMMEDIATE ADDRESS
521  !SPECIFY CSP 4 BIT ADDRESS. ONLY USED IF BEN/CSP.
522  .FIELD   CSPADDR::=<23:20>
523           D00::=17                  !NOTE INVERSION
524           D01::=16                  !
525           D02::=15                  !
526           D03::=14                  !
527           D04::=13                  !
528           D05::=12                  !
529           D06::=11                  !
530           D07::=10                  !
531           D10::=07                  !
532           D11::=06                  !
533           D12::=05                  !
534           D13::=04                  !
535           D14::=03                  !
536           D15::=02                  !
537           D16::=01                  !
538           D17::=00                  !
539
540
541  .TOC     *     SHIFT CONTROL
542
543
544  .TOC     *       <BMUX>-SECOND LEVEL OF SHIFT TREE
545  !BMUX CONTROLS SHIFT RIGHT OF 0 OR 4. ALWAYS IN EFFECT.
546  .FIELD   BMUX::=<23>
547           DIRECT::=0                !AMUX<15:00>
548           RIGHT-4::=1               !4*D[C] # AMUX <15:04>
549
```

```
550
551
552  .TOC     *       <AMUX>-FIRST LEVEL OF SHIFT TREE
553  !AMUX CONTROLS INPUT OF D-REG/COUNTER TO TREE. ALWAYS IN EFFECT.
554  .FIELD   AMUX::=<22:20>
555           DIRECT::=0              !D<HI> # D<LO>
556           D[LO]#D[LO]::=1         !D<LO> # D<LO>
557           SIGNEXT::=2             !8*D[C] # D<LO>
558           COUNTER#D[LO]::=3       !COUNTER # D<LO>
559           COUNTER::=3             !SAME
560           D[HI]#D[HI]::=4         !D<HI> # D<HI>
561           SWAB::=5                !D<LO> # D<HI>
562           RIGHT-8::=6             !8*D[C] # D<HI>
563           COUNTER#D[HI]::=7       !COUNTER # D<HI>
564
565  !-------------------------------------------------------------------------
566
567
568
569  .TOC     *   SP REWRITE & REGISTER CLOCKS
570
571
572
573  .TOC     *     <WRCSP>-WRITE TO CSP
574  !WRITE CSP FROM DMUX [BUSDIN/CACHE]. ALWAYS IN EFFECT.
575  .FIELD   WRCSP::=<19>,0
576           NO::=0                  !NOP
577           YES::=1                 !ON P3, 120-150 NS.
578
579
580
581  .TOC     *     <MOD>-MODE CONTROL OF FOLLOWING BITS
582  !CONTROLS REDEFINITION OF SP REWRITE/REGISTER CLOCK BITS.
583  !THIS IS ALWAYS IN EFFECT.
584  .FIELD   MOD::=<14>,0
585           CLKSP::=0               !CONTROL ASP/BSP CLOCKING
586           LOADREG::=1             !CONTROL RES-REG/COUNTER LOADING
587
588
589
590  .TOC     *     SP REWRITE [A,B] CONTROL
591  !WHEN MOD/CLKSP
592
593
594  .TOC     *       <HILO>-SP HI/LO SELECT
595  !WHICH HALF OF SP'S TO REWRITE. ONLY IF MOD/CLKSP.
596  .FIELD   HILO::=<18>
597           LO::=0                  !REWRITE ENABLE A/B SP LO [00-17]
598             L::=0                 !
599           HI::=1                  !REWRITE ENABLE A/B SP HI [20-37]
600             H::=1                 !
601
602
```

```
603
604  .TOC     *       <WRSEL>-REWRITE ADDRESS SELECT
605  !WHICH WRITE ADDRESS TO USE ON REWRITE. ONLY IF MOD/CLKSP.
606  .FIELD   WRSEL::=<17>
607           A-ADDR::=0              !USE A ADDRESS ON REWRITE
608             A::=0                 !
609           B-ADDR::=1              !USE B ADDRESS ON REWRITE
610             B::=1
611
612
613
614  .TOC     *       <WRSP>-REWRITE A/B SELECT
615  !ENABLE REWRITE OF SPECIFIC SP'S. ONLY IF MOD/CLKSP.
616  .FIELD   WRSP::=<16:15>
617           NOP::=0                 !NO ASP/BSP REWRITE
618           WR-A::=1                !WRITE ASP ONLY, ON P3 120-150 NS.
619             A::=1                 !
620             ASP::=1               !
621           WR-B::=2                !WRITE BSP ONLY, ON P3 120-150 NS.
622             B::=2                 !
623             BSP::=2               !
624           WR-A-AND-B::=3          !WRITE BOTH ON P3
625             AB::=3                !
626             BA::=3                !
627             ABSP::=3              !
628             BASP::=3              !
629             BOTH::=3              !
630
631
632
633  .TOC     *     REGISTER LOADING
634  !WHEN MOD/LOADREG
635
636
637  .TOC     *       <LOADRES>-LOAD RESIDUAL CONTROL REGISTER
638  !ENABLE LOAD OF RESIDUAL CONTROL REGISTER FROM B-BUS.
639  !THIS IS ONLY IF MOD/LOADREG.
640  .FIELD   LOADRES::=<18>
641           NO::=0                  !NOP
642           YES::=1                 !LOAD RES WITH B-BUS<14:11>
643                                   !AT P2 T[100 NS], B-BUS<14> COMPLEMENTED
644
645
646
647  .TOC     *       <LOADCOUNT>-LOAD COUNTER
648  !ENABLE LOAD OF COUNTER FROM B-BUS <7:0>. ONLY IF MOD/LOADREG.
649  .FIELD   LOADCOUNT::=<16>
650           NO::=0                  !NOP
651           YES::=1                 !LOAD COUNTER AT P2 T[100 NS].
652
653  !-----------------------------------------------------------------------
654
```

```
655
656
657  .TOC     *   SEQUENCING FIELD
658
659
660
661  .TOC     *     <UBF>-BUT MICROBRANCH FIELD
662  !SPECIFIES CONDITIONS TO MODIFY <UPF>/<J> FIELD DURING BRANCH.
663  !THIS IS ALWAYS IN EFFECT.
664  .FIELD   UBF::=<13:9>,30
665
666  .TOC     *       NO BUT
667           NULL::=30                          !SPECIFY NO MODIFICATION
668                                              !(DEFAULT CONDITON).
669
670  .TOC     *       ACTIVE ONLY
671  !PURELY ACTIVE BUTS GENERATE SIDE AFFECTS; THEY DO NOT MODIFY THE <UPF>
672  !FIELD BY THE "OR-ING"-IN-OF-CONDITIONS METHOD.   THEY MAY MODIFY
673  !EXPLICITLY THE ENTIRE <UPF> FIELD,  AS IN BUT(RETURN)
674           R-OR-1::=22                       !FORM R[SF]-IOR-"001
675           CUA-TRACK::=31                    !RESUME/RESTART CUA TRACKING
676           CLR-FLAG-RES-UCON::=32            !CLEAR FLAGS<2:0>, EX-FLAG<1>,
677                                             !RES-REGISTER, UCON-REGISTER
678           DIAGNOSE::=33                     !SPECIAL DIAGNOSTIC BUT
679           SUBRB::=34                        !RETURN <- EMIT<14:03>,
680                                             !PAGE <- EMIT<02:00>
681             SUBR-B::=34                     !SYNONYMS ARE:
682             GOTO::=34                       !
683             GO-TO::=34                      !
684           SUBRA::=35                        !RETURN <- D<14:03>,
685                                             !PAGE <- EMIT<02:00>
686             SUBR-A::=35                     !SYNONYM
687           B#36::=36                         !TBD
688           RETURN::=37                       !PAGE <- RETURN<11:09>,
689                                             !NUA <- RETURN<08:00>
690
691  .TOC     *       INACTIVE ONLY
692  !INACTIVE BUTS ONLY CAUSE MODIFICATION OF THE <UPF> FIELD BY THE
693  !"OR-ING"-IN-OF-CONDITIONS METHOD.
694                                             !----UPF MASK-----
695                                             !876 543 210 OCTAL
696                                             !*=NOT AFFECTED
697           SR3-0::=00                        !*** **0 000 (000)
698             CASE::=00                       !
699             SR03::=00                       !*** **0 111 (007)
700             SR02::=00                       !*** **1 011 (013)
701             SR01::=00                       !*** **1 101 (015)
702             SR00::=00                       !*** **1 110 (016)
703           IR15-12::=01                      !*** **0 000 (000)
704             DOP::=01                        !
705           INSTR5::=02                       !*** *00 000 (000)
706             INSTR-5::=02                    !
707           IR11#FLTPT3-0::=03                !*** *00 000 (000)
708             IR11-A::=03                     !*** *01 111 (017)
709           IR9-6::=04                        !*** **0 000 (000)
710             SOP::=04                        !
```

```
711           MOV-DR7#IR5-3::=05                !*** **0 000 (000)
712             MOV-DR7::=05                    !*** **0 111 (007)
713             IR5-3::=05                      !*** **1 000 (010)
714           BGSERV-FPSERV#D[C]#FPRET::=07     !*** **0 000 (000)
715             BGSERV-FPSERV::=07              !*** **0 111 (007)
716             D[C]-C::=07                     !*** **1 011 (013)
717             FPRET1-0::=07                   !*** **1 100 (014)
718           COUT07#DOUT07#FPS05::=10          !*** *** 000 (000)
719             COUT07::=10                     !*** *** 011 (003)
720             DOUT07::=10                     !*** *** 101 (005)
721             COUT07#DOUT07::=10              !*** *** 001 (001)
722             FPS05::=10                      !*** *** 110 (006)
723           DM0#SM0#BYTE::=11                 !*** *** 000 (000)
724             DM0::=11                        !*** *** 011 (003)
725             SM0::=11                        !*** *** 101 (005)
726             BYTE::=11                       !*** *** 110 (006)
727           GD3-2::=12                        !*** *** *00 (000)
728           BG-SERVCE-L#MFSS#MULTIPLE::=14    !*** *** 000 (000)
729             BG-SERVCE-L::=14                !*** *** 011 (003)
730             MFSS::=14                       !*** *** 101 (005)
731             MULTIPLE::=14                   !*** *** 110 (006)
732               MASKED-PS[T]::=14             !
733               D00::=14                      !
734               PS[N]::=14                    !
735               FLAG7::=14                    !
736               EXFLAG1::=14                  !
737               FLTPTS::=14                   !
738               EXFLAG2::=14                  !
739               INIT-JAM::=14                 !
740           D14-00EQ0#D15::=15                !*** *** *00 (000)
741             D14-00-EQ-0#D15::=15            !
742             D14-00-EQ-0::=15                !*** *** *01 (001)
743             D15::=15                        !*** *** *10 (002)
744           IR11#PS15::=16                    !*** *** *00 (000)
745             IR11-B::=16                     !*** *** *01 (001)
746             PS15::=16                       !*** *** *10 (002)
747           VECTOR-LOAD#DR6-7L::=21           !*** *** *00 (000)
748             VECTOR-LOAD::=21                !*** *** *01 (001)
749             DR6-7L::=21                     !*** *** *10 (002)
750           D[C]#BA00::=23                    !*** *** *00 (000)
751             D[C]-B::=23                     !*** *** *01 (001)
752             BA00::=23                       !*** *** *10 (002)
753           OTHER-JAM#FP-PROC::=24            !*** *** *00 (000)
754             OTHER-JAM::=24                  !*** *** *01 (001)
755             FP-PROC::=24                    !*** *** *10 (002)
756           INTR-HIGH#INSTR-BRANCH-L::=26     !*** *** *00 (000)
757             INTR-HIGH::=26                  !*** *** *01 (001)
758             INSTR-BRANCH-L::=26             !*** *** *10 (002)
759           PREFETCH-JAM#FP-FD::=27           !*** *** *00 (000)
760             PREFETCH-JAM::=27               !*** *** *01 (001)
761             FP-FD::=27                      !*** *** *10 (002)
762
```

```
763  .TOC     *       BOTH ACTIVE AND INACTIVE
764  !THESE BUTS HAVE BOTH ACTIVE AND INACTIVE EFFECTS
765                                          !----UPF MASK-----
766                                          !876 543 210 OCTAL
767                                          !*=NOT AFFECTED
768           INSTR1::=06                    !*00 000 000 (000)  BUS CONTROL,
769                                          !SP REWRITE DEFEAT
770             INSTR-1::=06                 !
771           SR1-0#COUNT-IS-377::=13        !*** *** 000 (000)  BUMP COUNTER
772             SR1-0::=13                   !*** *** 001 (001)  BUMP COUNTER
773             COUNT-IS-377-A::=13          !*** *** 110 (006)  BUMP COUNTER
774           COUNT-IS-377#D[C]::=17         !*** *** *00 (000)  BUMP COUNTER
775             COUNT-IS-377-B::=17          !*** *** *01 (001)  BUMP COUNTER
776             D[C]-A::=17                  !*** *** *10 (002)  BUMP COUNTER
777           COUNT-IS-377::=25              !*** *** **0 (000)  BUMP COUNTER
778           PREFETCH-L#SERVICE::=20        !*** *** *00 (000)  TIMING
779             PREFETCH-L::=20              !*** *** *01 (001)  TIMING
780             SERVICE::=20                 !*** *** *10 (002)  TIMING
781             LAST::=20                    !*** *** *11 (003)  TIMING
782
783
784
785
786  .TOC     *     <UPF>-MICRO POINTER FIELD
787  !SPECIFIES EITHER NEXT MICROINSTRUCTION ADDRESS OR BASE TARGET
788  !ADDRESS TO BE USED "UNDER" THE BUT-CODE IN <UBF>.
789  .FIELD           UPF::=<8:0>,000       !ACTUAL MICROWORD POINTER FIELD
790  .ADDRESS         J::=<8:0>             !THIS FIELD ALSO HAS
791                                         !MICROADDRESS QUALITIES
792
793           !BASE MACHINE MICROCODE ENTRY POINTS:
794
795           !THESE ENTRY POINTS HAVE BEEN FIXED AS OF 31-AUGUST-1976.
796           INIT01  ::=  0412       !INITIALIZATION SUBROUTINE (3412).
797           CON99   ::=  0040       !FORCE "CONSOLE-MODE HALT" (1040).
798           FET01   ::=  0702       !INSTR FETCH, NO OVERLAP
799           FET03   ::=  0700       !INSTR FETCH, OVERLAP
800           SER01   ::=  0701       !SERVICE ENTRY, OVERLAP
801           SER02   ::=  0703       !SERVICE ENTRY, NO OVERLAP
802           TRP00   ::=  0127       !
803           TRP07   ::=  0631       ! (4631).
804           BRA05   ::=  0003       !CHECK SERVICE WITH FET01 AS TARGET.
805           EOS1A   ::=  0460       !END OF SERVICE ROUTINE.  (1460).
806
807
808  !-------------------------------------------------------------------------
809
810
```

```
811
812 .TOC     *   MISCELLANEOUS FIELDS
813
814
815
816 .TOC     *     <NEXT-PAGE>-NEW PAGE ADDRESS LOADED DURING BUT[SUBROUTINE]
817 !THESE 3 BITS ARE CLOCKED INTO PAGE REGISTER DURING A BUT[SUBRA] OR
818 !BUT[SUBRB].  ONLY USED WHEN UBF/BUT[SUBRA] OR UBF/BUT[SUBRB].
819 .FIELD   NEXT-PAGE::=<32:30>
820
821
822 .TOC     *     <MULTIPLE>-SELECT CODE FOR BUT[MULTIPLE]
823 !MUST BE SET IN BOTH PREVIOUS AND CURRENT MICROWORDS WHEN BUT[MULTIPLE]
824 !IS TO BE EMPLOYED.
825 .FIELD   MULTIPLE::=<32:30>
826          MASKED-PS[T]::=0        !
827          D00::=1                 !
828          PS[N]::=2               !
829          FLAG7::=3               !
830          EXFLAG1::=4             !
831          FLTPTS::=5              !
832          EXFLAG2::=6             !
833          INIT-JAM::=7            !
834
835
836
837 .TOC     *     EMIT FIELD - IMMEDIATE DATA FROM MICROWORD
838 !USED WHENEVER LOADING IMMEDIATE DATA FROM MICROWORD
839 .FIELD   EMIT::=<47:44>'<41:30>
840 .FIELD   EMITH::=<47:44>
841 .FIELD   EMITM::=<41:38>
842 .FIELD   EMITL::=<37:30>
843 .FIELD   EMITML::=<41:30>
844 .FIELD   EMIT9-6::=<39:36>
845 .FIELD   EMIT15::=<47>
846 .FIELD   EMIT14::=<46>
847 .FIELD   EMIT13::=<45>
848 .FIELD   EMIT12::=<44>
849 .FIELD   EMIT11::=<41>
850 .FIELD   EMIT10::=<40>
851 .FIELD   EMIT09::=<39>
852 .FIELD   EMIT08::=<38>
853 .FIELD   EMIT07::=<37>
854 .FIELD   EMIT06::=<36>
855 .FIELD   EMIT05::=<35>
856 .FIELD   EMIT04::=<34>
857 .FIELD   EMIT03::=<33>
858 .FIELD   EMIT02::=<32>
859 .FIELD   EMIT01::=<31>
860 .FIELD   EMIT00::=<30>
861
862
```

```
863
864  .TOC     *     RETURN ADDRESS - FOR MICROSUBROUTINE CALLS
865  !USED WITH BUT[SUBRB] AND BUT[SUBRA]
866  .FIELD   RETURN::=<46:44>'<41:33>        !PAGE # D.I.P.
867
868
869
870  .TOC     *     UCON SELECTION AND CONTROL FIELDS
871
872
873  .TOC     *         SELECTION
874  .FIELD   UCON-SEL-I-O::=<46>             !SELECT I-O [BUS] CONTROL
875           NO::=0
876           YES::=1
877  .FIELD   UCON-SEL-WCS::=<45>             !SELECT WCS/ECS/DCS
878           NO::=0
879           YES::=1
880  .FIELD   UCON-SEL-CACHEKT::=<44>         !SELECT CACHE/KT
881           NO::=0
882           YES::=1
883  .FIELD   UCON-SEL-PROC::=<36>            !SELECT PROCESSOR CONTROL
884           NO::=0
885           YES::=1
886  .FIELD   UCON-SEL-FLTPT::=<33>           !SELECT HOT FLOATING POINT
887           NO::=0
888           YES::=1
889
890
```

```
891
892  .TOC     *          CONTROL (ALSO TMS ROUTINES)
893  !AFTER UCON[S] SELECTED FROM ABOVE, CONTROL COMES FROM HERE.
894  .FIELD   UCON::=<32:30>'<35:34>'<47>'<42:38>
895  !
896  !WHEN INVOKING TMS ROUTINES TO TALK TO LOCAL STORE THEN USE THE
897  !FOLLOWING NAMES AS FORMAL PARAMETERS IN THE SUBSTITUTION MACRO
898  !        TMSPTR_(XX).   THESE ARE THE ONLY LEGAL VALUES THAT WILL
899  !WORK IN THE MACRO.  FURTHER EXPLANATION OF THESE ROUTINES
900  !CAN BE OBTAINED IN THE LISTING OF THE TMS ROM.
901  !
902
903           READ             ::=0064 !READ DATA.
904           READANDINC       ::=0050 !READ DATA TO MD, INCREMENT ADDR.
905           LOADANDREAD      ::=0040 !LOAD ADDRESS AND THEN READ DATA.
906           LOADREADINC      ::=0070 !LOAD ADDRESS,READ DATA,
907                                    ! INCREMENT ADDRESS.
908           WRITE            ::=0030 !WRITE DATA.
909           WRITEANDINC      ::=0010 !WRITE DATA AND THEN INCREMENT ADDRESS.
910           LOADANDWRITE     ::=0020 !LOAD ADDRESS AND THEN WRITE DATA
911           LOADWRITEINC     ::=0002 !LOAD ADDRESS,WRITE DATA,
912                                    ! INCREMENT ADDRESS.
913           INCANDREAD       ::=0012 !INCREMENT ADDRESS AND THEN READ DATA.
914           LOADADDRESS      ::=0100 !LOAD ADDRESS
915
916           LOADGRS          ::=0104 !LOAD GR'S FROM
917                                    ! LOCAL STORE
918           STOREGRS         ::=0140 !SAVE GR'S INTO
919                                    ! LOCAL STORE
920           LOADFP           ::=0174 !LOAD FP REGISTERS FROM LOCAL STORE
921           STOREFP          ::=0266 !SAVE FP REGISTERS INTO LOCAL STORE
922           LOADCSP          ::=0360 !LOAD CSP[00-13] INTO LOCAL STORE.
923           STORECSP         ::=0420 !SAVE CSP[00-13] INTO LOCAL STORE.
924           LOADWCSAB        ::=0462 !LOAD WCS WORK REGISTERS FROM LOCAL STORE
925           STOREWCSAB       ::=0502 !SAVE WCS WORK REGISTERS INTO LOCAL STORE.
926           SETLOAD          ::=0522 !SAME AS LOADREADINC.
927           SETSTORE         ::=0530 !SAME AS LOAD ADDRESS.
928           ASPADLOAD        ::=0534 !LOAD ASP[00-37] FROM LOCAL STORE
929           ASPADSTORE       ::=0646 !SAVE ASP[00-37] INTO LOCAL STORE
930           BSPADLOAD        ::=0756 !LOAD BSP[00-37] FROM LOCAL STORE.
931           BSPADSTORE       ::=1070 !SAVE BSP[00-37] INTO LOCAL STORE.
932           ALLCSPLOAD       ::=1202 !LOAD CSP[00-17] FROM LOCAL STORE.
933           ALLCSPSTORE      ::=1252 !SAVE CSP[00-17] INTO LOCAL STORE.
934           LOADREADTWO      ::=1324 !LOAD ADDRESS AND READ TWO DATA ITEMS
935           INCREADTWO       ::=1334 !INCREMENT ADDRESS AND READ
936                                    ! TWO DATA ITEMS
937           LOADWRITETWO     ::=1342 !LOAD ADDRESS AND WRITE TWO DATA ITEMS.
938           WRITETWO         ::=1352 !INCREMENT ADDRESS AND WRITE
939                                    ! TWO DATA ITEMS.
```

```
940           READINDIRECT    ::=1362 !READ DATA ITEM INDIRECTLY.
941           WRITEINDIRECT   ::=1376 !WRITE DATA ITEM INDIRECTLY.
942           LOADFP01        ::=1412 !LOAD FP0 AND FP1 FROM LOCAL STORE.
943           LOADFP23        ::=1444 !LOAD FP2 AND FP3 FROM LOCAL STORE.
944           LOADFP45        ::=1476 !LOAD FP4 AND FP5 FROM LOCAL STORE.
945           STOREFP01       ::=1530 !SAVE FP0 AND FP1 INTO LOCAL STORE.
946           STOREFP23       ::=1562 !SAVE FP2 AND FP3 INTO LOCAL STORE.
947           STOREFP45       ::=1614 !SAVE FP4 AND FP5 INTO LOCAL STORE.
948           LOADTEMP        ::=1646 !LOAD TEMPS FROM LOCAL STORE.
949           STORETEMP       ::=1700 !SAVE TEMPS INTO LOCAL STORE.
950
951   .FIELD   UCONH::=<32:30>
952   .FIELD   UCONM::=<35:34>
953   .FIELD   UCONL::=<47>'<42:38>
954   .FIELD   UCON15::=<32>
955   .FIELD   UCON14::=<31>
956   .FIELD   UCON13::=<30>
957   .FIELD   UCON12::=<35>
958   .FIELD   UCON11::=<34>
959   .FIELD   UCON10::=<47>
960   .FIELD   UCON09::=<42>
961   .FIELD   UCON08::=<41>
962   .FIELD   UCON07::=<40>
963   .FIELD   UCON06::=<39>
964   .FIELD   UCON05::=<38>
965
966
967   .TOC     *     LOCAL STORE FIELDS
968   !
969   !EACH 48 BIT WORD IS DIVIDED INTO 3 SIXTEEN BIT FIELDS.
970   !BITS <15-00> ARE LSADR'S 0000-1777.  (COLUMN ZERO)
971   !BITS <31-16> ARE LSADR'S 2000-3777.  (COLUMN ONE)
972   !BITS <47-32> ARE LSADR'S 4000-5777.  (COLUMN TWO)
973
974   .FIELD   COLTWO::=<47:32>
975   .FIELD   COLONE::=<31:16>
976   .FIELD   COLZERO::=<15:00>
977
978   !-----------------------------------------------------------------------
979
980
981
982   !-----------------------------------------------------------------------
983   !END OF MICROWORD FIELD DEFINITIONS.
984   !-----------------------------------------------------------------------
985
986
```

```
 987
 988  !=============================================================================
 989
 990  .TOC     * MACRO DEFINITIONS
 991
 992
 993  .TOC     *   PRIMITIVE OPERATIONS
 994
 995
 996  .TOC     *     TIMING
 997  .MACRO   P0      ::=  NULL       !0 NS., UP3 VIEWED AS THE START OF A
 998                                    !MICROCYCLE
 999
1000  .MACRO   P1      ::=  NULL       !60 NS., AT P1
1001  .MACRO   P1-L    ::=  NULL       !30 NS., AT P1 LEADING EDGE
1002  .MACRO   P1-T    ::=  NULL       !60 NS., AT P1 TRAILING EDGE
1003
1004  .MACRO   P2      ::=  NULL       !100 NS., AT P2
1005  .MACRO   P2-L    ::=  NULL       !70 NS., AT P2 LEADING EDGE
1006  .MACRO   P2-T    ::=  WHEN/AT-P2-T  !100 NS., AT P2 TRAILING EDGE
1007  .MACRO   P2-U    ::=  NULL       !UNSUPPRESSED P2, CLOCK CONTINUOUSLY
1008
1009  .MACRO   P3      ::=  NULL       !150 NS., 120-150 NS., AT P3
1010  .MACRO   P3-L    ::=  NULL       !120 NS., AT P3 LEADING EDGE
1011  .MACRO   P3-T    ::=  WHEN/AT-P3-T  !150 NS., AT P3 TRAILING EDGE
1012  .MACRO   P3-U    ::=  NULL       !UNSUPPRESSED P3, CLOCK CONTINUOUSLY
1013
1014  .MACRO   UP3     ::=  NULL       !P3 DELAYED BY 5 NS., P0 VIEWED AS THE
1015                                    !END OF A MICROCYCLE.  LATCHES NEW
1016                                    !MICROINSTRUCTION INTO THE
1017                                    !MICROWORD BUFFER REGISTER.
1018
1019  .MACRO   DEFER   ::=  NULL       !CONTROL IS ISSUED AT THIS TIME,
1020                                    !  ANY REQUIRED CLOCKING OCCURS LATER
1021  .MACRO   NEXT    ::=  NULL       !WHERE TO GO NEXT, CLOCKED AT UP3
1022  .MACRO   SETUP   ::=  NULL       !SETUP DATA/CONTROL
1023  .MACRO   SELECT  ::=  NULL       !MAKE A HOT-BOX SELECTION
1024  .MACRO   ISSUE   ::=  NULL       !SET/CLEAR HOT-BOX FLAG
1025  .MACRO   ENABLE  ::=  NULL       !  DITTO
1026  .MACRO   EMITC   ::=  NULL       !SPECIFY AN EMIT-CONSTANT VALUE
1027
1028
1029
1030  !=============================================================================
1031
```

```
1032  .TOC     *     WRITING THE A AND B SCRATCH PADS
1033  !
1034  !  WRITING THE APPROPRIATE SCRATCH PADS:
1035  !
1036  !                                          (NOP       )
1037  !                                          (A   L     A)
1038  !                                       WR (B , H   , B)
1039  !                                          (AB        )
1040  !                                         /!\ /!\  /!\
1041  !                                         !   !    !
1042  !        ASP, BSP, BOTH, NEITHER----------   !    !
1043  !        LO[00-17], OR HI[20-37]--------------    !
1044  !        USE "A" SIDE OR "B" SIDE ADDRESS----------
1045  !
1046  !  WRITES CONTENTS OF D-REGISTER INTO ADDRESSED SCRATCH PADS [SEE
1047  !        BELOW] DURING P3
1048  !
1049  .MACRO   WR(AB,HL,ADDR)  ::=  MOD/CLKSP,          !CLOCK SP MODE
1050                                   WRSP/@AB,        !NOP, A, ASP, B, BSP,
1051                                                    !AB, ABSP, BA, BASP,
1052                                                    !BOTH ARE CHOICES
1053                                   HILO/@HL,        !HI, LO, H, L ARE CHOICES
1054                                   WRSEL/@ADDR      !A, B, A-ADDR, B-ADDR
1055                                                    !ARE CHOICES.
1056
1057
1058
1059  .TOC     *     ASP AND BSP PHYSICAL REGISTER ADDRESSES
1060  !
1061  !  ENABLE INPUT/OUTPUT [FOR READ AND/OR WRITE] OF THE APPROPRIATE
1062  !  SCRATCH PAD ONTO EITHER BUS-A OR BUS-B VIA EXACT PHYSICAL ADDRESS
1063  !
1064  .MACRO   ASPLO(XX)       ::=  AEN/ASPLO,          !SELECT
1065                                   ASEL/@XX,        !REGISTER &
1066                                   RIF/@XX          !ENABLE ON BUS-A
1067
1068  .MACRO   ASPHI(XX)       ::=  AEN/ASPHI,          !SELECT
1069                                   ASEL/@XX,        !REGISTER &
1070                                   RIF/@XX          !ENABLE ON BUS-A
1071
1072  .MACRO   ASP(XX)         ::=  ASEL/@XX,           !SELECT REGISTER,
1073                                   RIF/@XX          !NO ENABLE
1074
1075
1076  .MACRO   BSPLO(XX)       ::=  BEN/BSPLO,          !SELECT
1077                                   BSEL/@XX,        !REGISTER &
1078                                   RIF/@XX          !ENABLE ON BUS-B
1079
1080  .MACRO   BSPHI(XX)       ::=  BEN/BSPHI,          !SELECT
1081                                   BSEL/@XX,        !REGISTER &
1082                                   RIF/@XX          !ENABLE ON BUS-B
1083
1084  .MACRO   BSP(XX)         ::=  BSEL/@XX,           !SELECT REGISTER,
1085                                   RIF/@XX          !NO ENABLE
1086
1087
```

```
1088
1089
1090
1091  .TOC     *     ASP AND BSP BASE MACHINE FUNCTIONAL REGISTER ADDRESSES
1092  !
1093  !  ENABLE INPUT/OUTPUT [FOR READ AND/OR WRITE] OF THE APPROPRIATE
1094  !  SCRATCH PAD ONTO EITHER BUS-A "-A" OR BUS-B "-B" VIA FUNCTIONAL
1095  !  REGISTER DESIGNATION
1096  !
1097  .MACRO   R0-A            ::=  ASPLO(R00)
1098  .MACRO   R0-B            ::=  BSPLO(R00)
1099  .MACRO   R1-A            ::=  ASPLO(R01)
1100  .MACRO   R1-B            ::=  BSPLO(R01)
1101  .MACRO   R2-A            ::=  ASPLO(R02)
1102  .MACRO   R2-B            ::=  BSPLO(R02)
1103  .MACRO   R3-A            ::=  ASPLO(R03)
1104  .MACRO   R3-B            ::=  BSPLO(R03)
1105  .MACRO   R4-A            ::=  ASPLO(R04)
1106  .MACRO   R4-B            ::=  BSPLO(R04)
1107  .MACRO   R5-A            ::=  ASPLO(R05)
1108  .MACRO   R5-B            ::=  BSPLO(R05)
1109  .MACRO   SP-A            ::=  ASPLO(R06)
1110  .MACRO   SP-B            ::=  BSPLO(R06)
1111  .MACRO   PC-A            ::=  ASPLO(R07)
1112  .MACRO   PC-B            ::=  BSPLO(R07)
1113  .MACRO   FACA[0]-B       ::=  BSPHI(R10)
1114  .MACRO   FACB[0]-A       ::=  ASPHI(R10)
1115  .MACRO   FACC[0]-B       ::=  BSPLO(R10)
1116  .MACRO   FACD[0]-A       ::=  ASPLO(R10)
1117  .MACRO   FACA[1]-B       ::=  BSPHI(R11)
1118  .MACRO   FACB[1]-A       ::=  ASPHI(R11)
1119  .MACRO   FACC[1]-B       ::=  BSPLO(R11)
1120  .MACRO   FACD[1]-A       ::=  ASPLO(R11)
1121  .MACRO   FACA[2]-B       ::=  BSPHI(R12)
1122  .MACRO   FACB[2]-A       ::=  ASPHI(R12)
1123  .MACRO   FACC[2]-B       ::=  BSPLO(R12)
1124  .MACRO   FACD[2]-A       ::=  ASPLO(R12)
1125  .MACRO   FACA[3]-B       ::=  BSPHI(R13)
1126  .MACRO   FACB[3]-A       ::=  ASPHI(R13)
1127  .MACRO   FACC[3]-B       ::=  BSPLO(R13)
1128  .MACRO   FACD[3]-A       ::=  ASPLO(R13)
1129  .MACRO   FACA[4]-B       ::=  BSPHI(R14)
1130  .MACRO   FACB[4]-A       ::=  ASPHI(R14)
1131  .MACRO   FACC[4]-B       ::=  BSPLO(R14)
1132  .MACRO   FACD[4]-A       ::=  ASPLO(R14)
1133  .MACRO   FACA[5]-B       ::=  BSPHI(R15)
1134  .MACRO   FACB[5]-A       ::=  ASPHI(R15)
1135  .MACRO   FACC[5]-B       ::=  BSPLO(R15)
1136  .MACRO   FACD[5]-A       ::=  ASPLO(R15)
1137  .MACRO   FDSTA-B         ::=  BSPHI(R17)
1138  .MACRO   FDSTB-A         ::=  ASPHI(R17)
1139  .MACRO   FDSTC-B         ::=  BSPLO(R17)
1140  .MACRO   FDSTD-A         ::=  ASPLO(R17)
1141  .MACRO   FPSHI#FEC-A     ::=  ASPHI(R16)
1142  .MACRO   FEA-B           ::=  BSPHI(R16)
1143  .MACRO   USER-SP-A       ::=  ASPLO(R16)
```

```
1144  .MACRO   USER-SP-B       ::=  BSPLO(R16)
1145  .MACRO   WHAMI-A         ::=  ASPHI(R02)
1146  .MACRO   R[ZERO]-B       ::=  BSPHI(R03)
1147  .MACRO   R[IR]-A         ::=  ASPHI(R17)
1148  .MACRO   R[SRC]-B        ::=  BSPHI(R04)
1149  .MACRO   R[SRC]-A        ::=  ASPHI(R04)
1150  .MACRO   R[DST]-B        ::=  BSPHI(R05)
1151  .MACRO   R[DST]-A        ::=  ASPHI(R05)
1152  .MACRO   R[VECT]-B       ::=  BSPHI(R02)
1153  .MACRO   WCSB[0]-B       ::=  BSPHI(R00)
1154  .MACRO   WCSB[1]-B       ::=  BSPHI(R01)
1155  .MACRO   WCSA[0]-A       ::=  ASPHI(R00)
1156  .MACRO   WCSADR          ::=  ASPHI(R01)
1157  .MACRO   FPA-B           ::=  BSPHI(R06)
1158  .MACRO   CNSL-CNTL-B     ::=  BSPHI(R07)
1159  .MACRO   CNSL-CADR-A     ::=  ASPHI(R07)
1160  .MACRO   CNSL-SW-A       ::=  ASPHI(R06)
1161  .MACRO   CNSL-TMPSW-A    ::=  ASPHI(R03)
1162
1163
1164
1165  .TOC     *     ASP AND BSP INDIRECT REGISTER ADDRESSES
1166  !
1167  !  ENABLE INPUT/OUTPUT [FOR READ AND/OR WRITE] OF THE APPROPRIATE
1168  !   SCRATCH PAD ON BUS-A [A] OR BUS-B USING INDIRECT ADDRESSING
1169  !   WITH THE IR,  WHERE:
1170  !
1171  !        SF<3:0>H = [FLPADR H + KTSRCADRS3 H] # [FLTPT L * IR8 H] #
1172  !                           [IR7 H] # [IR6 H + RORl H]
1173  !
1174  !        DF<3:0>H = [FLPADR H + KTDSTADRS3 H] # [IR2 H] # [IRl H] # [IR0 H]
1175  !
1176  .MACRO   R[SF]-LO-A      ::=  AEN/ASPLO,ASEL/SF
1177  .MACRO   R[SF]-LO-B      ::=  BEN/BSPLO,BSEL/SF
1178  .MACRO   R[SF]-HI-A      ::=  AEN/ASPHI,ASEL/SF
1179  .MACRO   R[SF]-HI-B      ::=  BEN/BSPHI,BSEL/SF
1180  .MACRO   R[DF]-LO-A      ::=  AEN/ASPLO,ASEL/DF
1181  .MACRO   R[DF]-LO-B      ::=  BEN/BSPLO,BSEL/DF
1182  .MACRO   R[DF]-HI-A      ::=  AEN/ASPHI,ASEL/DF
1183  .MACRO   R[DF]-HI-B      ::=  BEN/BSPHI,BSEL/DF
1184  .MACRO   R[SF]-A         ::=  R[SF]-LO-A
1185  .MACRO   R[SF]-B         ::=  R[SF]-LO-B
1186  .MACRO   R[DF]-A         ::=  R[DF]-LO-A
1187  .MACRO   R[DF]-B         ::=  R[DF]-LO-B
1188
1189
1190
1191  .TOC     *     ASP, BSP INDIRECT ADDRESSING
1192  !
1193  !        THESE MACROS ONLY SELECT THE ADDRESS MODE FOR THE ASP AND BSP;
1194  !        THE SELECTED SP IS NOT ENABLED ONTO THE BUS
1195  !
1196  .MACRO   ASP-ADDRS-R[DF] ::=  ASEL/DF
1197  .MACRO   ASP-ADDRS-R[SF] ::=  ASEL/SF
1198  .MACRO   BSP-ADDRS-R[DF] ::=  BSEL/DF
1199  .MACRO   BSP-ADDRS-R[SF] ::=  BSEL/SF
```

```
1200
1201
1202
1203
1204  !=====================================================================
1205
1206  .TOC     *     WRITING THE C SCRATCH PAD
1207  !
1208  !  WRITE DATA ON BUSDIN [ACTUALLY DMUX OUTPUT] INTO ADDRESSED CSP
1209  !  LOCATION   [SEE BELOW] DURING P3
1210  !
1211  .MACRO   WR-CSP          ::=  WRCSP/YES
1212
1213
1214
1215  .TOC     *     CSP IMPLIED ADDRESSING
1216  !
1217  !  ENABLE FOR INPUT/OUTPUT [READ AND/OR WRITE] ONTO BUS-B ONLY A
1218  !  SPECIFIC CSP LOCATION,  WHERE THE ADDRESS IS DETERMINED
1219  !  AS FOLLOWS:
1220  !
1221  !        CSPADDR<3:0>H = -[ 0 # 0 # BSEL<1>H # BSEL<0>H ]
1222  !
1223  .MACRO   CSPB(XX)        ::=  BEN/BASCON,        !USE IMMEDIATE MODE
1224                                   BSEL/@XX        !WHICH ONE
1225
1226
1227
1228  .TOC     *     CSP DIRECT ADDRESSING
1229  !
1230  !  ENABLE FOR INPUT/OUTPUT [READ AND/OR WRITE] ONTO BUS-B ONLY A
1231  !  SPECIFIC CSP LOCATION,  WHERE THE ADDRESS IS DETERMINED
1232  !  AS FOLLOWS:
1233  !
1234  !        CSPADDR<3:0>H = -UWORD<23:20> H
1235  !
1236  .MACRO   CSPD(XX)        ::=  BEN/CSP,           !USE CSP-ADDR MODE
1237                                   CSPADDR/@XX     !WHICH ONE
1238
1239
```

```
1240
1241  !=====================================================================
1242
1243  .TOC     *     SHIFT TREE SPECIFICATION
1244  !N.B. MAY REQUIRE PRIOR SETUP OF RES-REGISTER FOR SHIFT END MUX
1245  !  SELECTION CONTROL  (E.G., WHEN ASEL/LEFT-A IS USED).
1246
1247  .TOC     *       ENABLED ONTO BUS A
1248  .MACRO   D-RIGHT-14              ::=  AEN/CMUX,AMUX/RIGHT-8,
1249                                           BMUX/RIGHT-4,ASEL/RIGHT-2
1250  .MACRO   D-RIGHT-13              ::=  AEN/CMUX,AMUX/RIGHT-8,
1251                                           BMUX/RIGHT-4,ASEL/RIGHT-1
1252  .MACRO   D-RIGHT-12              ::=  AEN/CMUX,AMUX/RIGHT-8,
1253                                           BMUX/RIGHT-4,ASEL/DIRECT
1254  .MACRO   D-RIGHT-11              ::=  AEN/CMUX,AMUX/RIGHT-8,
1255                                           BMUX/RIGHT-4,ASEL/LEFT-1
1256                                           !SENDMUX SETUP
1257  .MACRO   D-RIGHT-10              ::=  AEN/CMUX,AMUX/RIGHT-8,
1258                                           BMUX/DIRECT,ASEL/RIGHT-2
1259  .MACRO   D-RIGHT-9               ::=  AEN/CMUX,AMUX/RIGHT-8,
1260                                           BMUX/DIRECT,ASEL/RIGHT-1
1261  .MACRO   D-RIGHT-8               ::=  AEN/CMUX,AMUX/RIGHT-8,
1262                                           BMUX/DIRECT,ASEL/DIRECT
1263  .MACRO   D-RIGHT-7               ::=  AEN/CMUX,AMUX/RIGHT-8,
1264                                           BMUX/DIRECT,ASEL/LEFT-1
1265                                           !SENDMUX SETUP
1266  .MACRO   D-RIGHT-6               ::=  AEN/CMUX,AMUX/DIRECT,
1267                                           BMUX/RIGHT-4,ASEL/RIGHT-2
1268  .MACRO   D-RIGHT-5               ::=  AEN/CMUX,AMUX/DIRECT,
1269                                           BMUX/RIGHT-4,ASEL/RIGHT-1
1270  .MACRO   D-RIGHT-4               ::=  AEN/CMUX,AMUX/DIRECT,
1271                                           BMUX/RIGHT-4,ASEL/DIRECT
1272  .MACRO   D-RIGHT-3               ::=  AEN/CMUX,AMUX/DIRECT,
1273                                           BMUX/RIGHT-4,ASEL/LEFT-1
1274                                           !SENDMUX SETUP
1275  .MACRO   D-RIGHT-2               ::=  AEN/CMUX,AMUX/DIRECT,
1276                                           BMUX/DIRECT,ASEL/RIGHT-2
1277  .MACRO   D-RIGHT-1               ::=  AEN/CMUX,AMUX/DIRECT,
1278                                           BMUX/DIRECT,ASEL/RIGHT-1
1279  .MACRO   D-NO-SHIFT              ::=  AEN/CMUX,AMUX/DIRECT,
1280                                           BMUX/DIRECT,ASEL/DIRECT
1281  .MACRO   D-DIRECT                ::=  D-NO-SHIFT
1282  .MACRO   D-LEFT-1                ::=  AEN/CMUX,AMUX/DIRECT,
1283                                           BMUX/DIRECT,ASEL/LEFT-1
1284                                           !SENDMUX SETUP
1285  .MACRO   D-SWAB                  ::=  AEN/CMUX,AMUX/SWAB,
1286                                           BMUX/DIRECT,ASEL/DIRECT
1287  .MACRO   D-SWAB-RIGHT-3          ::=  AEN/CMUX,AMUX/SWAB,
1288                                           BMUX/RIGHT-4,ASEL/LEFT-1
1289                                           !SENDMUX SETUP
1290  .MACRO   D-SWAB-LEFT-1           ::=  AEN/CMUX,AMUX/SWAB,
1291                                           BMUX/DIRECT,ASEL/LEFT-1
1292                                           !SENDMUX SETUP
1293  .MACRO   D-SIGNEXT               ::=  AEN/CMUX,AMUX/SIGNEXT,
1294                                           BMUX/DIRECT,ASEL/DIRECT
1295  .MACRO   D-SIGNEXT-RIGHT-1       ::=  AEN/CMUX,AMUX/SIGNEXT,
```

```
1296                                                BMUX/DIRECT,ASEL/RIGHT-1
1297  .MACRO   D-SIGNEXT-LEFT-1          ::=  AEN/CMUX,AMUX/SIGNEXT,
1298                                           BMUX/DIRECT,ASEL/LEFT-1
1299                                           !SENDMUX SETUP
1300  .MACRO   NO-SHIFT                  ::=  AEN/CMUX,
1301                                           BMUX/DIRECT,ASEL/DIRECT
1302  .MACRO   DIRECT                    ::=  NO-SHIFT
1303  .MACRO   COUNT#D[HI]               ::=  AEN/CMUX,AMUX/COUNTER#D[HI],
1304                                           BMUX/DIRECT,ASEL/DIRECT
1305  .MACRO   COUNT#D[LO]               ::=  AEN/CMUX,AMUX/COUNTER#D[LO],
1306                                           BMUX/DIRECT,ASEL/DIRECT
1307
1308
1309
1310  .TOC     *       FIRST TWO LEVELS ONLY [AMUX,
1311                                           BMUX]
1312  !N.B.:  FOR USE WHEN SHIFTING SR RIGHT, SR<15> <- BMUX<00>
1313  .MACRO   D-DIRECT[BMUX]            ::=  AMUX/DIRECT,
1314                                           BMUX/DIRECT
1315
1316
1317
1318  !=====================================================================
1319
1320  .TOC     *     ALU FUNCTIONS
1321  ![SEE FIELD DESCRIPTION OF "ALU" FOR FULL DESCRIPTION]
1322  .MACRO   ZERO                      ::=  ALU/ZERO
1323  .MACRO   A-XOR-B                   ::=  ALU/A-XOR-B
1324  .MACRO   B                         ::=  ALU/B
1325  .MACRO   A-AND-B                   ::=  ALU/A-AND-B
1326  .MACRO   A-IOR-B                   ::=  ALU/A-IOR-B
1327  .MACRO   A                         ::=  ALU/A
1328  .MACRO   NOT-A                     ::=  ALU/NOT-A
1329  .MACRO   NOT-A-AND-B               ::=  ALU/NOT-A-AND-B
1330  .MACRO   A-AND-NOT-B               ::=  ALU/A-AND-NOT-B
1331
1332  .MACRO   DIVIDE                    ::=  ALU/DIVIDE
1333  .MACRO   A-PLUS-B                  ::=  ALU/A-PLUS-B
1334  .MACRO   A-MINUS-B                 ::=  ALU/A-MINUS-B
1335  .MACRO   A-PLUS-B-PLUS-PS[C]       ::=  ALU/A-PLUS-B-PLUS-PS[C]
1336  .MACRO   A-PLUS-B-PLUS-D[C]        ::=  ALU/A-PLUS-B-PLUS-D[C]
1337  .MACRO   A-PLUS-NOT-B-PLUS-D[C]    ::=  ALU/A-PLUS-NOT-B-PLUS-D[C]
1338  .MACRO   A-PLUS-B-PLUS-1           ::=  ALU/A-PLUS-B-PLUS-1
1339
1340
1341  .TOC     *     COUT GENERATION
1342  ![SEE FIELD DESCRIPTION OF "COUT" FOR FULL DESCRIPTION]
1343  .MACRO   COUT_CIN                        ::=  COUT/CIN
1344  .MACRO   COUT_PS[C]                ::=  COUT/PS[C]
1345  .MACRO   COUT_ALU00                ::=  COUT/ALU00
1346  .MACRO   COUT_ALU07                ::=  COUT/ALU07
1347  .MACRO   COUT_ALU15                ::=  COUT/ALU15
1348  .MACRO   COUT_COUT07               ::=  COUT/COUT07
1349  .MACRO   COUT_COUT15               ::=  COUT/COUT15
1350  .MACRO   COUT_D[C]                 ::=  COUT/D[C]
1351
```

```
1352
1353
1354  !=============================================================================
1355
1356  .TOC     *      CLOCKS
1357
1358
1359
1360  .TOC     *        BASIC REGISTER CLOCKS [D, SR, BA, CC]
1361  .MACRO   CLK-D                    ::=  CLKD/YES
1362                                    !MUST SPECIFY P2 T OR P3 T
1363  .MACRO   CLK-SR                   ::=  CLKSR/YES
1364                                    !MUST SPECIFY P2 T OR P3 T
1365  .MACRO   CLK-BA                   ::=  CLKBA/YES
1366                                    !AT P1 T ONLY
1367  .MACRO   SET-CC                   ::=  SCC/YES
1368                                    !SETUP HERE, CLOCKED AT P2 T
1369                                    !**OF NEXT UWORD**  ONLY
1370  .MACRO   CLK-CC                   ::=  NULL
1371                                    !IN NEXT UWORD, FOR DOCUMENTATION
1372
1373
1374
1375  .TOC     *        REDEFINED FROM SP REWRITE FIELD [RES, COUNTER]
1376  .MACRO   LOAD-RES                 ::=  MOD/LOADREG,LOADRES/YES
1377                                    !AT P2 T ONLY, FROM B-BUS<14:11>
1378  .MACRO   LOAD-COUNTER             ::=  MOD/LOADREG,LOADCOUNT/YES
1379                                    !DURING ENTIRE UWORD, FROM B-BUS<7:0>
1380
1381
1382
1383  .TOC     *        RES REGISTER CONTROL VALUES [FROM EMIT]
1384
1385  !LOADED VIA:  EMIT<14:11> -> CSP[XX]<14:11> -> B-BUS<14:11> -> RES<3:0>
1386  .MACRO   SENDMUX-0123-SEL         ::=  EMIT14/1
1387                                    !FOR SHIFT TREE
1388  .MACRO   SENDMUX-4567-SEL         ::=  EMIT14/0
1389                                    !FOR SHIFT TREE
1390  .MACRO   SR-LOAD                  ::=  EMIT13/0,EMIT12/0
1391                                    !FOR SR/GUARD
1392  .MACRO   SR-LEFT                  ::=  EMIT13/0,EMIT12/1
1393                                    !FOR SR/GUARD
1394  .MACRO   SR-RIGHT                 ::=  EMIT13/1,EMIT12/0
1395                                    !FOR SR/GUARD
1396  .MACRO   SR-NOP                   ::=  EMIT13/1,EMIT12/1
1397                                    !FOR SR/GUARD
1398  .MACRO   GUARD-EN                 ::=  EMIT11/1
1399                                    !FOR SR/GUARD
1400  .MACRO   GUARD-DIS                ::=  EMIT11/0    !FOR SR/GUARD
1401
1402
1403
```

```
1404
1405  .TOC     *       CC CONTROL  [FROM EMIT]
1406  !USED VIA: BUS-U37-H -> EMIT07-H -> MODIFY-V(1)-H
1407  .MACRO   MODIFY-VBIT              ::=  EMIT07/1
1408  .MACRO   NOT-MODIFY-VBIT          ::=  EMIT07/0
1409
1410
1411
1412  !=====================================================================
1413
1414  .TOC     *     BUS CONTROL MACROS
1415  .MACRO   DATI-CLKIR       ::=  BEGIN/YES,SELECT/BUS,BUSCODE/DATI-CLKIR
1416  .MACRO   DATI-NOINT       ::=  BEGIN/YES,SELECT/BUS,BUSCODE/DATI-NOINT
1417  .MACRO   DATI             ::=  BEGIN/YES,SELECT/BUS,BUSCODE/DATI
1418  .MACRO   DATI[P]          ::=  BEGIN/YES,SELECT/BUS,BUSCODE/DATI[P]
1419                                    !WITH ALTER/ALLOWED
1420  .MACRO   DATO             ::=  BEGIN/YES,SELECT/BUS,BUSCODE/DATO
1421  .MACRO   DATIB            ::=  BEGIN/YES,SELECT/BUS,BUSCODE/DATIB
1422  .MACRO   DATIB[P]         ::=  BEGIN/YES,SELECT/BUS,BUSCODE/DATIB[P]
1423                                    !WITH ALTER/ALLOWED
1424  .MACRO   DATIP            ::=  BEGIN/YES,SELECT/BUS,BUSCODE/DATIP
1425  .MACRO   DATOB            ::=  BEGIN/YES,SELECT/BUS,BUSCODE/DATOB
1426  .MACRO   INVALIDATE       ::=  BEGIN/YES,SELECT/BUS,BUSCODE/INVALIDATE
1427
1428
1429
1430
1431
1432  !=====================================================================
1433
1434  .TOC     *     UCON CONTROL MACROS
1435  .MACRO   SET-UCON-CONTROL         ::=  BEGIN/YES,SELECT/UCON,UCON-LOAD/YES
1436                                            !LOAD UCON CONTROL REGISTER AT P0
1437  .MACRO   UCON-OPERATION           ::=  BEGIN/YES,SELECT/UCON,UCON-XFER/YES
1438                                            !PERFORM UCON OPERATION
1439
1440
```

```
1441
1442  .TOC     *       PROCESSOR UCON CONTROL SETUP
1443  .MACRO   UCON-PROC                ::=  UCON-SEL-PROC/YES
1444                                           !SELECT PROCESSOR
1445  .MACRO   EN-CLK-IR[15-00]         ::=  UCON15/1
1446                                           !ENABLE OPERATIONS
1447  .MACRO   EN-CLK-PS[15-12]         ::=  UCON14/1
1448  .MACRO   EN-CLK-FLAG[8-0]         ::=  UCON13/1
1449  .MACRO   EN-CLK-FPS[7-4]          ::=  UCON12/1
1450  .MACRO   EN-CLK-PS[7-4]           ::=  UCON11/1
1451  .MACRO   EN-CLK-PS[3-0]           ::=  UCON10/1
1452  .MACRO   EN-CLK-UBREAK[11-00]     ::=  UCON09/1
1453           !UCON<8:7> ARE NOT USED IN PROCESSOR CONTROL
1454  .MACRO   BUSDIN_EMIT[15-00]       ::=  UCON06/0,UCON05/0 !HBMUX SELECT
1455  .MACRO   BUSDIN_CUA[14-03]        ::=  UCON06/0,UCON05/1
1456  .MACRO   BUSDIN_PS[15-00]         ::=  UCON06/1,UCON05/0
1457  .MACRO   BUSDIN_FLAG[8-0]#FPS[7-0] ::=  UCON06/1,UCON05/1
1458
1459
1460
1461
1462
1463
1464  .TOC     *       CACHE/KT UCON CONTROL
1465  .MACRO   UCON-CACHE-KT                    ::=  UCON-SEL-CACHEKT/YES
1466                                  !SELECT CACHE / KT UCON FUNCTION
1467           !UCON<15> NOT USED HERE
1468  .MACRO   EN-KT-NO-RELOCATE                ::=  UCON14/1
1469                          !INHIBIT KT FROM ANY RELOCATION OF BA -> PBA
1470  .MACRO   BUSDIN_BUS-INTERNAL-ADDR[15-00] ::=  UCON13/0,UCON12/1
1471                                  !FROM INTERNAL ADDR ROM
1472  .MACRO   BUSDIN_CPU-INTERNAL-ADDR[15-00] ::=  UCON13/1,UCON12/1
1473                                  ! DITTO ...
1474  .MACRO   BUSDIN_MMR2[15-00]               ::=  UCON11/1,UCON09/0
1475                                  !VIRTUAL PC
1476  .MACRO   BUSDIN_CACHE-STATUS[15-00]       ::=  UCON11/1,UCON09/1
1477                                  !CACHE INFO
1478  .MACRO   BUSDIN_KT-SEL                    ::=  UCON10/1
1479                                  !FOR PAR-S, PDR-S ETC
1480  .MACRO   KT-WRITE-HIGH                    ::=  UCON08/1
1481                                  !WRITE REGISTER <15:08>
1482  .MACRO   KT-WRITE-LOW                     ::=  UCON07/1
1483                                  !WRITE REGISTER <07:00>
1484  .MACRO   KT-WRITE                         ::=  UCON08/1,UCON07/1
1485                                  !WRITE REGISTER <15:00>
1486  .MACRO   KT-SEL-SLR#CCR                   ::=  UCON06/0,UCON05/0
1487                                  !SELECT KT-MUX OUTPUT
1488  .MACRO   KT-SEL-MMR0                      ::=  UCON06/0,UCON05/1
1489                                  !
1490  .MACRO   KT-SEL-PDR                       ::=  UCON06/1,UCON05/0
1491                                  !
1492  .MACRO   KT-SEL-PAR                       ::=  UCON06/1,UCON05/1
1493                                  !
1494
1495
1496
```

```
1497
1498  .TOC     *       I/O UCON CONTROL
1499  .MACRO   UCON-I-O        ::=  UCON-SEL-I-O/YES   !SELECT I-O CONTROL
1500
1501
1502
1503  .TOC     *         BUS CONTROL
1504  .MACRO   EN-LOAD-DBUF[15-00]      ::=  UCON15/1   !EN LOAD DBUF AT P3
1505  .MACRO   BUSDIN_DBUF[15-00]       ::=  UCON15/1   !DBUF ON BUSDIN
1506  .MACRO   EN-STATUS-MUX   ::=  UCON15/0   !STATUS-MUX ENABLE ON BUSDIN
1507           !UCON<14:11> ARE NOT USED IN UCON BUS CONTROL
1508  .MACRO   BUSDIN_SERVICE[15-00]    ::=  UCON10/0,UCON09/1
1509  .MACRO   BUSDIN_JAM[15-00]        ::=  UCON10/1,UCON09/0
1510  .MACRO   BUSDIN_PBA[15-00]        ::=  UCON10/1,UCON09/1
1511  .MACRO   DMUX_CACHEDATA[15-00]    ::=  UCON08/1
1512  .MACRO   EN-BC-FCN-0              ::=  UCON07/0,UCON06/0,UCON05/0
1513                                            !SELECT BUS CONTROL FUNCTION
1514  .MACRO   EN-START-DELAY           ::=  UCON07/0,UCON06/0,UCON05/1
1515  .MACRO   EN-CLR-JAM-ERRORS        ::=  UCON07/0,UCON06/1,UCON05/0
1516  .MACRO   EN-CLR-NPR-TIMEOUT       ::=  UCON07/0,UCON06/1,UCON05/1
1517  .MACRO   EN-CLR-PWR-FAIL          ::=  UCON07/1,UCON06/0,UCON05/0
1518  .MACRO   EN-CLR-YELLOW-ZONE       ::=  UCON07/1,UCON06/0,UCON05/1
1519  .MACRO   EN-ALLOW-BG[1]H          ::=  UCON07/1,UCON06/1,UCON05/0
1520  .MACRO   EN-BUS-INIT-UCON         ::=  UCON07/1,UCON06/1,UCON05/1
1521
1522
1523
1524  .TOC     *         CONSOLE I-O
1525  .MACRO   EN-CONSOLE-COMMAND       ::=  UCON15/0,UCON14/0
1526                                    !SETS UP UCON I-O BITS FOR CONSOLE COMMANDS
1527
1528                                    !ALSO SELECTS STATUS-MUX ON BUSDIN
1529  .MACRO   EN-CNSL-NOP              ::=  UCON13/0,UCON12/0,UCON11/0
1530                                    !ENABLE CONSOLE NO OPERATION
1531  .MACRO   EN-CLR-COUNTR            ::=  UCON13/0,UCON12/0,UCON11/1
1532                                    !ENABLE CLEAR DIGIT PAIR COUNTER
1533  .MACRO   EN-INCR-COUNTR           ::=  UCON13/0,UCON12/1,UCON11/0
1534                                    !ENABLE BUMP TO NEXT DIGIT PAIR
1535  .MACRO   EN-CLR-CNSL-SRVC         ::=  UCON13/0,UCON12/1,UCON11/1
1536                                    !ENABLE CLEAR CONSOLE SERVICE RQST FLOP
1537  .MACRO   EN-STRB-DISP             ::=  UCON13/1,UCON12/0,UCON11/0
1538                                    !ENABLE WRITE DIGIT PAIR TO DISPLAY LATCH
1539  .MACRO   EN-CLR-CNSL              ::=  UCON13/1,UCON12/0,UCON11/1
1540                                    !ENABLE CLEAR CONSOLE LED
1541  .MACRO   EN-SET-CNSL              ::=  UCON13/1,UCON12/1,UCON11/0
1542                                    !ENABLE SET CONSOLE LED
1543  .MACRO   EN-SET-DP                ::=  UCON13/1,UCON12/1,UCON11/1
1544                                    !ENABLE SET ALL DP LEDS
1545  .MACRO   BUSDIN_CONSOLE[06-00]    ::=  UCON10/0,UCON09/0
1546                                    !STATUS-MUX SELECT
1547           !UCON<8:5> ARE NOT USED IN UCON CONSOLE CONTROL
1548
1549
```

```
1550
1551  .TOC     *          REMOTE CONSOLE INTERFACE
1552  !N.B.:  "EN CONSOLE COMMAND" DOES NOT APPLY TO REMOTE CONSOLE
1553  .MACRO   EN-REMSTRB              ::=  UCON14/1    !EN REMOTE CONSOLE STROBE
1554  .MACRO   EN-REMCODE1             ::=  UCON12/1    !EN SPECIAL CODE 1
1555  .MACRO   EN-REMCODE0             ::=  UCON11/1    !EN SPECIAL CODE 0
1556
1557
1558
1559  !=====================================================================
1560
1561
1562  .TOC     *      MICROBRANCH FIELD MACROS
1563  ![SEE <UBF> FIELD DESCRIPTION FOR FULL INFO]
1564
1565  .MACRO   BUT(XX)                 ::= UBF/@XX   !INACTIVE, FULL WIDTH
1566  .MACRO   BUTR(XX)                ::=  UBF/@XX !INACTIVE, RESTRICTED WIDTH
1567
1568  .MACRO   BUTA(XX)                ::=  UBF/@XX !ACTIVE, FULL WIDTH
1569  .MACRO   BUTRA(XX)               ::=  UBF/@XX !ACTIVE, RESTRICTED WIDTH
1570
1571  .MACRO   TEST(XX)                ::=  MULTIPLE/@XX
1572                                            !FOR BUTR(MULTIPLE) SETUP
1573  .MACRO   BUTM(XX)                ::=  MULTIPLE/@XX,UBF/@XX
1574                                            !A MULTIPLE BUTR
1575
1576
1577
1578  !=====================================================================
1579
1580  .TOC     *      MISCELLANEOUS
1581
1582  .TOC     *        OTHER SOURCES ENABLED FOR A-BUS
1583  .MACRO   SR                      ::=  AEN/XMUX,ASEL0/SR
1584  .MACRO   FLTPT                   ::=  AEN/XMUX,ASEL0/FLTPT
1585
1586
1587
1588  .TOC     *        PAGING, RETURN REGISTER
1589
1590           !PAGE FIELD ONLY:
1591  .MACRO   PAGE(X)                 ::=  NEXT-PAGE/@X
1592
1593           !PAGE FIELD AND BUT[SUBR B]:
1594  .MACRO   GOTO-PAGE(X)            ::=  NEXT-PAGE/@X,UBF/SUBR-B
1595
1596           !RETURN REGISTER <- D<14:03>, PAGE <- EMIT<02:00> ON BUTA(SUBR-A)
1597  .MACRO   RETURN_D[14-03]         ::=  UBF/SUBR-A
1598
1599           !SUBROUTINE CALL (PAGE MUST ALSO BE SPECIFIED)
1600  .MACRO   CALL(SUB,RETURN)::=     BUT(SUBRA),J/@SUB,RETURN/@RETURN
1601
1602
```

```
1603  !=====================================================================
1604
1605  .TOC     *   ADVANCED OPERATIONS
1606
1607
1608
1609  .TOC     *     DATA INTO CSP, AT P3 ONLY
1610
1611           !N.B.:  BUSDIN IS ANY BUT EMIT [OVERLAPS BSEL<1:0>]
1612  .MACRO   CSPB[14]_BUSDIN          ::=  CSPB(B14),WR-CSP
1613  .MACRO   CSPB[15]_BUSDIN          ::=  CSPB(B15),WR-CSP
1614  .MACRO   CSPB[16]_BUSDIN          ::=  CSPB(B16),WR-CSP
1615  .MACRO   CSPB[17]_BUSDIN          ::=  CSPB(B17),WR-CSP
1616  .MACRO   CSPB[MD]_BUSDIN          ::=  CSPB(MD),WR-CSP
1617
1618           !N.B.:  GETS WHATEVER IS ON BUSDIN
1619  .MACRO   CSPD[00]_BUSDIN          ::=  CSPD(D00),WR-CSP
1620  .MACRO   CSPD[01]_BUSDIN          ::=  CSPD(D01),WR-CSP
1621  .MACRO   CSPD[02]_BUSDIN          ::=  CSPD(D02),WR-CSP
1622  .MACRO   CSPD[03]_BUSDIN          ::=  CSPD(D03),WR-CSP
1623  .MACRO   CSPD[04]_BUSDIN          ::=  CSPD(D04),WR-CSP
1624  .MACRO   CSPD[05]_BUSDIN          ::=  CSPD(D05),WR-CSP
1625  .MACRO   CSPD[06]_BUSDIN          ::=  CSPD(D06),WR-CSP
1626  .MACRO   CSPD[07]_BUSDIN          ::=  CSPD(D07),WR-CSP
1627  .MACRO   CSPD[10]_BUSDIN          ::=  CSPD(D10),WR-CSP
1628  .MACRO   CSPD[11]_BUSDIN          ::=  CSPD(D11),WR-CSP
1629  .MACRO   CSPD[12]_BUSDIN          ::=  CSPD(D12),WR-CSP
1630  .MACRO   CSPD[13]_BUSDIN          ::=  CSPD(D13),WR-CSP
1631  .MACRO   CSPD[14]_BUSDIN          ::=  CSPD(D14),WR-CSP
1632  .MACRO   CSPD[15]_BUSDIN          ::=  CSPD(D15),WR-CSP
1633  .MACRO   CSPD[16]_BUSDIN          ::=  CSPD(D16),WR-CSP
1634  .MACRO   CSPD[17]_BUSDIN          ::=  CSPD(D17),WR-CSP
1635  .MACRO   CSPD[MD]_BUSDIN          ::=  CSPD(D15),WR-CSP
1636
1637           !N.B.:  REQUIRED THAT BUSDIN_EMIT[15-00] PREVIOUSLY SET UP
1638  .MACRO   CSPD[00]_EMIT            ::=  CSPD(D00),WR-CSP
1639  .MACRO   CSPD[01]_EMIT            ::=  CSPD(D01),WR-CSP
1640  .MACRO   CSPD[02]_EMIT            ::=  CSPD(D02),WR-CSP
1641  .MACRO   CSPD[03]_EMIT            ::=  CSPD(D03),WR-CSP
1642  .MACRO   CSPD[04]_EMIT            ::=  CSPD(D04),WR-CSP
1643  .MACRO   CSPD[05]_EMIT            ::=  CSPD(D05),WR-CSP
1644  .MACRO   CSPD[06]_EMIT            ::=  CSPD(D06),WR-CSP
1645  .MACRO   CSPD[07]_EMIT            ::=  CSPD(D07),WR-CSP
1646  .MACRO   CSPD[10]_EMIT            ::=  CSPD(D10),WR-CSP
1647  .MACRO   CSPD[11]_EMIT            ::=  CSPD(D11),WR-CSP
1648  .MACRO   CSPD[12]_EMIT            ::=  CSPD(D12),WR-CSP
1649  .MACRO   CSPD[13]_EMIT            ::=  CSPD(D13),WR-CSP
1650  .MACRO   CSPD[14]_EMIT            ::=  CSPD(D14),WR-CSP
1651  .MACRO   CSPD[15]_EMIT            ::=  CSPD(D15),WR-CSP
1652  .MACRO   CSPD[16]_EMIT            ::=  CSPD(D16),WR-CSP
1653  .MACRO   CSPD[17]_EMIT            ::=  CSPD(D17),WR-CSP
1654  .MACRO   CSPD[MD]_EMIT(XX)        ::=  CSPD(D15),WR-CSP,EMIT/@XX
1655
1656
1657
1658
1659
```

```
1660
1661  .TOC     *     DATA INTO ASP, BSP, AT P2-T * P3
1662
1663  .MACRO   ASPLO[17]_CSPB(XX)      ::=  B,ASPLO(R17),CSPB(@XX),
1664                                           CLK-D,P2-T,WR(A,L,A)
1665  .MACRO   ASPLO[17]_CSPD(XX)      ::=  B,ASPLO(R17),CSPD(@XX),
1666                                           CLK-D,P2-T,WR(A,L,A)
1667  .MACRO   PC_D                    ::=  PC-A,WR(AB,L,A)
1668  .MACRO   R5_D                    ::=  R5-A,WR(AB,L,A)
1669
1670  .MACRO   ASPLO[00]_D             ::=  ASP(R00),WR(A,L,A)
1671  .MACRO   ASPLO[01]_D             ::=  ASP(R01),WR(A,L,A)
1672  .MACRO   ASPLO[02]_D             ::=  ASP(R02),WR(A,L,A)
1673  .MACRO   ASPLO[03]_D             ::=  ASP(R03),WR(A,L,A)
1674  .MACRO   ASPLO[04]_D             ::=  ASP(R04),WR(A,L,A)
1675  .MACRO   ASPLO[05]_D             ::=  ASP(R05),WR(A,L,A)
1676  .MACRO   ASPLO[06]_D             ::=  ASP(R06),WR(A,L,A)
1677  .MACRO   ASPLO[07]_D             ::=  ASP(R07),WR(A,L,A)
1678  .MACRO   ASPLO[10]_D             ::=  ASP(R10),WR(A,L,A)
1679  .MACRO   ASPLO[11]_D             ::=  ASP(R11),WR(A,L,A)
1680  .MACRO   ASPLO[12]_D             ::=  ASP(R12),WR(A,L,A)
1681  .MACRO   ASPLO[13]_D             ::=  ASP(R13),WR(A,L,A)
1682  .MACRO   ASPLO[14]_D             ::=  ASP(R14),WR(A,L,A)
1683  .MACRO   ASPLO[15]_D             ::=  ASP(R15),WR(A,L,A)
1684  .MACRO   ASPLO[16]_D             ::=  ASP(R16),WR(A,L,A)
1685  .MACRO   ASPLO[17]_D             ::=  ASP(R17),WR(A,L,A)
1686
1687  .MACRO   ASPHI[00]_D             ::=  ASP(R00),WR(A,H,A)
1688  .MACRO   ASPHI[01]_D             ::=  ASP(R01),WR(A,H,A)
1689  .MACRO   ASPHI[02]_D             ::=  ASP(R02),WR(A,H,A)
1690  .MACRO   ASPHI[03]_D             ::=  ASP(R03),WR(A,H,A)
1691  .MACRO   ASPHI[04]_D             ::=  ASP(R04),WR(A,H,A)
1692  .MACRO   ASPHI[05]_D             ::=  ASP(R05),WR(A,H,A)
1693  .MACRO   ASPHI[06]_D             ::=  ASP(R06),WR(A,H,A)
1694  .MACRO   ASPHI[07]_D             ::=  ASP(R07),WR(A,H,A)
1695  .MACRO   ASPHI[10]_D             ::=  ASP(R10),WR(A,H,A)
1696  .MACRO   ASPHI[11]_D             ::=  ASP(R11),WR(A,H,A)
1697  .MACRO   ASPHI[12]_D             ::=  ASP(R12),WR(A,H,A)
1698  .MACRO   ASPHI[13]_D             ::=  ASP(R13),WR(A,H,A)
1699  .MACRO   ASPHI[14]_D             ::=  ASP(R14),WR(A,H,A)
1700  .MACRO   ASPHI[15]_D             ::=  ASP(R15),WR(A,H,A)
1701  .MACRO   ASPHI[16]_D             ::=  ASP(R16),WR(A,H,A)
1702  .MACRO   ASPHI[17]_D             ::=  ASP(R17),WR(A,H,A)
1703
1704  .MACRO   BSPLO[00]_D             ::=  BSP(R00),WR(B,L,B)
1705  .MACRO   BSPLO[01]_D             ::=  BSP(R01),WR(B,L,B)
1706  .MACRO   BSPLO[02]_D             ::=  BSP(R02),WR(B,L,B)
1707  .MACRO   BSPLO[03]_D             ::=  BSP(R03),WR(B,L,B)
1708  .MACRO   BSPLO[04]_D             ::=  BSP(R04),WR(B,L,B)
1709  .MACRO   BSPLO[05]_D             ::=  BSP(R05),WR(B,L,B)
1710  .MACRO   BSPLO[06]_D             ::=  BSP(R06),WR(B,L,B)
1711  .MACRO   BSPLO[07]_D             ::=  BSP(R07),WR(B,L,B)
1712  .MACRO   BSPLO[10]_D             ::=  BSP(R10),WR(B,L,B)
1713  .MACRO   BSPLO[11]_D             ::=  BSP(R11),WR(B,L,B)
1714  .MACRO   BSPLO[12]_D             ::=  BSP(R12),WR(B,L,B)
1715  .MACRO   BSPLO[13]_D             ::=  BSP(R13),WR(B,L,B)
```

```
1716  .MACRO   BSPLO[14]_D          ::=  BSP(R14),WR(B,L,B)
1717  .MACRO   BSPLO[15]_D          ::=  BSP(R15),WR(B,L,B)
1718  .MACRO   BSPLO[16]_D          ::=  BSP(R16),WR(B,L,B)
1719  .MACRO   BSPLO[17]_D          ::=  BSP(R17),WR(B,L,B)
1720
1721  .MACRO   BSPHI[00]_D          ::=  BSP(R00),WR(B,H,B)
1722  .MACRO   BSPHI[01]_D          ::=  BSP(R01),WR(B,H,B)
1723  .MACRO   BSPHI[02]_D          ::=  BSP(R02),WR(B,H,B)
1724  .MACRO   BSPHI[03]_D          ::=  BSP(R03),WR(B,H,B)
1725  .MACRO   BSPHI[04]_D          ::=  BSP(R04),WR(B,H,B)
1726  .MACRO   BSPHI[05]_D          ::=  BSP(R05),WR(B,H,B)
1727  .MACRO   BSPHI[06]_D          ::=  BSP(R06),WR(B,H,B)
1728  .MACRO   BSPHI[07]_D          ::=  BSP(R07),WR(B,H,B)
1729  .MACRO   BSPHI[10]_D          ::=  BSP(R10),WR(B,H,B)
1730  .MACRO   BSPHI[11]_D          ::=  BSP(R11),WR(B,H,B)
1731  .MACRO   BSPHI[12]_D          ::=  BSP(R12),WR(B,H,B)
1732  .MACRO   BSPHI[13]_D          ::=  BSP(R13),WR(B,H,B)
1733  .MACRO   BSPHI[14]_D          ::=  BSP(R14),WR(B,H,B)
1734  .MACRO   BSPHI[15]_D          ::=  BSP(R15),WR(B,H,B)
1735  .MACRO   BSPHI[16]_D          ::=  BSP(R16),WR(B,H,B)
1736  .MACRO   BSPHI[17]_D          ::=  BSP(R17),WR(B,H,B)
1737
1738  .MACRO   A#BSPLO[00]_D        ::=  ASP(R00),BSP(R00),WR(AB,L,A)
1739  .MACRO   A#BSPLO[01]_D        ::=  ASP(R01),BSP(R01),WR(AB,L,A)
1740  .MACRO   A#BSPLO[02]_D        ::=  ASP(R02),BSP(R02),WR(AB,L,A)
1741  .MACRO   A#BSPLO[03]_D        ::=  ASP(R03),BSP(R03),WR(AB,L,A)
1742  .MACRO   A#BSPLO[04]_D        ::=  ASP(R04),BSP(R04),WR(AB,L,A)
1743  .MACRO   A#BSPLO[05]_D        ::=  ASP(R05),BSP(R05),WR(AB,L,A)
1744  .MACRO   A#BSPLO[06]_D        ::=  ASP(R06),BSP(R06),WR(AB,L,A)
1745  .MACRO   A#BSPLO[07]_D        ::=  ASP(R07),BSP(R07),WR(AB,L,A)
1746  .MACRO   A#BSPLO[10]_D        ::=  ASP(R10),BSP(R10),WR(AB,L,A)
1747  .MACRO   A#BSPLO[11]_D        ::=  ASP(R11),BSP(R11),WR(AB,L,A)
1748  .MACRO   A#BSPLO[12]_D        ::=  ASP(R12),BSP(R12),WR(AB,L,A)
1749  .MACRO   A#BSPLO[13]_D        ::=  ASP(R13),BSP(R13),WR(AB,L,A)
1750  .MACRO   A#BSPLO[14]_D        ::=  ASP(R14),BSP(R14),WR(AB,L,A)
1751  .MACRO   A#BSPLO[15]_D        ::=  ASP(R15),BSP(R15),WR(AB,L,A)
1752  .MACRO   A#BSPLO[16]_D        ::=  ASP(R16),BSP(R16),WR(AB,L,A)
1753  .MACRO   A#BSPLO[17]_D        ::=  ASP(R17),BSP(R17),WR(AB,L,A)
1754
1755  .MACRO   A#BSPHI[00]_D        ::=  ASP(R00),BSP(R00),WR(AB,H,A)
1756  .MACRO   A#BSPHI[01]_D        ::=  ASP(R01),BSP(R01),WR(AB,H,A)
1757  .MACRO   A#BSPHI[02]_D        ::=  ASP(R02),BSP(R02),WR(AB,H,A)
1758  .MACRO   A#BSPHI[03]_D        ::=  ASP(R03),BSP(R03),WR(AB,H,A)
1759  .MACRO   A#BSPHI[04]_D        ::=  ASP(R04),BSP(R04),WR(AB,H,A)
1760  .MACRO   A#BSPHI[05]_D        ::=  ASP(R05),BSP(R05),WR(AB,H,A)
1761  .MACRO   A#BSPHI[06]_D        ::=  ASP(R06),BSP(R06),WR(AB,H,A)
1762  .MACRO   A#BSPHI[07]_D        ::=  ASP(R07),BSP(R07),WR(AB,H,A)
1763  .MACRO   A#BSPHI[10]_D        ::=  ASP(R10),BSP(R10),WR(AB,H,A)
1764  .MACRO   A#BSPHI[11]_D        ::=  ASP(R11),BSP(R11),WR(AB,H,A)
1765  .MACRO   A#BSPHI[12]_D        ::=  ASP(R12),BSP(R12),WR(AB,H,A)
1766  .MACRO   A#BSPHI[13]_D        ::=  ASP(R13),BSP(R13),WR(AB,H,A)
1767  .MACRO   A#BSPHI[14]_D        ::=  ASP(R14),BSP(R14),WR(AB,H,A)
1768  .MACRO   A#BSPHI[15]_D        ::=  ASP(R15),BSP(R15),WR(AB,H,A)
1769  .MACRO   A#BSPHI[16]_D        ::=  ASP(R16),BSP(R16),WR(AB,H,A)
1770  .MACRO   A#BSPHI[17]_D        ::=  ASP(R17),BSP(R17),WR(AB,H,A)
1771
```

```
1772  .MACRO   A#BSPLO[00]_D-[A]       ::=  ASP(R00),WR(AB,L,A)
1773  .MACRO   A#BSPLO[01]_D-[A]       ::=  ASP(R01),WR(AB,L,A)
1774  .MACRO   A#BSPLO[02]_D-[A]       ::=  ASP(R02),WR(AB,L,A)
1775  .MACRO   A#BSPLO[03]_D-[A]       ::=  ASP(R03),WR(AB,L,A)
1776  .MACRO   A#BSPLO[04]_D-[A]       ::=  ASP(R04),WR(AB,L,A)
1777  .MACRO   A#BSPLO[05]_D-[A]       ::=  ASP(R05),WR(AB,L,A)
1778  .MACRO   A#BSPLO[06]_D-[A]       ::=  ASP(R06),WR(AB,L,A)
1779  .MACRO   A#BSPLO[07]_D-[A]       ::=  ASP(R07),WR(AB,L,A)
1780  .MACRO   A#BSPLO[10]_D-[A]       ::=  ASP(R10),WR(AB,L,A)
1781  .MACRO   A#BSPLO[11]_D-[A]       ::=  ASP(R11),WR(AB,L,A)
1782  .MACRO   A#BSPLO[12]_D-[A]       ::=  ASP(R12),WR(AB,L,A)
1783  .MACRO   A#BSPLO[13]_D-[A]       ::=  ASP(R13),WR(AB,L,A)
1784  .MACRO   A#BSPLO[14]_D-[A]       ::=  ASP(R14),WR(AB,L,A)
1785  .MACRO   A#BSPLO[15]_D-[A]       ::=  ASP(R15),WR(AB,L,A)
1786  .MACRO   A#BSPLO[16]_D-[A]       ::=  ASP(R16),WR(AB,L,A)
1787  .MACRO   A#BSPLO[17]_D-[A]       ::=  ASP(R17),WR(AB,L,A)
1788
1789  .MACRO   A#BSPHI[00]_D-[A]       ::=  ASP(R00),WR(AB,H,A)
1790  .MACRO   A#BSPHI[01]_D-[A]       ::=  ASP(R01),WR(AB,H,A)
1791  .MACRO   A#BSPHI[02]_D-[A]       ::=  ASP(R02),WR(AB,H,A)
1792  .MACRO   A#BSPHI[03]_D-[A]       ::=  ASP(R03),WR(AB,H,A)
1793  .MACRO   A#BSPHI[04]_D-[A]       ::=  ASP(R04),WR(AB,H,A)
1794  .MACRO   A#BSPHI[05]_D-[A]       ::=  ASP(R05),WR(AB,H,A)
1795  .MACRO   A#BSPHI[06]_D-[A]       ::=  ASP(R06),WR(AB,H,A)
1796  .MACRO   A#BSPHI[07]_D-[A]       ::=  ASP(R07),WR(AB,H,A)
1797  .MACRO   A#BSPHI[10]_D-[A]       ::=  ASP(R10),WR(AB,H,A)
1798  .MACRO   A#BSPHI[11]_D-[A]       ::=  ASP(R11),WR(AB,H,A)
1799  .MACRO   A#BSPHI[12]_D-[A]       ::=  ASP(R12),WR(AB,H,A)
1800  .MACRO   A#BSPHI[13]_D-[A]       ::=  ASP(R13),WR(AB,H,A)
1801  .MACRO   A#BSPHI[14]_D-[A]       ::=  ASP(R14),WR(AB,H,A)
1802  .MACRO   A#BSPHI[15]_D-[A]       ::=  ASP(R15),WR(AB,H,A)
1803  .MACRO   A#BSPHI[16]_D-[A]       ::=  ASP(R16),WR(AB,H,A)
1804  .MACRO   A#BSPHI[17]_D-[A]       ::=  ASP(R17),WR(AB,H,A)
1805
1806  .MACRO   A#BSPLO[00]_D-[B]       ::=  BSP(R00),WR(AB,L,B)
1807  .MACRO   A#BSPLO[01]_D-[B]       ::=  BSP(R01),WR(AB,L,B)
1808  .MACRO   A#BSPLO[02]_D-[B]       ::=  BSP(R02),WR(AB,L,B)
1809  .MACRO   A#BSPLO[03]_D-[B]       ::=  BSP(R03),WR(AB,L,B)
1810  .MACRO   A#BSPLO[04]_D-[B]       ::=  BSP(R04),WR(AB,L,B)
1811  .MACRO   A#BSPLO[05]_D-[B]       ::=  BSP(R05),WR(AB,L,B)
1812  .MACRO   A#BSPLO[06]_D-[B]       ::=  BSP(R06),WR(AB,L,B)
1813  .MACRO   A#BSPLO[07]_D-[B]       ::=  BSP(R07),WR(AB,L,B)
1814  .MACRO   A#BSPLO[10]_D-[B]       ::=  BSP(R10),WR(AB,L,B)
1815  .MACRO   A#BSPLO[11]_D-[B]       ::=  BSP(R11),WR(AB,L,B)
1816  .MACRO   A#BSPLO[12]_D-[B]       ::=  BSP(R12),WR(AB,L,B)
1817  .MACRO   A#BSPLO[13]_D-[B]       ::=  BSP(R13),WR(AB,L,B)
1818  .MACRO   A#BSPLO[14]_D-[B]       ::=  BSP(R14),WR(AB,L,B)
1819  .MACRO   A#BSPLO[15]_D-[B]       ::=  BSP(R15),WR(AB,L,B)
1820  .MACRO   A#BSPLO[16]_D-[B]       ::=  BSP(R16),WR(AB,L,B)
1821  .MACRO   A#BSPLO[17]_D-[B]       ::=  BSP(R17),WR(AB,L,B)
1822
```

```
1823  .MACRO   A#BSPHI[00]_D-[B]       ::=  BSP(R00),WR(AB,H,B)
1824  .MACRO   A#BSPHI[01]_D-[B]       ::=  BSP(R01),WR(AB,H,B)
1825  .MACRO   A#BSPHI[02]_D-[B]       ::=  BSP(R02),WR(AB,H,B)
1826  .MACRO   A#BSPHI[03]_D-[B]       ::=  BSP(R03),WR(AB,H,B)
1827  .MACRO   A#BSPHI[04]_D-[B]       ::=  BSP(R04),WR(AB,H,B)
1828  .MACRO   A#BSPHI[05]_D-[B]       ::=  BSP(R05),WR(AB,H,B)
1829  .MACRO   A#BSPHI[06]_D-[B]       ::=  BSP(R06),WR(AB,H,B)
1830  .MACRO   A#BSPHI[07]_D-[B]       ::=  BSP(R07),WR(AB,H,B)
1831  .MACRO   A#BSPHI[10]_D-[B]       ::=  BSP(R10),WR(AB,H,B)
1832  .MACRO   A#BSPHI[11]_D-[B]       ::=  BSP(R11),WR(AB,H,B)
1833  .MACRO   A#BSPHI[12]_D-[B]       ::=  BSP(R12),WR(AB,H,B)
1834  .MACRO   A#BSPHI[13]_D-[B]       ::=  BSP(R13),WR(AB,H,B)
1835  .MACRO   A#BSPHI[14]_D-[B]       ::=  BSP(R14),WR(AB,H,B)
1836  .MACRO   A#BSPHI[15]_D-[B]       ::=  BSP(R15),WR(AB,H,B)
1837  .MACRO   A#BSPHI[16]_D-[B]       ::=  BSP(R16),WR(AB,H,B)
1838  .MACRO   A#BSPHI[17]_D-[B]       ::=  BSP(R17),WR(AB,H,B)
1839
1840  .MACRO   ASPLO[DF]_D             ::=  ASP-ADDRS-R[DF],WR(A,L,A)
1841  .MACRO   ASPHI[DF]_D             ::=  ASP-ADDRS-R[DF],WR(A,H,A)
1842  .MACRO   BSPLO[DF]_D             ::=  BSP-ADDRS-R[DF],WR(B,L,B)
1843  .MACRO   BSPHI[DF]_D             ::=  BSP-ADDRS-R[DF],WR(B,H,B)
1844
1845  .MACRO   ASPLO[SF]_D             ::=  ASP-ADDRS-R[SF],WR(A,L,A)
1846  .MACRO   ASPHI[SF]_D             ::=  ASP-ADDRS-R[SF],WR(A,H,A)
1847  .MACRO   BSPLO[SF]_D             ::=  BSP-ADDRS-R[SF],WR(B,L,B)
1848  .MACRO   BSPHI[SF]_D             ::=  BSP-ADDRS-R[SF],WR(B,H,B)
1849
1850  .MACRO   A#BSPLO[DF]_D-[A]       ::=  ASP-ADDRS-R[DF],WR(AB,L,A)
1851  .MACRO   A#BSPHI[DF]_D-[A]       ::=  ASP-ADDRS-R[DF],WR(AB,H,A)
1852  .MACRO   A#BSPLO[DF]_D-[B]       ::=  BSP-ADDRS-R[DF],WR(AB,L,B)
1853  .MACRO   A#BSPHI[DF]_D-[B]       ::=  BSP-ADDRS-R[DF],WR(AB,H,B)
1854
1855  .MACRO   A#BSPLO[SF]_D-[A]       ::=  ASP-ADDRS-R[SF],WR(AB,L,A)
1856  .MACRO   A#BSPHI[SF]_D-[A]       ::=  ASP-ADDRS-R[SF],WR(AB,H,A)
1857  .MACRO   A#BSPLO[SF]_D-[B]       ::=  BSP-ADDRS-R[SF],WR(AB,L,B)
1858  .MACRO   A#BSPHI[SF]_D-[B]       ::=  BSP-ADDRS-R[SF],WR(AB,H,B)
1859
1860  .MACRO   A#BSPLO[SF]_D           ::=  ASP-ADDRS-R[SF],
1861                                           BSP-ADDRS-R[SF],WR(AB,L,A)
1862  .MACRO   A#BSPLO[DF]_D           ::=  ASP-ADDRS-R[DF],
1863                                           BSP-ADDRS-R[DF],WR(AB,L,A)
1864  .MACRO   A#BSPHI[SF]_D           ::=  ASP-ADDRS-R[SF],
1865                                           BSP-ADDRS-R[SF],WR(AB,H,A)
1866  .MACRO   A#BSPHI[DF]_D           ::=  ASP-ADDRS-R[DF],
1867                                           BSP-ADDRS-R[DF],WR(AB,H,A)
1868
1869
1870
```

```
1871  !=====================================================================
1873  .TOC     *     D AND SR <- (BUS-A FCN BUS-B), AT P2-T OR P3-T
1874
1875           !LOGIC FUNCTIONS:
1876  .MACRO   SR_ZERO                  ::= ZERO,CLK-SR
1877  .MACRO   SR_A-XOR-B                ::= A-XOR-B,CLK-SR
1878  .MACRO   SR_B                      ::= B,CLK-SR
1879  .MACRO   SR_A-AND-B                ::= A-AND-B,CLK-SR
1880  .MACRO   SR_A-IOR-B                ::= A-IOR-B,CLK-SR
1881  .MACRO   SR_A                      ::= A,CLK-SR
1882  .MACRO   SR_NOT-A                        ::= NOT-A,CLK-SR
1883  .MACRO   SR_NOT-A-AND-B            ::= NOT-A-AND-B,CLK-SR
1884  .MACRO   SR_A-AND-NOT-B            ::= A-AND-NOT-B,CLK-SR
1885  .MACRO   D_ZERO                    ::= ZERO,CLK-D
1886  .MACRO   D_A-XOR-B                 ::= A-XOR-B,CLK-D
1887  .MACRO   D_B                       ::= B,CLK-D
1888  .MACRO   D_A-AND-B                 ::= A-AND-B,CLK-D
1889  .MACRO   D_A-IOR-B                 ::= A-IOR-B,CLK-D
1890  .MACRO   D_A                       ::= A,CLK-D
1891  .MACRO   D_NOT-A                   ::= NOT-A,CLK-D
1892  .MACRO   D_NOT-A-AND-B             ::= NOT-A-AND-B,CLK-D
1893  .MACRO   D_A-AND-NOT-B             ::= A-AND-NOT-B,CLK-D
1894
1895           !ARITH FUNCTIONS:
1896  .MACRO   D_DIVIDE-STEP                   ::= DIVIDE,CLK-D
1897  .MACRO   D_A-PLUS-B                      ::= A-PLUS-B,CLK-D
1898  .MACRO   D_A-PLUS-B-PLUS-0               ::= A-PLUS-B,CLK-D
1899  .MACRO   D_A-MINUS-B                     ::= A-MINUS-B,CLK-D
1900  .MACRO   D_A-PLUS-B-PLUS-PS[C]           ::= A-PLUS-B-PLUS-PS[C],CLK-D
1901  .MACRO   D_A-PLUS-B-PLUS-D[C]            ::= A-PLUS-B-PLUS-D[C],CLK-D
1902  .MACRO   D_A-PLUS-NOT-B-PLUS-D[C]        ::= A-PLUS-NOT-B-PLUS-D[C],CLK-D
1903  .MACRO   D_A-PLUS-B-PLUS-1               ::= A-PLUS-B-PLUS-1,CLK-D
1904  .MACRO   SR_DIVIDE-STEP                  ::= DIVIDE,CLK-SR
1905  .MACRO   SR_A-PLUS-B                     ::= A-PLUS-B,CLK-SR
1906  .MACRO   SR_A-PLUS-B-PLUS-0              ::= A-PLUS-B,CLK-SR
1907  .MACRO   SR_A-MINUS-B                    ::= A-MINUS-B,CLK-SR
1908  .MACRO   SR_A-PLUS-B-PLUS-PS[C]          ::= A-PLUS-B-PLUS-PS[C],CLK-SR
1909  .MACRO   SR_A-PLUS-B-PLUS-D[C]           ::= A-PLUS-B-PLUS-D[C],CLK-SR
1910  .MACRO   SR_A-PLUS-NOT-B-PLUS-D[C]       ::= A-PLUS-NOT-B-PLUS-D[C],CLK-SR
1911  .MACRO   SR_A-PLUS-B-PLUS-1              ::= A-PLUS-B-PLUS-1,CLK-SR
1915
1916  .TOC     *     D[C] GETS SET
1917
1918  .MACRO   D[C]_CINMUX               ::=  CLK-D,COUT_CIN
1919  .MACRO   D[C]_1                    ::=  CLK-D,COUT_CIN
1920                                             !NEEDS SPECIFIC ALU/---
1921  .MACRO   D[C]_0                    ::=  CLK-D,COUT_CIN
1922                                             !NEEDS SPECIFIC ALU/---
1923  .MACRO   D[C]_PS[C]                ::=  CLK-D,COUT_PS[C]
1924  .MACRO   D[C]_ALU00                ::=  CLK-D,COUT_ALU00
1925  .MACRO   D[C]_ALU07                ::=  CLK-D,COUT_ALU07
1926  .MACRO   D[C]_ALU15                ::=  CLK-D,COUT_ALU15
1927  .MACRO   D[C]_COUT07               ::=  CLK-D,COUT_COUT07
1928  .MACRO   D[C]_COUT15               ::=  CLK-D,COUT_COUT15
1929  .MACRO   D[C]_D[C]                 ::=  CLK-D,COUT_D[C]
1930  .MACRO   SAVE-D[C]                 ::=  CLK-D,COUT_D[C]
```

```
1931
1932
1933
1934
1935
1936  .TOC     *     D-REGISTER <- [BBUS = ABUS], BITWISE, AT P2-T OR P3-T
1937
1938           !N.B.:  SHIFT TREE ENABLED SEPARATELY
1939  .MACRO   D_D-SHIFTED-XOR-CSPB(XX)::=  A-XOR-B,CSPB(@XX),CLK-D
1940  .MACRO   D_D-SHIFTED-XOR-BSPHI(XX)::=  A-XOR-B,BSPHI(@XX),CLK-D
1941
1942  .MACRO   D_FLTPT-XOR-CSPB(XX)     ::=  A-XOR-B,FLTPT,CSPB(@XX),CLK-D
1943  .MACRO   D_FLTPT-XOR-CSPD(XX)     ::=  A-XOR-B,FLTPT,CSPD(@XX),CLK-D
1944  .MACRO   D_FLTPT-XOR-BSPHI(XX)    ::=  A-XOR-B,FLTPT,BSPHI(@XX),CLK-D
1945
1946  .MACRO   D_SR-XOR-CSPB(XX)        ::=  A-XOR-B,SR,CSPB(@XX),CLK-D
1947  .MACRO   D_SR-XOR-CSPD(XX)        ::=  A-XOR-B,SR,CSPD(@XX),CLK-D
1948  .MACRO   D_SR-XOR-BSPHI(XX)       ::=  A-XOR-B,SR,BSPHI(@XX),CLK-D
1949
1950  .MACRO   D_ASPLO[17]-XOR-CSPD(XX)::=  A-XOR-B,ASPLO(R17),CSPD(@XX),CLK-D
1951  .MACRO   D_ASPLO[07]-XOR-BSPHI(XX)::=  A-XOR-B,ASPLO(R07),BSPHI(@XX),CLK-D
1952  .MACRO   D_ASPLO[05]-XOR-BSPHI(XX)::=  A-XOR-B,ASPLO(R05),BSPHI(@XX),CLK-D
1953
1954  .MACRO   D_SR-XOR-BSPLO[SF]       ::=  A-XOR-B,SR,R[SF]-LO-B,CLK-D
1955  .MACRO   D_SR-XOR-BSPHI[DF]       ::=  A-XOR-B,SR,R[DF]-HI-B,CLK-D
1956
1957  .MACRO   D_ASPLO[DF]-XOR-BSPHI[SF]::=  A-XOR-B,R[DF]-LO-A,R[SF]-HI-B,CLK-D
1958  .MACRO   D_ASPHI[SF]-XOR-BSPLO[DF]::=  A-XOR-B,R[SF]-HI-A,R[DF]-LO-B,CLK-D
1959
1960  .MACRO   D_CSPD[05]-XOR-ASPLO(XX)::=  A-XOR-B,CSPD(D05),ASPLO(@XX),CLK-D
1961  .MACRO   D_CSPD[05]-XOR-ASPHI(XX)::=  A-XOR-B,CSPD(D05),ASPHI(@XX),CLK-D
1962  .MACRO   D_CSPD[06]-XOR-ASPLO(XX)::=  A-XOR-B,CSPD(D06),ASPLO(@XX),CLK-D
1963  .MACRO   D_CSPD[06]-XOR-ASPHI(XX)::=  A-XOR-B,CSPD(D06),ASPHI(@XX),CLK-D
1964  .MACRO   D_CSPD[17]-XOR-ASPHI(XX)::=  A-XOR-B,CSPD(D17),ASPHI(@XX),CLK-D
1965
1966  .MACRO   D_ASPLO[02]-XOR-BSPLO(XX)::=  A-XOR-B,ASPLO(R02),BSPLO(@XX),CLK-D
1967  .MACRO   D_ASPLO[03]-XOR-BSPLO(XX)::=  A-XOR-B,ASPLO(R03),BSPLO(@XX),CLK-D
1968  .MACRO   D_ASPLO[04]-XOR-BSPLO(XX)::=  A-XOR-B,ASPLO(R04),BSPLO(@XX),CLK-D
1969  .MACRO   D_ASPLO[05]-XOR-BSPLO(XX)::=  A-XOR-B,ASPLO(R05),BSPLO(@XX),CLK-D
1970
1971
1972
```

```
1973  .TOC     *     D-REGISTER <- D-REGISTER THRU SHIFT-TREE
1974
1975  .MACRO   D_D-RIGHT-14            ::=  A,D-RIGHT-14,CLK-D
1976  .MACRO   D_D-RIGHT-13            ::=  A,D-RIGHT-13,CLK-D
1977  .MACRO   D_D-RIGHT-12            ::=  A,D-RIGHT-12,CLK-D
1978  .MACRO   D_D-RIGHT-11            ::=  A,D-RIGHT-11,CLK-D
1979  .MACRO   D_D-RIGHT-10            ::=  A,D-RIGHT-10,CLK-D
1980  .MACRO   D_D-RIGHT-9             ::=  A,D-RIGHT-9,CLK-D
1981  .MACRO   D_D-RIGHT-8             ::=  A,D-RIGHT-8,CLK-D
1982  .MACRO   D_D-RIGHT-7             ::=  A,D-RIGHT-7,CLK-D
1983  .MACRO   D_D-RIGHT-6             ::=  A,D-RIGHT-6,CLK-D
1984  .MACRO   D_D-RIGHT-5             ::=  A,D-RIGHT-5,CLK-D
1985  .MACRO   D_D-RIGHT-4             ::=  A,D-RIGHT-4,CLK-D
1986  .MACRO   D_D-RIGHT-3             ::=  A,D-RIGHT-3,CLK-D
1987  .MACRO   D_D-RIGHT-2             ::=  A,D-RIGHT-2,CLK-D
1988  .MACRO   D_D-RIGHT-1             ::=  A,D-RIGHT-1,CLK-D
1989  .MACRO   D_D-NO-SHIFT            ::=  A,D-NO-SHIFT,CLK-D
1990  .MACRO   D_D-DIRECT              ::=  A,D-DIRECT,CLK-D
1991  .MACRO   D_D                     ::=  A,D-DIRECT,CLK-D
1992  .MACRO   SAVE-D                  ::=  A,D-DIRECT,CLK-D
1993  .MACRO   D_D-LEFT-1              ::=  A,D-LEFT-1,CLK-D
1994  .MACRO   D_D-SWAB                ::=  A,D-SWAB,CLK-D
1995  .MACRO   D_D-SWAB-RIGHT-3        ::=  A,D-SWAB-RIGHT-3,CLK-D
1996  .MACRO   D_D-SWAB-LEFT-1         ::=  A,D-SWAB-LEFT-1,CLK-D
1997  .MACRO   D_D-SIGNEXT             ::=  A,D-SIGNEXT,CLK-D
1998  .MACRO   D_D-SIGNEXT-RIGHT-1     ::=  A,D-SIGNEXT-RIGHT-1,CLK-D
1999  .MACRO   D_D-SIGNEXT-LEFT-1      ::=  A,D-SIGNEXT-LEFT-1,CLK-D
2000  .MACRO   D_NO-SHIFT              ::=  A,NO-SHIFT,CLK-D
2001  .MACRO   D_DIRECT                ::=  A,DIRECT,CLK-D
2002  .MACRO   D_COUNT#D[HI]           ::=  A,COUNT#D[HI],CLK-D
2003  .MACRO   D_COUNT#D[LO]           ::=  A,COUNT#D[LO],CLK-D
2004
2005
2006
```

```
2007  .TOC     *     D <- WHATEVER'S LEFT, AT P2-T OR P3-T
2008
2009  .MACRO   D_NOT-ASPHI(XX)                    ::=  NOT-A,ASPHI(@XX),CLK-D
2010  .MACRO   D_NOT-ASPLO(XX)                    ::=  NOT-A,ASPLO(@XX),CLK-D
2011
2012  .MACRO   D_CSPD(XX)                         ::=  B,CSPD(@XX),CLK-D
2013  .MACRO   D_CSPB(XX)                         ::=  B,CSPB(@XX),CLK-D
2014
2015  .MACRO   D_BSPHI(XX)                        ::=  B,BSPHI(@XX),CLK-D
2016  .MACRO   D_BSPLO(XX)                        ::=  B,BSPLO(@XX),CLK-D
2017  .MACRO   D_ASPHI(XX)                        ::=  A,ASPHI(@XX),CLK-D
2018  .MACRO   D_ASPLO(XX)                        ::=  A,ASPLO(@XX),CLK-D
2019
2020  .MACRO   D_ASPLO[DF]                        ::=  A,R[DF]-LO-A,CLK-D
2021  .MACRO   D_ASPHI[DF]                        ::=  A,R[DF]-HI-A,CLK-D
2022  .MACRO   D_BSPLO[DF]                        ::=  B,R[DF]-LO-B,CLK-D
2023  .MACRO   D_BSPHI[DF]                        ::=  B,R[DF]-HI-B,CLK-D
2024  .MACRO   D_ASPLO[SF]                        ::=  A,R[SF]-LO-A,CLK-D
2025  .MACRO   D_ASPHI[SF]                        ::=  A,R[SF]-HI-A,CLK-D
2026  .MACRO   D_BSPLO[SF]                        ::=  B,R[SF]-LO-B,CLK-D
2027  .MACRO   D_BSPHI[SF]                        ::=  B,R[SF]-HI-B,CLK-D
2028
2029  .MACRO   D_CSPD[14]-AND-ASPHI(XX)::=  A-AND-B,CSPD(D14),ASPHI(@XX),CLK-D
2030  .MACRO   D_CSPD[15]-AND-ASPHI(XX)::=  A-AND-B,
2031                                           CSPD(D15),ASPHI(@XX),CLK-D
2032
2033  .MACRO   SR_ASPHI[17]-AND-007700            ::=  A-AND-B,ASPHI(R17),
2034                                           CSPB(B17),CLK-SR
2035  .MACRO   D_SR-IOR-170000                    ::=  A-IOR-B,SR,
2036                                           CSPB(B16),CLK-D
2037  .MACRO   SR_ASPHI[17]-AND-000077            ::=  A-AND-B,ASPHI(R17),
2038                                           CSPB(B15),CLK-SR
2039  .MACRO   D_SR-IOR-000100                    ::=  A-IOR-B,SR,
2040                                           CSPB(B14),CLK-D
2041
2042  .MACRO   D_ASPLO[17]-AND-CSPD(XX)           ::=  A-AND-B,ASPLO(R17),
2043                                           CSPD(@XX),CLK-D
2044  .MACRO   D_ASPHI[00]-IOR-CSPD(XX)           ::=  A-IOR-B,ASPHI(R00),
2045                                           CSPD(@XX),CLK-D
2046  .MACRO   D_ASPHI[00]-IOR-CSPB(XX)::=  A-IOR-B,ASPHI(R00),CSPB(@XX),CLK-D
2047
2048  .MACRO   D_SR                               ::=  A,SR,CLK-D
2049  .MACRO   D_JUNK                             ::=  ZERO,CLK-D
2050
2051
2052
```

```
2053  !=====================================================================
2054
2055  .TOC      *      SR <- DATA, AT P2 T OR P3 T
2056
2057            !N.B.:  THE PARTICULAR FUNCTION SELECTED REQUIRES THE RESIDUAL
2058            !       CONTROL REGISTER ("RES-REG") TO HAVE THE APPROPRIATE
2059            !       FUNCTION SETUP FOR THE SR OPERATION.
2060            !
2061            !       POSSIBLE FUNCTIONS:  LOAD, LEFT, RIGHT, NOP
2062
2063  .MACRO    SR_ASPHI(XX)                ::=  A,ASPHI(@XX),CLK-SR
2064  .MACRO    SR_NOT-ASPHI(XX)            ::=  NOT-A,ASPHI(@XX),CLK-SR
2065  .MACRO    SR_CSPB(XX)                 ::=  B,CSPB(@XX),CLK-SR
2066  .MACRO    SR_CSPD(XX)                 ::=  B,CSPD(@XX),CLK-SR
2067  .MACRO    SR_BSPHI(XX)                ::=  B,BSPHI(@XX),CLK-SR
2068  .MACRO    SR_SR-PLUS-1                ::=  A-PLUS-B,C000001-B,SR,CLK-SR
2069  .MACRO    SR_ALL-ONES                 ::=  A,C177777-A,CLK-SR
2070  .MACRO    SR_SR-RIGHT-1               ::=  D-DIRECT[BMUX],CLK-SR
2071  .MACRO    SR_SR-LEFT-1                ::=  CLK-SR
2072  .MACRO    SR_JUNK                     ::=  ZERO,CLK-SR
2073  .MACRO    SR_D                        ::=  A,D-DIRECT,CLK-SR
2074  .MACRO    SR_ASPLO[DF]                ::=  A,R[DF]-LO-A,CLK-SR
2075  .MACRO    SR_ASPHI[DF]                ::=  A,R[DF]-HI-A,CLK-SR
2076  .MACRO    SR_BSPLO[DF]                ::=  B,R[DF]-LO-B,CLK-SR
2077  .MACRO    SR_BSPHI[DF]                ::=  B,R[DF]-HI-B,CLK-SR
2078  .MACRO    SR_ASPLO[SF]                ::=  A,R[SF]-LO-A,CLK-SR
2079  .MACRO    SR_ASPHI[SF]                ::=  A,R[SF]-HI-A,CLK-SR
2080  .MACRO    SR_BSPLO[SF]                ::=  B,R[SF]-LO-B,CLK-SR
2081  .MACRO    SR_BSPHI[SF]                ::=  B,R[SF]-HI-B,CLK-SR
2082
2083
2084
2085
2086
2087  .TOC      *      RES-REG OPERATION MACROS
2088
2089  .MACRO    RES_CSPD(XX)                ::=  CSPD(@XX),LOAD-RES
2090  .MACRO    RES_CSPB(XX)                ::=  CSPB(@XX),LOAD-RES
2091
2092
2093
2094
2095
2096  .TOC      *      BASE MACHINE COUNTER
2097
2098  .MACRO    COUNTER_CSPD(XX)            ::=  LOAD-COUNTER,CSPD(@XX)
2099  .MACRO    COUNTER_BSPHI(XX)           ::=  LOAD-COUNTER,BSPHI(@XX)
2100
2101
2102
2103
```

```
2104  .TOC     *     ENABLE ON BUS-A/B ONLY
2105
2106  .MACRO   BUS-A_ASPLO[SF]          ::=  R[SF]-LO-A
2107  .MACRO   BUS-A_ASPLO[DF]          ::=  R[DF]-LO-A
2108  .MACRO   BUS-A_ASPHI[SF]          ::=  R[SF]-HI-A
2109  .MACRO   BUS-A_ASPHI[DF]          ::=  R[DF]-HI-A
2110  .MACRO   BUS-A                    ::=  NULL
2111  .MACRO   BUS-A_ASPLO(XX)          ::=  ASPLO(@XX)
2112  .MACRO   BUS-A_ASPHI(XX)          ::=  ASPHI(@XX)
2113  .MACRO   BUS-A_SR                 ::=  SR
2114  .MACRO   BUS-A_FLTPT              ::=  FLTPT
2115
2116  .MACRO   BUS-B_BSPLO[SF]          ::=  R[SF]-LO-B
2117  .MACRO   BUS-B_BSPLO[DF]          ::=  R[DF]-LO-B
2118  .MACRO   BUS-B_BSPHI[SF]          ::=  R[SF]-HI-B
2119  .MACRO   BUS-B_BSPHI[DF]          ::=  R[DF]-HI-B
2120  .MACRO   BUS-B                    ::=  NULL
2121  .MACRO   BUS-B_BSPLO(XX)          ::=  BSPLO(@XX)
2122  .MACRO   BUS-B_BSPHI(XX)          ::=  BSPHI(@XX)
2123  .MACRO   BUS-B_CSPD(XX)           ::=  CSPD(@XX)
2124  .MACRO   BUS-B_CSPB(XX)           ::=  CSPB(@XX)
2125
2126
2127
2128
2129
2130  .TOC     *     LOADING BA REGISTER
2131  !LOADED AT P1-T ONLY, FROM BUS-B<01:00>#BUS-A<15:00> -> BA<17:00>
2132
2133  .MACRO   BA_BSPLO(XX)             ::=  CLK-BA,BSPLO(@XX)
2134  .MACRO   BA_BSPHI(XX)             ::=  CLK-BA,BSPHI(@XX)
2135  .MACRO   BA_SR                    ::=  CLK-BA,SR
2136  .MACRO   BA_ASPLO(XX)             ::=  CLK-BA,ASPLO(@XX)
2137  .MACRO   BA_ASPHI(XX)             ::=  CLK-BA,ASPHI(@XX)
2138
2139
2140
2141
2142
2143  .TOC     *     D AND SR TOGETHER
2144
2145  .MACRO   SR#D_SR-PLUS-CSPD(XX)    ::=  A-PLUS-B,SR,CSPD(@XX),CLK-D,CLK-SR
2146
2147
```

```
2148
2149  !=============================================================================
2150
2151  .TOC     *     UCON FUNCTIONS
2152
2153
2154
2155  .TOC     *       PROCESSOR UCON FUNCTIONS
2156
2157   !PREVIOUSLY SET UP [UCON-PROC, SET-UCON-CONTROL, EN "FUNCTION"]
2158  .MACRO   IR_EMIT                  ::=  UCON-OPERATION
2159  .MACRO   PS[15-12]_D[15#13]       ::=  UCON-OPERATION
2160  .MACRO   FLAG[8-0]_D[15-8]        ::=  UCON-OPERATION
2161  .MACRO   FPS[7-4]_D[7-4]          ::=  UCON-OPERATION
2162  .MACRO   PS[7-4]_D[7-4]           ::=  UCON-OPERATION
2163  .MACRO   PS[3-0]_D[3-0]           ::=  UCON-OPERATION
2164  .MACRO   PS_D                     ::=  UCON-OPERATION
2165  .MACRO   UBREAK_BUSDIN[11-00]     ::=  UCON-OPERATION
2166
2167           !SETUP UCON AND EXECUTE IN 1 MICROWORD:
2168  .MACRO   PS[15-12]_D[15#13]-[I]   ::=  UCON-PROC,SET-UCON-CONTROL,
2169                                             UCON-OPERATION,EN-CLK-PS[15-12]
2170  .MACRO   FLAG[8-0]_D[15-8]-[I]    ::=  UCON-PROC,SET-UCON-CONTROL,
2171                                             UCON-OPERATION,EN-CLK-FLAG[8-0]
2172  .MACRO   FPS[7-4]_D[7-4]-[I]      ::=  UCON-PROC,SET-UCON-CONTROL,
2173                                             UCON-OPERATION,EN-CLK-FPS[7-4]
2174  .MACRO   PS[7-4]_D[7-4]-[I]       ::=  UCON-PROC,SET-UCON-CONTROL,
2175                                             UCON-OPERATION,EN-CLK-PS[7-4]
2176  .MACRO   PS[3-0]_D[3-0]-[I]       ::=  UCON-PROC,SET-UCON-CONTROL,
2177                                             UCON-OPERATION,EN-CLK-PS[3-0]
2178  .MACRO   PS_D-[I]                 ::=  UCON-PROC,SET-UCON-CONTROL,
2179           UCON-OPERATION,EN-CLK-PS[15-12],EN-CLK-PS[7-4],EN-CLK-PS[3-0]
2180  .MACRO   BUSDIN_CUA-[I]           ::=  UCON-PROC,SET-UCON-CONTROL,
2181                                             BUSDIN_CUA[14-03]
2182  .MACRO   BUSDIN_FLAGS#FPS-[I]     ::=  UCON-PROC,SET-UCON-CONTROL,
2183                                             BUSDIN_FLAG[8-0]#FPS[7-0]
2184  .MACRO   BUSDIN_PS-[I]            ::=  UCON-PROC,SET-UCON-CONTROL,
2185                                             BUSDIN_PS[15-00]
2186  .MACRO   BUSDIN_EMIT-[I]          ::=  UCON-PROC,SET-UCON-CONTROL,
2187                                             BUSDIN_EMIT[15-00]
2188
2189
```

```
2190
2191  .TOC     *       CACHE/KT UCON FUNCTIONS
2192
2193           !SETUP, EXECUTE IN 1 MICROWORD
2194  .MACRO   KT-NO-RELOCATE-[I]              ::=  UCON-CACHE-KT,
2195           SET-UCON-CONTROL,EN-KT-NO-RELOCATE
2196  .MACRO   BUSDIN_BUS-INTERNAL-ADDR-[I]    ::=  UCON-CACHE-KT,
2197           SET-UCON-CONTROL,BUSDIN_BUS-INTERNAL-ADDR[15-00]
2198  .MACRO   BUSDIN_CPU-INTERNAL-ADDR-[I]    ::=  UCON-CACHE-KT,
2199           SET-UCON-CONTROL,BUSDIN_CPU-INTERNAL-ADDR[15-00]
2200  .MACRO   BUSDIN_MMR2-[I]                 ::=  UCON-CACHE-KT,
2201           SET-UCON-CONTROL,BUSDIN_MMR2[15-00]
2202  .MACRO   BUSDIN_CACHE-STATUS-[I]         ::=  UCON-CACHE-KT,
2203           SET-UCON-CONTROL,BUSDIN_CACHE-STATUS[15-00]
2204  .MACRO   BUSDIN_SLR#CCR-[I]              ::=  UCON-CACHE-KT,
2205           SET-UCON-CONTROL,BUSDIN_KT-SEL,KT-SEL-SLR#CCR
2206  .MACRO   BUSDIN_MMR0-[I]                 ::=  UCON-CACHE-KT,
2207           SET-UCON-CONTROL,BUSDIN_KT-SEL,KT-SEL-MMR0
2208  .MACRO   BUSDIN_PDR-[I]                  ::=  UCON-CACHE-KT,
2209           SET-UCON-CONTROL,BUSDIN_KT-SEL,KT-SEL-PDR
2210  .MACRO   BUSDIN_PAR-[I]                  ::=  UCON-CACHE-KT,
2211           SET-UCON-CONTROL,BUSDIN_KT-SEL,KT-SEL-PAR
2212  .MACRO   SLR[15-08]_D[15-08]-[I]         ::=  UCON-CACHE-KT,
2213           SET-UCON-CONTROL,UCON-OPERATION,KT-SEL-SLR#CCR,KT-WRITE-HIGH
2214  .MACRO   CCR[07-02]_D[07-02]-[I]         ::=  UCON-CACHE-KT,
2215           SET-UCON-CONTROL,UCON-OPERATION,KT-SEL-SLR#CCR,KT-WRITE-LOW
2216  .MACRO   MMR0_D-[I]                      ::=  UCON-CACHE-KT,
2217           SET-UCON-CONTROL,UCON-OPERATION,KT-SEL-MMR0,KT-WRITE
2218  .MACRO   MMR0[00]_D[00]-[I]              ::=  UCON-CACHE-KT,
2219           SET-UCON-CONTROL,UCON-OPERATION,KT-SEL-MMR0,KT-WRITE-LOW
2220  .MACRO   MMR0[15-01]_D[15-01]-[I]        ::=  UCON-CACHE-KT,
2221           SET-UCON-CONTROL,UCON-OPERATION,KT-SEL-MMR0,KT-WRITE-HIGH
2222  .MACRO   PDR_D-[I]                       ::=  UCON-CACHE-KT,
2223           SET-UCON-CONTROL,UCON-OPERATION,KT-SEL-PDR,KT-WRITE
2224  .MACRO   PDR[03-01]_D[03-01]-[I]         ::=  UCON-CACHE-KT,
2225           SET-UCON-CONTROL,UCON-OPERATION,KT-SEL-PDR,KT-WRITE-LOW
2226  .MACRO   PDR[14-08]_D[14-08]-[I]         ::=  UCON-CACHE-KT,
2227           SET-UCON-CONTROL,UCON-OPERATION,KT-SEL-PDR,KT-WRITE-HIGH
2228  .MACRO   PAR_D-[I]                       ::=  UCON-CACHE-KT,
2229           SET-UCON-CONTROL,UCON-OPERATION,KT-SEL-PAR,KT-WRITE
2230  .MACRO   PAR[07-00]_D[07-00]-[I]         ::=  UCON-CACHE-KT,
2231           SET-UCON-CONTROL,UCON-OPERATION,KT-SEL-PAR,KT-WRITE-LOW
2232  .MACRO   PAR[11-08]_D[11-08]-[I]         ::=  UCON-CACHE-KT,
2233           SET-UCON-CONTROL,UCON-OPERATION,KT-SEL-PAR,KT-WRITE-HIGH
2234
2235
2236
```

```
2237  .TOC     *       I-O UCON FUNCTIONS
2238
2239           !N.B.:  SETUP IN 1 MICROWORD
2240  .MACRO   BUSDIN_JAM-[I]          ::=  UCON-I-O,EN-STATUS-MUX,
2241                                   SET-UCON-CONTROL,BUSDIN_JAM[15-00]
2242  .MACRO   BUSDIN_SERVICE-[I]      ::=  UCON-I-O,EN-STATUS-MUX,
2243                                   SET-UCON-CONTROL,BUSDIN_SERVICE[15-00]
2244  .MACRO   BUSDIN_PBA-[I]          ::=  UCON-I-O,EN-STATUS-MUX,
2245                                   SET-UCON-CONTROL,BUSDIN_PBA[15-00]
2246  .MACRO   BC-FCN-0-[I]            ::=  UCON-I-O,SET-UCON-CONTROL,
2247                                   UCON-OPERATION,EN-BC-FCN-0
2248  .MACRO   START-DELAY-[I]         ::=  UCON-I-O,SET-UCON-CONTROL,
2249                                   UCON-OPERATION,EN-START-DELAY
2250  .MACRO   CLR-JAM-ERRORS-[I]      ::=  UCON-I-O,SET-UCON-CONTROL,
2251                                   UCON-OPERATION,EN-CLR-JAM-ERRORS
2252  .MACRO   CLR-NPR-TIMEOUT-[I]     ::=  UCON-I-O,SET-UCON-CONTROL,
2253                                   UCON-OPERATION,EN-CLR-NPR-TIMEOUT
2254  .MACRO   CLR-PWR-FAIL-[I]        ::=  UCON-I-O,SET-UCON-CONTROL,
2255                                   UCON-OPERATION,EN-CLR-PWR-FAIL
2256  .MACRO   CLR-YELLOW-ZONE-[I]     ::=  UCON-I-O,SET-UCON-CONTROL,
2257                                   UCON-OPERATION,EN-CLR-YELLOW-ZONE
2258  .MACRO   ALLOW-BG[1]H-[I]        ::=  UCON-I-O,SET-UCON-CONTROL,
2259                                   UCON-OPERATION,EN-ALLOW-BG[1]H
2260  .MACRO   BUS-INIT-UCON-[I]       ::=  UCON-I-O,SET-UCON-CONTROL,
2261                                   UCON-OPERATION,EN-BUS-INIT-UCON
2262
2263
2264
2265
2266  .TOC     *       CONSOLE UCON FUNCTIONS
2267
2268           !SETS UP AND PERFORMS INDICATED OPERATION IN 1 MICROWORD
2269  .MACRO   CONSOLE-NOP                    ::=  UCON-I-O,
2270           EN-CONSOLE-COMMAND,SET-UCON-CONTROL,UCON-OPERATION,EN-CNSL-NOP
2271  .MACRO   CLEAR-CONSOLE-COUNTER           ::=  UCON-I-O,
2272           EN-CONSOLE-COMMAND,SET-UCON-CONTROL,UCON-OPERATION,EN-CLR-COUNTR
2273  .MACRO   INCREMENT-CONSOLE-COUNTER       ::=  UCON-I-O,
2274           EN-CONSOLE-COMMAND,SET-UCON-CONTROL,UCON-OPERATION,EN-INCR-COUNTR
2275  .MACRO   CLEAR-CONSOLE-SERVICE           ::=  UCON-I-O,
2276           EN-CONSOLE-COMMAND,SET-UCON-CONTROL,UCON-OPERATION,EN-CLR-CNSL-SRVC
2277  .MACRO   STROBE-CONSOLE-DISPLAY          ::=  UCON-I-O,
2278           EN-CONSOLE-COMMAND,SET-UCON-CONTROL,UCON-OPERATION,EN-STRB-DISP
2279  .MACRO   CLEAR-CONSOLE-LED               ::=  UCON-I-O,
2280           EN-CONSOLE-COMMAND,SET-UCON-CONTROL,UCON-OPERATION,EN-CLR-CNSL
2281  .MACRO   SET-CONSOLE-LED                 ::=  UCON-I-O,
2282           EN-CONSOLE-COMMAND,SET-UCON-CONTROL,UCON-OPERATION,EN-SET-CNSL
2283  .MACRO   SET-CONSOLE-DP-LEDS             ::=  UCON-I-O,
2284           EN-CONSOLE-COMMAND,SET-UCON-CONTROL,UCON-OPERATION,EN-SET-DP
2285  .MACRO   BUSDIN_CONSOLE-[I]              ::=  UCON-I-O,
2286           EN-STATUS-MUX,SET-UCON-CONTROL,BUSDIN_CONSOLE[06-00]
2287
2288
2289
```

```
2290  .TOC      *       DBUF UCON FUNCTIONS
2291
2292            !PREVIOUSLY SETUP UCON-I-O, EN LOAD DBUF
2293  .MACRO    DBUF_D                  ::=  UCON-OPERATION
2294
2295            !SETUP AND EXECUTE IN 1 MICROWORD:
2296  .MACRO    DBUF_D-[I]              ::=  UCON-I-O,SET-UCON-CONTROL,
2297                                           UCON-OPERATION,EN-LOAD-DBUF[15-00]
2298
2299
2300  .TOC      *       MULTIPLE UCON FUNCTIONS
2301
2302            !THESE ARE FUNCTIONS OF MORE THAN 1 UCON ENABLED SIMULTANEOUSLY
2303
2304            !PREVIOUSLY SETUP:
2305  .MACRO    IR_DBUF                 ::=  UCON-OPERATION
2306
2307            !SETUP AND EXECUTE IN 1 MICROWORD:
2308  .MACRO    IR_DBUF-[I]             ::=  UCON-PROC,UCON-I-O,SET-UCON-CONTROL,
2309                    UCON-OPERATION,EN-CLK-IR[15-00],BUSDIN_DBUF[15-00]
2310
2311  .TOC      *       WCS FUNCTIONS
2312
2313            !INVOKE A TMS ROUTINE TO USE LOCAL STORE.
2314  .MACRO    TMSPTR_(XX)             ::=     BEGIN/YES,SELECT/UCON,
2315                                    UCON-XFER/YES,UCON-LOAD/YES,UCON/@XX,
2316                                    UCON-SEL-WCS/YES
2317
2318
2319  !=====================================================================
2320
2321
2322
2323  !=====================================================================
2324
2325  .TOC      *     JAM UPP LOG MACROS
2326
2327            !MACROS CONCERNED WITH CSP LOG AFTER UNEXPECTED JAMUPP
2328            !MACROS REQUIRE APPROPRIATE REGISTER ENABLED ON BUSDIN
2329
2330  .MACRO    CSPD[00]_LOG-CUA        ::=  CSPD(D00),WR-CSP
2331  .MACRO    CSPD[01]_LOG-SERVICE    ::=  CSPD(D01),WR-CSP
2332  .MACRO    CSPD[02]_LOG-JAM        ::=  CSPD(D02),WR-CSP
2333
2334
2335
2336  !*****  END OF MACRO DEFINITIONS  *****
2337
2338
2339
```

APPENDIX C

THE DISPATCH FILE AND MEMORY PARTITIONING

This appendix discusses the dispatch file, which is normally included in every microprogram assembly. Then, a technique for partitioning the Writable Control Store so that several separately assembled microprograms can be loaded together and executed is described.

C.1 THE DISPATCH FILE

The dispatch file, DSPTCH, serves two purposes in an 11/60 microprogram assembly. First, it provides for the reservation of the first two hundred words of the Writable Control Store, so that the assembler does not overwrite the words required for the resident section. Second, it provides an entry point mechanism, so that the microprogrammer can designate entry points within the microprogram for any one or more of the eight possible XFCs.

The contents of the dispatch file is as follows:

```
2345
2346    .NLIST
2347
2348    .CODE
2349
2350    AAA:
2351    .BEGIN=10[6002:6003]
2352    .BEGIN=100[6004:6007]
2353    .BEGIN=1000[6010:6017]
2354    .BEGIN=000[6020:6027]
2355    DISPCH:
2356    .BEGIN=1000[6030:6037]
2357    .BEGIN=100000[6040:6077]
2358    .BEGIN=1000000[6100:6177]
2359
2360            PAGE(1),BUT(SUBRB),J/CON99
        6000  0 00000000 00000000 01000000 00000000 00111000 00100000
2361    P3,     CSPD[MD].EMIT(0010),
2362    NEXT,   BUT(SUBRB),PAGE(0),J/TRP00
        6001  0 00001000 00000010 00000000 00101000 00111000 01010111
2363    .LIST
```

C.2 PARTITIONING THE WRITABLE CONTROL STORE

If the programmer wants to have several separately assembled microprograms operating in the Writable Control Store, he must partition the WCS by the following set of actions:

1. Divide the microprograms into a main program with one or more subordinate microprograms.

2. Determine the address ranges to be associated with the microprograms.

3. Assemble the main program with the dispatch file and with entry points for that program and all the subordinate programs.

4. Assemble the subordinate programs without the dispatch file, but with the appropriate .BOUNDS keyword line followed by a .CODE keyword line.

Suppose, for example, the programmer has three programs: LOOKUP, SORT, and MATPAK. The programmer decides that LOOKUP will be the main program (assembled at 6200) and that SORT (assembled at 7000) and MATPAK (assembled at 7400) will be the subordinate programs. He begins by adding the following information to the source for LOOKUP:

```
.CASE 1 OF DISPCH
SORTENTRY:
   J/7000;
.CASE 2 OF DISPCH
MATPAKENTRY:
   J/7400;
.CASE 0 OF DISPCH
LOOKUP:
```

Then, he adds the following to the SORT source:

```
.BOUNDS[7000:7377]
.CODE
```

And the following to the MATPAK source:

```
.BOUNDS[7400:7777]
.CODE
```

He can then assemble and load the three programs as follows:

```
>MIC LOOKUP,LOOKUP=PREDEF,DISPTCH,LOOKUP
>MIC SORT,SORT=PREDEF,SORT
>MIC MATPAK,MATPAK=PREDEF,MATPAK
>MLD WCS=MICPAK,LOOKUP,SORT,MATPAK
```

The .BOUNDS keyword can be used, in separate assemblies, to partition the WCS for several users or for several logical functions for a single user. If the assembler cannot find the required addresses within the specified bounds, then the assembly fails. In specifying the .BOUNDS keyword, the programmer should take into account any future expansions or corrections for the microprogram and allocate some additional space, since discontinuous bounds cannot be specified.

.

APPENDIX D

LINKED LIST EXAMPLE

The example given in this section is a microprogram that implements three subroutines for handling a linked list. The linked list is kept in the local store portion of the WCS. It is assumed to have been initialized.

The microprogram is written in the standard format, which is described in Section 14.3.2. The program documentation is included in the listing as comments.

The three subroutines are:

INSERT	Insert element pointed to by R2 before the element in the list pointed to by R1.
REMOVE	Remove the entry pointed to by R1 from the linked list.
APPEND	Add the entry pointed to by R1 to the end of the list.

TABLE OF CONTENTS

```
   1  .NLIST
2422
2423  .NLIST
2424
2425  !
2426  !
2427  !       *****                    FORWARD                    *****
2428  !       *   *----------------------------------------------->*   *
2429  !       *   *                                          R1 ---->*   *
2430  !       *   *<-----------------------------------------------*   *
2431  !       *****                    BACKWARD                   *****
2432  !
2433  !                                 *****   FORWARD
2434  !                                 *   *---------------> ?
2435  !                        R2 ---->*   *
2436  !                ? <-------------*   *
2437  !                      BACKWARD   *****
2438  !
2439  !
4040  !  THE ABOVE DIAGRAM IS AN ATTEMPT TO DESCRIBE THE INPUT CONDITIONS
2441  !  WHICH EXIST AT THE START OF THE INSERT XFC INSTRUCTION.   THE
2442  !  REGISTERS CONTAIN THE FOLLOWING INFORMATION:
2443  !  R1-     POINTS TO THE ENTRY TO BE INSERTED IN FRONT OF.
2444  !  R2-     POINTS TO THE ENTRY TO BE INSERTED.
2445  !
2446  !
2447  !       *****                                  FORWARD       *****
2448  !       *   *--------.                    .---------------->*   *
2449  !       *   *        |                    |          R1 ---->*   *
2450  !       *   *<---.   |                    |   .------------*   *
2451  !       *****    |   |                    |   |  BACKWARD  *****
2452  !                |   |                    |   |
2453  !                |   |  FORWARD  *****    |   |
2454  !                |   '---------->*   *----'   |
2455  !                |      R2 ---->*   *         |
2456  !                '--------------*   *<--------'
2457  !                     BACKWARD    *****
2458  !
2459  !
2460  !  THE ABOVE DIAGRAM ATTEMPTS TO DOCUMENT THE POINTERS AND THE
2461  !  WAY THEY LOOK AFTER THE INSERT ISTRUCTION HAS COMPLETED ITS EXECUTION.
```

```
2462  !  THE FOLLOWING REPRESENTATION IS A MACRO CODE
2463  !  VERSION OF THE LINKED LIST ALGORITHM:
2464  !
2465  !  INSERT: MOV  R3,-(SP)           ;INSERT ELEMENT POINTED TO BY R2.
2466  !          MOV  R1,FLINK(R2)       ;BEFORE ELEMENT POINTED TO BY R1.
2467  !          MOV  BLINK(R1),BLINK(R2)
2468  !          MOV  BLINK(R1),R3
2469  !          MOV  R2,FLINK(R3)
2470  !          MOV  R2,BLINK(R1)
2471  !          MOV  (SP)+,R3
2472  !
2473  !  THE INSERT INSTRUCTION HAS ONE BIG DIFFERENCE THAN THE MACRO
2474  !  CODE VERSION IN THAT THE LINKED LIST DATA STRUCTURE IS IN
2475  !  LOCAL STORE RATHER THAN MAIN MEMORY.   UNDERSTANDING LOCAL STORE
2476  !  AND THE ADDRESSING MODES REQUIRED WE CAN REWRITE THE ABOVE
2477  !  ALGORITH AS THE FOLLOWING EXPRESSIONS:
2478  !
2479  !          GRAB BLINK(R1)
2480  !          MOV  R2,BLINK(R1)
2481  !          MOV  R1,FLINK(R2)
2482  !          MOV  GRABBED,BLINK(R2)
2483  !          MOV  R2,(GRABBED)
2484  !
2485  !  THE LATTER ALGORITHM ALTHOUGH NOT AS CLEAR AS THE FIRST ONE
2486  !  DOES HAVE THE FOLLOWING PROPERTY THAT THE ADDRESS REGISTER ON
2487  !  THE WCS BOARD NEEDS TO BE LOADED ONLY THREE TIMES WHEREAS
2488  !  IMPLEMENTING THE FIRST METHOD WOULD REQUIRE LOADING THE
2489  !  ADDRESS REGISTER SIX TIMES.  ALSO IMPLEMENTING THE
2490  !  LINKED LIST IN LOCAL STORE SHOULD SHOW SOME IMPROVEMENT
2491  !  OVER THE MAIN MEMORY VERSION ALTHOUGH THE DATA TO
2492  !  SUBSTANTIATE THIS HAS NOT BEEN COLLECTED YET.
2493  !
2494  !
2495
2496   INSERT:
2497     P2-T,     D_A-PLUS-B,R1-A,CSPB(ONE),      !D<--BLINK(R1) ADDRESS.
2498     NEXT,     J/INS1
     6200  0 10011100 10011000 00010000 00000000 00110000 10000001
2499
2500   INS1:
2501     P3,       TMSPTR_(LOADANDREAD),           !INITIATE LOADANDREAD.
2502     NEXT,     J/INS2
     6201  0 10100000 00000000 00000001 10110000 00110000 10000010
2503
2504   INS2:
2505     NEXT,     J/INS3                          !NULL WORD ONE.
     6202  0 00000000 00000000 00000000 00000000 00110000 10000011
2506
2507   INS3:
2508     P3,       CSPB[MD]_BUSDIN,                !BLINK(R1) ARRIVES FROM LOCAL STORE.
2509     NEXT,     J/INS4
     6203  0 00001110 00000000 00000000 00001000 00110000 10000100
2510
```

```
2511  INS4:
2512    P2-T,    D_A,R2-A,                        !D<-- R2 FOR WRITE INTO BLINK(R1)
2513    NEXT,    J/INS5
     6204  0 11110000 10001010 00010000 00000000 00110000 10000101
2514
2515  INS5:
2516    P3,      TMSPTR_(WRITE),                  !INITIATE MOV R2,BLINK(R1)
2517    NEXT,    J/INS6
     6205  0 00100110 00000000 00000001 10110000 00110000 10000110
2518
2519  INS6:
2520    NEXT,    J/INS7                           !NULL WORD ONE.
     6206  0 00000000 00000000 00000000 00000000 00110000 10000111
2521
2522  INS7:
2523    P2-T,    D_A,R2-A,                        !D<-- FLINK(R2) ADDRESS
2524    NEXT,    J/INS8
     6207  0 11110000 10001010 00010000 00000000 00110000 10001000
2525
2526  INS8:
2527    P3,      TMSPTR_(LOADWRITEINC),           !INITIATE LOAD, WRITE AND INC
2528    NEXT,    J/INS9
     6210  0 00100000 10000000 00000001 10110000 00110000 10001001
2529
2530  INS9:
2531    P2-T,    D_A,R1-A,                        !D<-- R1 DATA FOR MOV R1,FLINK(R2)
2532    NEXT,    J/INS10
     6211  0 11110000 10011000 00010000 00000000 00110000 10001010
2533
2534  INS10:
2535    NEXT,    J/INS11                          !NULL WORD TWO.
     6212  0 00000000 00000000 00000000 00000000 00110000 10001011
2536
2537  INS11:
2538    P2-T,    D_B,CSPB(MD),                    !D<-- BLINK(R1) FETCHED EARLIER.
2539    NEXT,    J/INS12
     6213  0 10101110 00000000 00010000 00000000 00110000 10001100
2540
2541  INS12:
2542    P3,      TMSPTR_(WRITE),                  !INITIATE MOV BLINK(R1),BLINK(R2)
2543    NEXT,    J/INS13
     6214  0 00100110 00000000 00000001 10110000 00110000 10001101
2544
2545  INS13:
2546    NEXT,    J/INS14                          !NULL WORD ONE.
     6215  0 00000000 00000000 00000000 00000000 00110000 10001110
2547
2548  INS14:
2549    NEXT,    J/INS15                          !NULL WORD TWO.
     6216  0 00000000 00000000 00000000 00000000 00110000 10001111
2550
```

```
2551  INS15:
2552    P3,      TMSPTR_(LOADANDWRITE),          !INITIATE MOV R2,BLINK(R1)
2553    NEXT,    J/INS16                         !D REGISTER STILL OKAY.
     6217  0 00100100 00000000 00000001 10110000 00110000 10010000
2554
2555  INS16:
2556    P2-T,    D_A,R2-A,                       !D<-- R2 FOR WRITE.
2557    NEXT,    J/INS17
     6220  0 11110000 10001010 00010000 00000000 00110000 10010001
2558
2559  INS17:
2560    NEXT,    J/INS18                         !NULL WORD TWO.
     6221  0 00000000 00000000 00000000 00000000 00110000 10010010
2561
2562  INS18:
2563    NEXT,    BUT(SUBRB),PAGE(0),J/BRA05      !RETURN TO GET NEXT INSTRUCTION.
     6222  0 00000000 00000000 00000000 00000000 00111000 00000011
2564
2565
2566
2567  !
2568  !  THE REMOVE INSTRUCTION TAKES AN ENTRY OUT OF THE LINKED LIST.
2569  !  THE INPUT REGISTERS CONSIST OF:
2570  !  R1-      POINTER TO THE ENTRY TO BE DELETED.
2571  !  THE ALGORITHM USED TO REMOVE THE ENTRY FROM THE DOUBLY LINKED
2572  !  LIST IS EXPRESSED IN THE FOLLOWING MACRO CODE:
2573  !
2574  !  REMOVE: MOV  R2,-(SP)
2575  !          MOV  R3,-(SP)
2576  !          MOV  BLINK(R1),R3
2577  !          MOV  FLINK(R1),R2
2578  !          MOV  R2,FLINK(R3)
2579  !          MOV  R3,BLINK(R2)
2580  !          CLR  FLINK(R1)
2581  !          CLR  BLINK(R1)
2582  !          MOV  (SP)+,R3
2583  !          MOV  (SP)+,R2
2584  !
2585  !  HERE AGAIN ANALYZING THE ALGORITHM AND KNOWING WE ARE IN LOCAL
2586  !  STORE AND THAT WE HAVE SUCH FUNCTIONS AS READ TWO VALUES AT A
2587  !  TIME WE GET THE FOLLOWING ALGORITH IN A SHORTHAND NOTATION:
2588  !
2589  !          GRAB FLINK AND BLINK (R1)
2590  !          MOV  R2,FLINK(BLINK(R1))
2591  !          MOV  FLINK(R1),BLINK(FLINK(R1))
2592  !          CLR  FLINK AND BLINK (R1)
2593  !
2594  !  IN THIS METHOD WE ONLY REFERENCE LOCAL STORE FOUR TIMES WHEREAS
2595  !  USING THE ABOVE METHOD WE DO FIVE REFERENCES.
2596
2597
```

```
2598  REMOVE:
2599    P2-T,     D_A,R1-A,                        !D<-- ADDRESS OF ENTRY.
2600    NEXT,     J/REM1
     6223  0 11110000 10011000 00010000 00000000 00110000 10010100
2601
2602  REM1:
2603    P3,       TMSPTR_(LOADREADTWO),            !INITIATE READ OF FLINK AND BLINK.
2604    NEXT,     J/REM2
     6224  0 00100101 00001100 10000001 10110000 00110000 10010101
2605
2606  REM2:
2607    NEXT,     J/REM3                           !FIRST NULL WORD.
     6225  0 00000000 00000000 00000000 00000000 00110000 10010110
2608
2609  REM3:
2610    P3,       CSPB[MD]_BUSDIN,                 !FIRST DATA ITEM INTO MD.
2611    NEXT,     J/REM4
     6226  0 00001110 00000000 00000000 00001000 00110000 10010111
2612
2613  ! D HAS FLINK(R1)
2614  ! MD HAS BLINK(R1)
2615
2616  REM4:
2617    P2-T,     SR_A,D-NO-SHIFT,                 !SAVE FLINK(R1) INTO SR.
2618    NEXT,     J/REM5
     6227  0 11110000 01010000 00001000 00000000 00110000 10011000
2619
2620  REM5:
2621    P2-T,     D_B,CSPB(MD),                    !D<-- ADDRESS OF BACK FLINK.
2622    NEXT,     J/REM6
     6230  0 10101110 00000000 00010000 00000000 00110000 10011001
2623
2624  REM6:
2625    P3,       TMSPTR_(WRITE),                  !INITIATE WRITE.
2626    NEXT, J/REM7
     6231  0 00100110 00000000 00000001 10110000 00110000 10011010
2627
2628  REM7:                                        !FIRST NULL WORD.
2629    P2-T,     D_A,SR,                          !DATA TO CHANGE BACK FLINK
2630    NEXT,     J/REM8
     6232  0 11110000 00000000 00010000 00000000 00110000 10011011
2631
2632  REM8:                                        !DATA WRITTEN IN THIS CYCLE.
2633    P2-T,     D_A-PLUS-B,SR,CSPB(ONE),         !POINT TO FLINK OF FORWARD ENTRY.
2634    NEXT,     J/REM9
     6233  0 10011100 00000000 00010000 00000000 00110000 10011100
2635
```

```
2636  REM9:
2637    P3,       TMSPTR_(LOADANDWRITE),          !INITIATE WRITE.
2638    NEXT,     J/REM10
     6234  0 00100100 00000000 00000001 10110000 00110000 10011101
2639
2640  REM10:
2641    P2-T,     D_B,CSPB(MD),                   !DATA TO CHANGE FORWARD BLINK.
2642    NEXT,     J/REM11
     6235  0 10101110 00000000 00010000 00000000 00110000 10011110
2643
2644  REM11:
2645    P2-T,     D_A,R1-A,                       !D<-- ADDRESS OF ENTRY REMOVED.
2646    NEXT,     J/REM12
     6236  0 11110000 10011000 00010000 00000000 00110000 10011111
2647
2648  REM12:
2649    P3,       TMSPTR_(LOADWRITETWO),          !INITIATE CLEAR OF FLINK AND BLINK.
2650    NEXT,     J/REM13
     6237  0 10100000 10001100 10000001 10110000 00110000 10100000
2651
2652  REM13:
2653    P2-T,     D_ZERO,                         !ZERO DATA FOR FORWARD LINK.
2654    NEXT,     J/REM14
     6240  0 00110000 00000000 00010000 00000000 00110000 10100001
2655
2656  REM14:
2657    P2-T,     D_ZERO,                         !ZERO DATA FOR BACKWARD LINK.
2658    NEXT,     J/REM15
     6241  0 00110000 00000000 00010000 00000000 00110000 10100010
2659
2660  REM15:
2661    NEXT,     BUT(SUBRB),PAGE(0),J/BRA05      !RETURN TO GET NEXT INSTRUCTION.
     6242  0 00000000 00000000 00000000 00000000 00111000 00000011
2662
2663
2664
```

```
2665  !
2666  !  THE APPEND INSTRUCTION IS USED TO ADD AN ENTRY ONTO THE POINT
2667  !  WANTED.   THIS INSTRUCTION CAN BE USED IN TWO WAYS. (1) TO ADD
2668  !  THE ENTRY ONTO THE FRONT OF A QUEUE DIRECTLY AFTER THE HEAD
2669  !  ELEMENT AND ALSO TO ADD THE ELEMENT ONTO THE LAST POSITION OF
2670  !  A LIST.   THE FOLLOWING INPUT CONDITIONS OCCUR:
2671  !  R1-      POINTS TO HEAD OF LINKED LIST.
2672  !  R2-      POINTS TO ENTRY TO BE APPENDED TO THE LINKED LIST.
2673  !
2674  !  THE ALGORITHM CAN BE REPRESENTED IN THE FOLLOWING MACRO FORM:
2675  !
2676  !  APPEND: MOV R3,-(SP)
2677  !          MOV FLINK(R1),FLINK(R2)
2678  !          MOV FLINK(R1),R3
2679  !          MOV R2,FLINK(R1)
2680  !          MOV BLINK(R3),BLINK(R2)
2681  !          MOV R2,BLINK(R3)
2682  !          MOV (SP)+,R3
2683  !
2684  !  ANALYZING THE ALGORITHM AGAINST THE CONSTRAINTS AND THE
2685  !  POWERS OF USING A LOCAL STORE REPRESENTATION OF THE LINKED
2686  !  LIST THE FOLLOWING SHORTHAND NOTATION ALGORITHM IS DEVELOPED.:
2687  !
2688  !          GRAB FLINK(R1)
2689  !          MOV  R2,FLINK(R1)
2690  !          GRAB BLINK(FLINK(R1))
2691  !          MOV  R2,BLINK(FLINK(R1))
2692  !          MOV  FLINK(R1),FLINK(R2)
2693  !          MOV  BLINK(FLINK(R1)),BLINK(R2)
2694  !
2695  !  USING THIS REPRESENTATION THE WCS ADDRESS REGISTER IS LOADED ONLY
2696  !  FOUR TIMES COMPARED TO THE SEVEN OR SO TIMES FOR THE FIRST
2697  !  METHOD.
2698
2699
2700  APPEND:
2701    P2-T,     D_A,R1-A,                          !D<-- ADDRESS OF APPEND PT.
2702    NEXT,     J/APP1
     6243  0 11110000 10011000 00010000 00000000 00110000 10100100
2703
2704  APP1:
2705    P3,       TMSPTR_(READ),                     !INITIATE READ OF FLINK(R1).
2706    NEXT,     J/APP2
     6244  0 10100101 00000000 00000001 10110000 00110000 10100101
2707
2708  APP2:
2709    NEXT,     J/APP3                             !NULL WORD ONE.
     6245  0 00000000 00000000 00000000 00000000 00110000 10100110
2710
```

```
2711  APP3:
2712    P2-T,     D_A,R2-A,                          !PUT R2 INTO D FOR WRITE.
2713    P3,       CSPB[MD]_BUSDIN,                   !INPUT FLINK(R1).
2714    NEXT,     J/APP4
     6246  0 11111110 10001010 00010000 00001000 00110000 10100111
2715
2716  APP4:
2717    P3,       TMSPTR_(WRITE),                    !INITIATE MOV R2,FLINK(R1).
2718    NEXT,     J/APP5
     6247  0 00100110 00000000 00000001 10110000 00110000 10101000
2719
2720  APP5:
2721    P2-T,     SR_B,CSPB(MD),                     !SR GETS FLINK(R1)
2722    NEXT,     J/APP6                             !NULL WORD ONE.
     6250  0 10101110 00000000 00001000 00000000 00110000 10101001
2723
2724  APP6:
2725    P2-T,     D_A-PLUS-B,SR,CSPB(ONE),           !CALCULATE BLINK(FLINK(R1)).
2726    NEXT,     J/APP7
     6251  0 10011100 00000000 00010000 00000000 00110000 10101010
2727
2728  APP7:
2729    P3,       TMSPTR_(LOADANDREAD),              !INITIATE READ OF BLINK(FLINK(R1)).
2730    NEXT,     J/APP8
     6252  0 10100000 00000000 00000001 10110000 00110000 10101011
2731
2732  APP8:
2733    NEXT,     J/APP9                             !NULL WORD ONE.
     6253  0 00000000 00000000 00000000 00000000 00110000 10101100
2734
2735  APP9:
2736    P2-T,     D_A,R2-A,                          !PUT R2 INTO D FOR WRITE.
2737    P3,       CSPB[MD]_BUSDIN,                   !INPUT BLINK(FLINK(R1)).
2738    NEXT,     J/APP10
     6254  0 11111110 10001010 00010000 00001000 00110000 10101101
2739
2740  APP10:
2741    P3,       TMSPTR_(WRITE),                    !INITIATE MOV R2,BLINK(FLINK(R1))
2742    NEXT,     J/APP11
     6255  0 00100110 00000000 00000001 10110000 00110000 10101110
2743
2744  APP11:
2745    NEXT,     J/APP12                            !NULL WORD ONE.
     6256  0 00000000 00000000 00000000 00000000 00110000 10101111
2746
2747  APP12:
2748    P2-T,     D_A,R2-A,                          !D<-- ADDRESS OF FLINK(R2).
2749    NEXT,     J/APP13
     6257  0 11110000 10001010 00010000 00000000 00110000 10110000
2750
```

```
2751  APP13:
2752    P3,       TMSPTR_(LOADWRITEINC),          !INITATE MOV FLINK(R1),FLINK(R2)
2753    NEXT,     J/APP14
     6260  0 00100000 10000000 00000001 10110000 00110000 10110001
2754
2755  APP14:
2756    P2-T,     D_A,SR,                         !FLINK(R1) INTO DO FOR WRITE.
2757    NEXT,     J/APP15
     6261  0 11110000 00000000 00010000 00000000 00110000 10110010
2758
2759  APP15:
2760    P2-T,     D_B,CSPB(MD),                   !BLINK(FLINK(R1)) INTO D FOR WRITE.
2761                                      !(DATA ABOVE WRITTEN AND THEN INCREMENT
2762    NEXT,     J/APP16                         !ADDRESS REGISTER TO POINT TO BLINK)
     6262  0 10101110 00000000 00010000 00000000 00110000 10110011
2763
2764  APP16:
2765    P3,       TMSPTR_(WRITE),                 !MOV BLINK(FLINK(R1)),BLINK(R2)
2766    NEXT,     J/APP17
     6263  0 00100110 00000000 00000001 10110000 00110000 10110100
2767
2768  APP17:
2769    NEXT,     J/APP18                         !NULL WORD ONE.
     6264  0 00000000 00000000 00000000 00000000 00110000 10110101
2770
2771  APP18:
2772    NEXT,     J/APP19                         !NULL WORD TWO.
     6265  0 00000000 00000000 00000000 00000000 00110000 10110110
2773
2774  APP19:
2775    NEXT,     BUT(SUBRB),PAGE(0),J/BRA05      !RETURN TO GET NEXT INSTRUCTION.
     6266  0 00000000 00000000 00000000 00000000 00111000 00000011


 MIC -- ERRORS DETECTED:      0
 MIC -- NUMBER OF LINES PROCESSED:   2775
```

.

APPENDIX E

ERROR MESSAGES

This appendix contains the error messages for the microprogramming tools. Error messages are given first for the assembler, then for the debugger, and finally for the command language interpreter.

In addition to the error messages listed here, additional error messages can arise from any of the following sources:

- Operating System. An operating system error message has the form.

  FCS number file-name error message

  For an explanation of operating system error messages the programmer is referred to:

  FORTRAN IV PLUS User's Guide (DEC-11-LFPUA-BD), Appendix C, Section C.2.3

- FORTRAN Run-Time System. A FORTRAN error message has the form:

  FCS number file-name error message

  For an explanation of FORTRAN error messages, the programmer is referred to:

  FORTRAN IV PLUS User's Guide (DEC-11-LFPUA-BD), Appendix C, Section C.2.2

- Program Errors. Error messages that are reported as a result of the failure of consistency checks within the microprogramming tools have the following form:

  PROGRAM ERROR - error-message

  These errors are described within the program documentation. The occurence of such an error indicates a malfunction that is outside the programmer's control.

E.1 MICRO-11/60 ERROR MESSAGES

The following error messages are produced by the assembler. The first nine errors are fatal errors. These errors indicate a problem with the hardware or software support for the MICRO-11/60 assembler program. For each of these errors a suggested procedure is indicated.

1. WRITE ERROR IN WORK FILE

   Suggested procedure: Try again.

2. INTERNAL BUFFER ERROR

   Suggested procedure: Try again in a less active environment.

3. (reserved)

4. WORK FILE TOO BIG

   Suggested procedure: Try to break down the microprogram either by dividing into modules or by removing comments.

5. READ ERROR IN WORK FILE

   Suggested procedure: Check disk. Try again in a less active environment.

6. INTERNAL INITIALIZATION ERROR

   Suggested procedure: Try running in a different spot in memory.

7. END OF OBJECT FILE ERROR

   Suggested procedure: Try again.

8. WRITE ERROR IN OBJECT FILE

   Suggested procedure: Try again.

9. (reserved)

The remaining errors are non-fatal. After the detection of an error, the assembly continues. For each of these errors, a Probable cause is indicated.

10. ILLEGAL NUMERIC LABEL

    Probable cause: A label outside the legal limits of the program.

11. BEGIN BLOCK ALREADY ENDED

    Probable cause: A case-microinstruction or an end-definition seen for a branch-label that has already been ended by an end-definition.

12. CASE NUMBER TOO LARGE

    Probable cause: A case number larger than the number calculated by taking 2**k, where k is the number of 0's in the mask associated with the branch-label.

13. THIS CASE ALREADY HANDLED

    Probable cause: A case number for a given branch-label is given more than once in the microprogram.

14. DATA SET ERROR

    Probable cause: The internal stack is too big.

15. ILLEGAL RANGE

    Probable cause: The address range given with a branch-definition is in the wrong order or is outside the legal limits of the program.

16. (reserved)

17. ILLEGAL USE OF SYMBOL

    Probable cause: A symbol given in a place where only a numeric value is acceptable.

18. ILLEGAL CONDITION BEFORE .CODE

    Probable cause: A language construct that can only be given in the action-part of the program appears in the definition part.

19. ILLEGAL VALUE

    Probable cause: A value outside the legal range or a signed value.

20. PAGE BOUNDS ERROR IN DEFAULT ADDRESS

21. NO .ADDRESS KEYWORD

Note: This error can occur only if this predefinitions file is not used.

Probable cause: The .ADDRESS keyword, as described in Appendix B, is not present.

22. ILLEGAL STATEMENT AFTER .CODE

Probable cause: A language construct that can be given only in the definition part of the program is used in the action part.

23. MACRO EXPANSION ERROR

Probable cause: The arguments of the macro-call created a problem in the expansion of the macro-body.

24. MICRO-INSTRUCTION ILLEGAL

Probable cause: A name given in the microinstruction is undefined. This error sometimes indicates a problem with the definition of the name.

25. INTERNAL STACK OVERFLOW

Probable cause: The stack associated with macro expansion has too many entries. This error can be caused by nesting macros too deeply or by supplying too many arguments for a macro.

26. ATTEMPT TO REWRITE BIT IN MICROWORD

Probable cause: A field-setting is given that sets a bit already set in the microword by another field-setting.

27. CONSTRAINT FIELD PARAMETER ILLEGAL

Probable cause: The number of 0's in a mask is either less than 1 or greater than the allowable number (7).

28. CANNOT SATISFY CONSTRAINT REQUEST

    Probable cause: A set of addresses to satisfy the constraint request cannot be found.

    Suggested procedure: Move the branch-definition to the beginning of the program so that the necessary addresses can be reserved.

29. ADDRESS ALREADY SEEN

    Probable cause: The address has already been either explicitly allocated by the programmer or selected by the assembler for allocation.

30. ALREADY DEFINED
    Probable cause: The name has already been defined for the same name type.

31. ILLEGAL FIELD MODE

32. ILLEGAL PASS 2 OPERATION

    Probable cause: System or hardware failure.

33. BAD INITIALIZATION

    Probable cause: System or hardware failure.

34. (reserved)

35. ERROR ROUTINE FAILURE

    Probable cause: System or hardware failure.

36. SYMBOL ALREADY USED AS A LABEL

    Probable cause: Symbol has been already been defined by its use as a label.

37. (reserved)

38. PRE-SCAN ERROR

    Probable cause: System or hardware failure.

39. SYNTAX ERROR

    Probable cause: The source line does not have the correct syntax.

E.2 MDT ERROR MESSAGES

The debugger error messages and the error source are listed below.

Error		Source
ADDRESS ERROR AT address		Run-time
ATTEMPT TO EXECUTE AN ILLEGAL INSTRUCTION AT address		Run-time
ILLEGAL BREAKPOINT INSTRUCTION AT address		Run-time
UNKNOWN TRAP OCCURRED AT address		Run-time
ATTEMPT TO LOAD A CODE SEGMENT FAILED AT ADDRESS address		Load
BREAKPOINT NUMBER breakpoint-id IS NOT SET		Breakpoint
BREAKPOINT TABLE FULL		Breakpoint
DID NOT FIND ADDRESS		Breakpoint
DISPLAY POINT NUMBER display-id IS NOT SET		Display
ILLEGAL	ADDRESS	Proceed, Go, Open
	ADDRESS TYPE	Breakpoint
	BIT NUMBER	Open
	BREAK COMMAND	Breakpoint
	BREAKPOINT NUMBER	Breakpoint
	NUMBER, TOO LARGE	Display Point
	TERMINATION	(all)
INCORRECT	BIT FIELD NAME	(all)
	STATE VARIABLE NAME	(all)
INPUT ERROR		(all)
LEGAL BREAKPOINT DID NOT OCCUR		Proceed, Go
NO FREE ENTRIES IN DISPLAY TABLE		Display
ODD MACRO ADDRESS		Breakpoint
SYNTAX ERROR		(all)
UNABLE TO	LOAD DISPATCH TABLE	Load
	READ SPECIFIED LOCATION	Open
	WRITE SPECIFIED LOCATION	Open
UNKNOWN TERMINATOR		Open

E.3 COMMAND LANGUAGE ERROR MESSAGES

The following error messages are command language interpreter error messages. These error messages may be encountered when trying to use the MICRO-11/60 assembler.

1. COMMAND LINE SYNTAX ERROR rest-of-line

   If the command language interpreter detects an error in the command line, it prints this message and the part of the line after the point at which the error was detected.

2. COMMAND SWITCH ERROR
   where: n = 1 implies the object module file
   n = 2 implies the list file
   n = 4 implies the first input file

   The command language interpreter found an illegal switch on the indicated file.

.

INDEX

READER'S COMMENTS

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

Did you find errors in this manual? If so, specify by page.

_____

_____

_____

_____

_____

_____

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

_____

_____

_____

_____

_____

_____

Is there sufficient documentation on associated system programs required for use of the software described in this manual? If not, what material is missing and where should it be placed?

_____

_____

_____

_____

_____

_____

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

- ☐ Assembly language programmer
- ☐ Higher-level language programmer
- ☐ Occasional programmer (experienced)
- ☐ User with little programming experience
- ☐ Student programmer
- ☐ Non-programmer interested in computer concepts and capabilities

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, MASS.

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

Postage will be paid by:

digital

Software Documentation
146 Main Street ML5-5/E39
Maynard, Massachusetts 01754