PERKIN-ELMER

OS/32 SYSTEM MACRO LIBRARY

Reference Manual

48-006 F00 R02

The Perkin-Elmer Corporation, Data Systems Group, 2 Crescent Place, Oceanport, New Jersey 07757

TABLE OF CONTENTS

CHAPTERS (Continued)

CHAPTERS (Continued)

CHAPTERS (Continued)

CHAPTERS (Continued)

CHAPTERS (Continued)

CHAPTERS (Continued)

CHAPTERS (Continued)

PREFACE

This reference manual details the OS/32 System Macro Library. The user should have an in-depth knowledge of the OS/32 Application Level Programmer Reference Manual and the CAL Macro Processor and Macro Library Utility Reference Manual.

Chapter 1, Overview of the System Macro Library, explains macro instructions, parameters, parameter field value mnemonics, macro expansion errors, constructing parameter blocks, and error handling and recovery. The remaining seven chapters of this manual explain the formats, parameter values, default values, required parameters, programming considerations, examples, and messages for all supervisor, task management, input/output (I/O), file management, timer management, and miscellaneous macros. Chapter 7 is a new chapter that explains the macros used for the Perkin-Elmer Multiprocessor System (Model 3200MPS). This chapter applies to the Model 3200MPS System only. The previous Chapter 7 is now Chapter 8.

This manual is intended for use with the OS/32 R07.1 software release or higher. Additional material specifically related to the Model 3200MPS System has been included and is supported by the OS/32 R07.1 software release and higher. Throughout the text these features are identified as applicable only to the Model 3200MPS System.

For further information on the contents of all Perkin-Elmer 32-bit manuals, see the 32-Bit Systems User Documentation Summary.

CHAPTER 1
OVERVIEW OF THE SYSTEM MACRO LIBRARY

1.1 INTRODUCTION

This chapter explains macro instructions, parameters, parameter field value mnemonics, macro expansion errors, constructing parameter blocks, and error handling and recovery.

1.2 MACRO INSTRUCTIONS

A macro instruction, a single instruction that expands to a series of instructions, is written like an assembler instruction; but the output, when processed by the Common Assembler Language (CAL) Macro Processor, is in assembly language. The output of a macro instruction can be machine instructions, another macro instruction, assembler instructions, or a combination of these instructions.

1.2.1 Macro Instruction Formatting

As with CAL instruction statements, macro instruction statements are written in fixed or free format. A macro instruction statement of either format has five separate fields:

- Name
- Operation
- Operand
- Comment
- Identification/sequence

1.2.1.1 Fixed Formatting of Macro Instruction Statements

In fixed formatting, the macro instruction fields are normally defined as:

COLUMNS	DEFINITION
1 - 8	Name field
9	Blank
10 - 14	Operation field
15	Blank
16 - n	Operand field
n+1	Blank
n+2 - 71	Comment field*
72	Continuation
73 - 80	Identification/sequence field

* Wherever possible, the comment field begins at column 36.

The operand field can be continued by placing a nonblank character into column 72. The operand field and the comment field are variable in length; the first blank encountered after column 16 delimits these two fields. Due to how the output listings are tabulated, the comment field cannot contain more than 37 characters; if it does, only the first 37 characters appear on the listing.

1.2.1.2 Free Formatting of Macro Instruction Statements

In free formatting, blanks delimit the five separate fields. For example, if the name field is not used, a blank character in column 1 indicates the start of the operation field. Similarly, the first blank following the MACRO code in the operation field indicates the start of the operand field. As in fixed formatting, the first blank within the operand field indicates the start of the comment field. Free formatting has three restrictions:

- Comment length, including blanks, is limited to 37 characters.
- Unless the operand field is being continued, column 72 must contain a blank.
- If present, the identification/sequence field must start in column 73.

1.2.2 Macro Instruction Fields

The following paragraphs detail the fields for free and fixed format macro instructions:

- name field
  - The name field contains a symbol or blanks. A symbol, which must begin in column 1, is the name associated with the first executable instruction or the first byte of a generated table of constants. The name field is optional with most instructions; however, with some instructions, it is required. The symbol can be any valid CAL symbol, but it must not begin with an @ sign because system macros use this character as an internal symbol.
- operation field
  - The operation field contains a mnemonic operation code for a macro instruction. This mnemonic operation code is a string of not more than 8 alphanumeric characters. The first character must be alphabetic.
- operand field
  - The operand field contains blanks, or one or more operands separated by commas. Blanks cannot be embedded within operands, except when enclosed by apostrophes. The operand field can be continued by placing a nonblank character in column 72.
- comment field
  - The comment field follows the operand field, separated by at least one blank column. This field contains user comments.
- identification/sequence field
  - The identification/sequence field occupies columns 73 through 80. The user has the option of identifying and maintaining the sequence of the source field.

1.3 PARAMETERS

Three types of parameters (positional, keyword, and mixed mode) can be used in the operand field of a macro instruction. The following sections explain each parameter type.

1.3.1 Positional Parameters

Postional parameters have a particular position within the operand field. Positional parameters are represented with lowercase characters; the user must supply a value or expression for the parameters. Positional parameters are processed by the macro processor according to their positions, and positions are maintained by separating commas. When entering or omitting positional parameters, supply the separating commas to mark the position of succeeding parameters. For example:

```
    [symbol]  MACRO  oper1,oper2,oper3
```

In this example, the parameter field consists of three positional parameters. These are processed by position, left-to-right. If omitting:

- oper2, then write oper1,,oper3
- oper1 and oper2, then write ,,oper3
- the last parameter(s), then write either oper1 OR oper1, OR oper1,,
- all parameters, then leave blanks or insert commas

1.3.2 Keyword Parameters

Keyword parameters have no particular position within the operand field. They consist of a keyword, immediately followed by an equal sign (=) and a keyword value. Since the keyword uniquely defines the parameter to the macro processor, the user can write keyword parameters in any sequence within the parameter field. If keyword parameters are omitted, their separating commas are also omitted. Uppercase characters represent keyword parameters. For example:

```
    [symbol] MACRO KEYWORD1=expression,KEYWORD2=value
```

1.3.3 Mixed Mode Parameters

An operand field is referred to as mixed mode when it contains positional and keyword parameters. In mixed mode parameter fields, all positional parameters must precede keyword parameters:

```
    [symbol] MACRO oper1,oper2,KEYWORD1=A,KEYWORD2=B
```

The previously discussed rules for omitting positional and keyword parameters also apply to mixed mode parameter fields:

- If omitting oper1 and KEYWORD2=B, write: ,oper2,KEYWORD1=A
- If omitting all positional parameters, then write: ,,KEYWORD1=A,KEYWORD2=B or KEYWORD1=A,KEYWORD2=B

Parameters are represented by lowercase or uppercase characters. Lowercase characters represent positional parameters or expressions, the values of which the user must supply. Uppercase characters represent keyword parameters or option codes, which the user must enter as shown.

1.4 PARAMETER FIELD VALUE MNEMONICS

A value mnemonic is a lowercase abbreviation appearing in the parameter field and representing an address, expression, or value. The user must supply the actual value.

When macro instruction formats are presented in subsequent sections of this manual, value mnemonics appear as parameter values. Refer to this section for the description of these value mnemonics:

- addr

  - The addr mnemonic represents a valid address-expression that CAL evaluates at assembly time. The address that addr references cannot be indexed.

    Example:

    ```
        [symbol] MACRO dest-addr,source-addr
    ```

    Where the user might supply one of these address-expressions: A, LOOP, A-B, A-16, or A+4.

- addrx

  - The addrx mnemonic represents a valid indexable address-expression that can be partially evaluated at assembly time and (by adding the INDEX register) at execution time. Targeted 32-bit assemblies permit double-indexing.

    Example:

    ```
    [symbol] MACRO dest-addrx,PCB=addrx
    ```

    Where the user might supply one of these indexable address-expressions: A, A+B, A(5), A+B(5), or 0(5).

- reg

  - The reg mnemonic represents a register-expression that CAL can evaluate to a value between 0 and 15, corresponding to one of the 16 general registers. By supplying a register-expression for the reg mnemonic, the user designates one of the general registers to be used for a special function.

    Example:

    ```
    [symbol] MACRO pointer-reg,reg-reg
    ```

    Where the user might specify one of these register-expressions: 2, 0, R2, or R0.

- (reg)

  - The (reg) mnemonic represents a register-expression, enclosed in parentheses that CAL can evaluate to a value between 0 and 15. The register must contain the parameter to be supplied to the macro.

    Example:

    ```
    [symbol] MACRO dest-(reg),PCB=(reg)
    ```

    Where the user might specify one of these register-expressions: (3), (R3), (7), or (R7).

- abs address exp

  - The abs address exp mnemonic represets a return code specifying the condition code that the operating system returns after a task is terminated. The condition code can refer to a parameter block size field; the bytes of temporary storage released by a RELSTORE macro; a 4-byte condition code setting in the program status word (PSW); or the length (maximum 63) of a number to be converted by the UNPK macro.

    Example:

    ```
    [symbol] UNPK num, dest [LEN=60]
    ```

- absexp

  - The absexp mnemonic represents an absolute expression that CAL is to evaluate. It can be a byte or halfword expression. If it is a byte expression, the mnemonic is abs byte exp. If it is a halfword expression, the mnemonic is abs halfword exp. It is evaluated by a load immediate instruction; so, it can contain an INDEX register.

    Example:

    ```
    [symbol] MACRO size-absexp,LEN=absexp
    ```

    Where the user might specify one of these absolute expressions: 3, X'FF', or C'ABC'.

- 'string'

  - The 'string' mnemonic represents any string of characters enclosed in apostrophes.

    Example:

    ```
    [symbol] MACRO msg-'string'
    ```

    Where the user might supply this message: 'MY NAME IS O''BRIEN'. (An apostrophe is specified as two consecutive apostrophes.)

SYSVOL	The SYSVOL mnemonic specifies the volume name field of a packed file descriptor (fd).
SYSVOLB	The SYSVOLB mnemonic specifies the volume name field of a packed fd with leading blanks skipped.
NOVOL	The NOVOL mnemonic specifies that no volume name is to be moved into the volume name field of the packed fd. The default volume name is moved into the packed fd.
NOVOLB	The NOVOLB mnemonic specifies that no volume name and, therefore, the default volume name with leading blanks skipped, is to be moved into the packed fd.
SYSVOL	The SYSVOL mnemonic specifies the name of the spool volume to be moved into the spool volume name field of the packed fd.
SYSVOLB	The SYSVOLB mnemonic specifies the name, with leading blanks skipped, of the spool volume name field of the packed fd.
OD	This abbreviation stands for other-directed assignment of a logical processing unit (LPU) number, where a task's LPU number is assigned by another task. This option is used with the DIR parameter of the Model 3200MPS System SETLPU macro and others as well as with other non-3200MPS System macros.
	Example:
	`[symbol] SETLPU tmpcb, DIR=0`
	In the example, the SETLPU macro points to the previously created tmpcb parameter block and specifies that the task's LPU number is to be set by another task.
SD	This abbreviation specifies self-directed LPU number assignment. It means that the task is to assign its own LPU number. See Chapter 7 for an explanation of the Model 3200MPS System macros.
INT	This abbreviation specifies a table of intervals in milliseconds from midnight. It is used with the options parameter of the time management macros.

Example:

```
[symbol] CYCTIME NUMBINIT=reg, OPT=INT
```

TOD	This abbreviation specifies a table of intervals in seconds from midnight. It is used with the option parameter of timer management macros. See Chapter 6.
D	This abbreviation specifies decimal. DB specifies decimal, skip leading blanks. It is used with the options parameter of the PACK macro and other macros. See Section 2.10.
H	This abbreviation specifies a hexadecimal number. HB specifies hexadecimal, skip leading blanks. It is used with the option parameter of the PACK macro and with other macros.
L	This abbreviation specifies a list form-only, build a parameter control block (PCB). It is used with the form parameter of the CANTIME macro and other timer management macros. See Chapter 6.

1.5 MACRO EXPANSION ERRORS

Depending on their cause, macro expansion errors can be grouped into several categories. The return code at macro processor end of task determines the source of error.

- return code 0

  - A return code of 0 indicates that no errors or warnings were detected. (CAL can later detect such errors as invalid operation codes or undefined symbols.)

- return code 1

  - A return code of 1 is a warning from one or more system macros. An MNOTE in the listing determines the default action taken. The usual case is an invalid code.

- return code 2

  - A return code of 2 indicates that the macro processor detected an error. Check the syntax of the macro; the most common error is an invalid keyword.

- return code 4

  - A return code of 4 means the macro detected the error. Look for an MNOTE in the listing to determine the cause. The most common error is an omitted required parameter.

- execution time error messages

  - The nonproceed input/output macros produce error messages at execution time if an error occurs and the task pauses. The operator can take appropriate action and continue. The task continues at the RESTART address.

  - The file management macros also produce execution time error messages. The ERR parameter, which specifies a table of addresses that the FMERRTBL macro built, traps each of these errors. Any nontrapped error produces a message. Depending on the PAUS flag in the error table, the task will or will not pause. Whether or not the task pauses, it continues at the RESTART address. If the task pauses for either cause, the state of the user registers is:

REGISTERS	STATUS
R0-R13	same as when macro was issued
R14	pointer to PCB
R15	undefined

Upon continuation, R15 is restored to its original value.

1.6 CONSTRUCTING PARAMETER BLOCKS

Parameter blocks can be constructed in several ways depending on the type of macro. Because supervisor macros use small parameter blocks of different formats and sizes, it is more efficient to construct a block for each call. Because I/O, file management, and task management macros use larger parameter blocks, it is more efficient to construct the block once and reuse it for different calls, modifying fields as required. Special macros generate these blocks.

1.6.1 Parameter Blocks for Supervisor Macros

Supervisor macros require miscellaneous services from the operating system. Two mutually exclusive parameters, PCB= and FORM=, control parameter block contruction for supervisor macros.

1.6.1.1 Omitting the PCB= and FORM= Parameters

When omitting both the PCB= and FORM= parameters:

- If called from within a PURE segment, the macro switches to an IMPURE segment, constructs the parameter block, and returns to the PURE segment.
- If called from within an IMPURE segment, the macro constructs the parameter block and branches around it.

In both cases, R14 (or the PCBREG from the ENVIRON macro) is set pointing to the parameter block. CAL constructs any other parameters that were coded as constants. Any other parameters coded as indexed expressions or registers containing values are constructed as zeros and code is generated to modify the parameter block at execution time using R15 (or SCRREG from the ENVIRON macro). Examples are:

- FETDATE ALPHA
  - constructs a FETDATE parameter block with an address constant ALPHA, sets R14 pointing to the parameter block, and executes the SVC.
- FETDATE ALPHA(R2)
  - constructs a FETDATE parameter block, generates an LDAI R15 and ALPHA(2) and a store into the block, and executes the SVC.
- FETDATE (R2)
  - generates a FETDATE parameter block, stores R2 into the block, and executes the SVC.

1.6.1.2 Code FORM=L (List Form)

Only the parameter block is constructed. A label, if specified in the NAME field, is associated with the first byte of the aligned parameter block. Any other parameters coded as constants are filled in as constants by CAL. Any other parameters coded as indexed expressions or registers containing values are ignored.

Examples:

- BETA FETDATE ALPHA,FORM=L

  - constructs a FETDATE parameter block with the address constant ALPHA. The symbol BETA is associated with the first byte of the parameter block.

- GAMMA FETDATE FORM=L

  - constructs a FETDATE parameter block with zero in the address field. This parameter block can be referenced by another FETDATE macro by coding PCB=GAMMA.

1.6.1.3 Code PCB= (Execute Form)

Code PCB= executes a remote parameter block at the address that the PCB parameter block specifies. Any other parameters; whether they are constants, indexed expressions, or registers containing values; generate code to store into the parameter block and then execute the SVC. R14 is set to point to the parameter block. If PCB= (reg) is coded, that register is used as the parameter block pointer.

Example:

```
    FETDATE PCB=BETA
```

Since the parameter block at BETA had the address field coded, R14 is set to point to the block and the SVC is executed.

Example:

```
    FETDATE ALPHA,PCB=GAMMA
```

R14 is set to point to the parameter block at GAMMA; ALPHA is stored into the address field; and the SVC is executed.

Example:

```
    LDAI R3,GAMMA
    FETDATE ALPHA,PCB=(R3)
```

The address ALPHA is stored in the parameter block that R3 points to and the SVC is executed. A subsequent call of FETDATE, using GAMMA as the parameter block address, has the address ALPHA already stored in the block.

Example:

```
    FETDATE PCB=GAMMA
```

Once set in a parameter block, it remains unchanged until a subsequent macro modifies the field.

1.6.2 Parameter Blocks for Input/Output, File Management, Task Management, and Timer Management Macros

The first positional parameter controls the construction of the parameter blocks for the input/output, file management, and task management macros.

- If the first positional parameter is coded, it is assumed that the IOPCB, FMPCB, or TMPCB macros built the parameter block. Several macros can use the same parameter block providing only parameters change from call to call. Omitted operands are left unchanged or revert to default values.
- If the macro is in a PURE segment and the first positional parameter is omitted, the macro switches to an IMPURE segment, constructs the parameter block, and returns to the PURE segment. If the macro is in an IMPURE segment and the first positional parameter is omitted, the macro constructs the parameter block and branches around it.

In both cases, R14 (or the PCBREG from the ENVIRON macro) is set pointing to the parameter block. Any other parameters coded as constants are constructed as constants by CAL. Any other parameters coded as indexed expressions or registers containing values are constructed as zeros and code is generated to modify the parameter block at execution time using R15 (or the SCRREG from the ENVIRON macro).

Example:

```
    REWIND LU=2
```

The parameter block IOPCB is constructed as described and the logical unit (lu) field is set to 2. The device attached to lu2 is rewound.

```
    READ   LU=1,ADDR=BUFFER,RECL=256
    LDA    2,IO.TRANS(R14)
```

In this example, the parameter block IOPCB is constructed with the appropriate fields filled in. The READ SVC is then issued with R14 pointing to the parameter block. The actual number of transferred bytes can be found at offset IO.TRANS past R14. The IO., FM., or TM. STRUCs are constructed as part of the appropriate macros.

Example:

```
    IOPARBLK IOPCB   ADDR=BUFFER,RECL=80
    BUFFER   DS      80
             .
             .
             .
    LOOP     READ    IOPARBLK,LU=1
             WRITE   IOPARBLK,LU=2
             B       LOOP
             .
             .
             .
```

In this example, the parameter block is constructed with the IOPCB macro. The ADDR and RECL fields are filled in, while the lu field is set to zero because it was omitted. The READ macro specifies IOPARBLK as the first positional parameter. The lu field is modified; the ADDR and RECL fields are left unchanged. Care must be exercised in reusing parameter blocks after modifying fields that were defined when the block was first constructed.

Example:

```
    IOPARBLK IOPCB LU=1,...
             .
             .
             .
    LOOP     READ  IOPARBLK
             WRITE IOPARBLK,LU=2
             B     LOOP
             .
             .
             .
```

The READ macro uses 1 in the LU field because it was defined in the IOPCB macro. The WRITE macro modifies the LU field and writes to LU 2. When READ is reexecuted, the LU field has been modified and attempts to READ from LU 2.

NOTE

> As a general principle, do not modify fields that have been defined as constants.

This example is a more subtle representation of this principle:

```
    LOOP     READ  LU=1,...
             WRITE O(R14),LU=2
             B     LOOP
             .
             .
             .
```

The READ macro constructs the parameter block; the LU field is set as a constant, and R14 points to the block. The first READ executes as desired. WRITE reuses the same parameter block using R14 as a pointer to it and modifying the LU field. When READ is reexecuted, the LU field is not modified because it was constructed as a constant.

Example:

```
             LIS   R3,1
    LOOP     READ  LU=(R3),...
             WRITE O(R14),LU=2
             B     LOOP
             .
             .
             .
```

Even though the READ macro constructs the parameter block, the LU field is not defined until execution time because it is coded as a register. In this case, the LU field is modified for the READ and WRITE macros as desired.

1.7 ERROR HANDLING AND RECOVERY

When an I/O or file management operation has completed, the operating system writes a status code into the parameter block. A zero status indicates the operation was performed successfully; a nonzero status means that an error was detected and the operation was not successful.

When an I/O or file management macro is first encountered, a subroutine is generated to check the status and take action. A zero status results in the next instruction being executed. If a nonzero status is detected, the ERR, PAUS, and RESTART parameters determine the following sequence of events.

The ERR parameter specifies the address of a table that the IOERRTBL or FMERRTBL macros built. The entries in this table specify branch addresses of user-written routines to handle each specific error. The ERR parameter can be specified when the parameter block is constructed or in a macro. If the ERR parameter is specified in a macro, it replaces the address in the block.

Example:

An ALLOCATE macro is issued and the file already exists. Since the file exists, the program wants to delete the file:

```
          ALLOCATE  FD='FILE1',ERR=NAMERR,...
           .
           .
           .
DELETE    EQU    *
          FMERRET   12
          DELETE    (14)
          ALLOCATE  (14)
          BR        12
NAMERR    FMERRTBL  NAME=DELETE
```

When detecting the NAME error, the program branches to the label DELETE. In the DELETE routine, the program then deletes the file by using the parameter block pointed to by R14 because this parameter block contains the filename FILE1. The file can be reallocated by using the same parameter block and returning to the instruction following the original ALLOCATE macro.

If the ERR parameter is omitted from the parameter block or an error occurs for which there is no entry in the table, then:

- An error message is written to the log device.
- The task does or does not pause depending on the PAUS= parameter:
  - PAUS=N does not pause the task.
  - Omitting the PAUS= parameter causes the task to pause.
- The task resumes execution at the RESTART= address. If RESTART= is omitted:
  - Input/output macros retry the operation.
  - File management macros restart at the next instruction.

The RESTART parameter cannot be specified in the IOPCB or FMPCB macros. If the task pauses, all registers except R14 and R15 contain the values prior to the macro. R14 points to the parameter block. R15 is undefined.

1.8 MACROS IN CONDITIONAL ASSEMBLY

CAL conditional assembly, such as IFZ, IFNZ, cannot be evaluated at macro processing time because the values of the EQUs are not known to the CAL macro processor. Therefore, any macros within conditional code will always be expanded regardless of whether CAL will actually generate the expanded code. Usually, this would not be a problem since CAL would not generate the expanded code if the conditional failed. However, the PUR and IMPUR macros also set global flags that are used by other macros. These flags are set regardless of whether CAL includes the statements in the assembly. Therefore, the conditionals IFZ and IFNZ should not be used with EQU flags to generate PURE or IMPURE statements. Macros should be written with global macro flags to alter the flow of controls.

Example:

```
FLAG      EQU   0
          IFNZ  FLAG
          PURE
          ENDC
```

In this example, CAL will not generate the PURE statement; however, CAL macro will set a global flag within the PURE macro, thus affecting other macros. An alternate approach follows:

Example:

```
         MACRO
         SETFLAG
         GBLB     %FLAG
%FLAG    SETB     0                     FLAG SETTING
         MEND
         MACRO
         ISPURE
         GBLB     %FLAG
         SETFLAG                        TO SET OR RESET THE
         AIF      (%FLAG)&PURE          GLOBAL FLAG
         MEXIT
&PURE    PURE
         MEND
```

A call to ISPURE with %FLAG set to 0 will not generate the PURE statement. A call to ISPURE with %FLAG set to 1 will generate the PURE statement.

CHAPTER 2
SUPERVISOR MACROS

2.1 INTRODUCTION

Supervisor macros are those macros that request services from the operating system. These macros enable the user to access the system calendar and clock, pause or end a task, build and search a mnemonic table, and perform various other functions.

The following sections detail the formats, parameter values, default values, required parameters, programming considerations, examples, and error messages for all supervisor macros.

Section 1.4, Parameter Field Value Mnemonics, explains the lowercase abbreviations that appear in the parameter fields of supervisor macros.

2.2 END OF TASK (EOT)

The EOT macro enables the task to terminate in an orderly manner. If the task has input/output (I/O) in progress, I/O is terminated; write operations terminate normally; read operations abort.

Format:

```
    [symbol]  EOT  [RC=]
```

Parameter Values:

RC	=	abs address exp (return code)
	=	addrx

Default Values:

RC	=	0

Required Parameters:

none

Programming Considerations:

If a nonresident task issues an EOT, all of its files and devices are closed, it is removed from memory, and all control information pertaining to the task is deleted. If the task is resident, its files are check-pointed, but not closed; it is not removed from memory.

The return code (RC) specifies the condition code that the operating system returns after the task is terminated.

A parameter block is not associated with the EOT macro.

Example:

```
    EOT
    EOT    RC=4
    EOT    RC=0(9)     R9 contains the return code
```

2.3 FETCH DATE (FETDATE)

The FETDATE macro returns the current date from the operating system. The format of the returned date is MMDDYY or DDMMYY, depending on the selection at system generation time.

Format:

```
    [symbol]  FETDATE  dest[,PCB=][,FORM=]
```

Parameter Values:

```
    dest        -  addrx (destination)
                -  (reg)

    PCB         =  addrx
                =  (reg)

    FORM        =  L
```

Default Values:

    none

Required Parameters:

    dest

Programming Considerations:

The parameter, dest, gives the starting address of an 8-byte buffer to receive the fetched date. This buffer must be in a writable segment.

Example:

```
             FETDATE   PLACE
             FETDATE   0(9)
             FETDATE   (8)
               .
               .
               .
             IMPUR
    DATE     FETDATE   ABC,FORM=L
    PLACE    DS        8
             PURE
             FETDATE PCB=DATE
```

Error Messages:

```
    MNOTE NO ADDRESS SPECIFIED - NO EXPANSION
    Return code = 4
```

2.4 FETCH POINTER (FETPTR)

The FETPTR macro fetches a pointer to the task's user-dedicated locations (UDLs). This pointer, or starting address, is returned in a specific general register. Also, FETPTR copies the address of UTOP, CTOP, and UBOT from the task's task control block (TCB) into its UDL. The register where the pointer is returned is reg.

Format:

```
    [symbol]  FETPTR  reg[,PCB=][,FORM=]
```

Parameter Values:

```
    reg       -  reg (register pointer)

    PCB       =  addrx
              =  (reg)

    FORM      =  L
```

Default Values:

none

Structure Generated:

UDLS

Required Parameters:

reg

Example:

```
    FETPTR  6       (pointer returned in R6)
```

Error Messages:

```
MNOTE NO REGISTER SPECIFIED - NO EXPANSION
Return code = 4
```

2.5 FETCH TIME (FETTIME)

The FETTIME macro fetches the current time of day from the operating system and returns it in a 4- or 8-byte buffer that the destination address points to. This buffer must be in a writable segment.

Format:

```
    [symbol]  FETTIME  dest[,OPT=][,PCB=][,FORM=]
```

Parameter Values:

dest	-	addrx (destination)
	-	(reg)
OPT	=	A (ASCII)
	=	B (binary)
PCB	=	addrx
	=	(reg)
FORM	=	L

Default Values:

OPT	=	A (ASCII)

Required Parameters:

dest

Programming Considerations:

OPT specifies ASCII or binary format for the receiving buffer. If OPT=A, the default, the buffer must be 8 bytes long and aligned on any byte boundary. The time stored within this buffer is HH:MM:SS.

If OPT=B (binary), the buffer is 4 bytes long aligned on a fullword boundary. The time stored has a binary value indicating seconds from midnight.

The operating system maintains a 24-hour clock, calibrated in seconds from midnight. A zero value equals midnight. A decimal value of 86,399 is equivalent to 23:59:59.

Example:

```
          FETTIME  ABC         TIME RETURNED TO ABC
          FETTIME  (9)         R9 POINTS TO THE BUFFER
          .
          .
          .
          IMPUR
ABC       DS       8
```

2.6 GET STORAGE (GETSTORE)

The GETSTORE macro increases the user's storage by adjusting UTOP upward according to the number of bytes specified in SIZE. Once the UTOP address is adjusted, the starting address of the reserved temporary storage area, which is the original or previous UTOP, is stored in the register POINTER.

Format:

```
    [symbol] GETSTORE pointer,size[,PCB=][,FORM=]
```

Parameter Values:

```
    pointer    -  reg (pointer to storage)

    size       -  abs address exp (size of storage)
               -  ALL
               -  (reg)

    PCB        =  addrx
               =  (reg)

    FORM       =  L
```

Default Values:

    none

Structure Generated:

    GETSTORS

Required Parameters:

    pointer
    size

Programming Considerations:

If the SIZE is a negative value or greater than the task's current allocation size (task's current CTOP):

- UTOP's address is not adjusted.
- A zero address is returned in the user-specified register.
- The condition code is set to 4 (V bit set).

NOTE

The number of bytes should be specified in fullword increments because UTOP's address is rounded up to the nearest fullword boundary.

If SIZE specifies ALL, UTOP is adjusted to CTOP+2 and the number of bytes actually reserved is stored in the SIZE field of the parameter block. In this case, the parameter block must be in a writable segment. The reserved storage address is returned in the register REG.

Example:

```
GETSTORE 3,600
GETSTORE 3,(4)               R4 CONTAINS THE NUMBER
                             OF BYTES
GETSTORE 3,ALL               GETS ALL STORAGE
LDA      4,GS.SIZE(14)       NUMBER OF BYTES TO R4
```

NOTE

R14 is pointing to the parameter block and the operating system returns the number of bytes into the GS.SIZE field of the parameter block.

2.7 GENERATE A GETSTORE STRUCTURE (GETSTORS)

The GETSTORS macro creates the structure that the GETSTORE macro needs. This structure can only be generated once. Repeated GETSTORS macros have no effect.

Format:

```
    blank    GETSTORS    blank
```

Structure Generated:

```
    *  GETSTORE PARBLK
    *
    GETSTORS  STRUC
    GS.OPT    DS     2              OPTIONS
    GS.REG    DS     2              REGISTER
    GS.SIZE   DAS    1              SIZE
              ENDS
```

NOTE

GETSTORS is automatically generated in the expansion of the GETSTORE macro.

2.8 BUILD A MNEMONIC TABLE (MNEMTBL)

The MNEMTBL macro builds a mnemonic table suitable for use with the SCAN macro.

Format:

```
[symbol] MNEMTBL (required,optional),...[,EOT]
```

Programming Considerations:

Operands must be paired and enclosed in parentheses. For each pair, the first parameter is the required portion of the mnemonic and the second parameter is the optional portion. The required portion of the mnemonic is the minimum number of characters that must be supplied for the mnemonic to be recognized. The optional portion completes the full spelling of the mnemonic and can or cannot be specified for the mnemonic to be recognized. The mnemonic separating character is inserted between each pair. Coding EOT indicates the end of the table. If an optional portion does not exist, the comma and parentheses can be omitted.

Example:

```
TAB MNEMTBL (PA,USE),(AS,SIGN),(END),(ST,ART),RW,EOT
```

Where:

PA is required; USE is optional.
AS is required; SIGN is optional.
END is required; no optional portion.
ST is required; ART is optional.
RW is required; no optional portion.

NOTE

In the first specified parameter, PA, these character sequences are recognized as the mnemonic: PA, PAU, PAUS, and PAUSE.

To obtain the characters EOT as a required part, use: MNEMTBL (EOT),EOT.

2.9 MOVE ASCII (MVASCII)

The MVASCII macro moves ASCII characters from an input string to a specified address. The input string can include ending characters for controlling the number of moved characters.

Format:

```
[symbol] MVASCII dest,source,len[,EC=][,OPT=]
                                [,PCB=][,FORM=]
```

Parameter Values:

dest	-	reg (destination pointer)
source	-	reg (source pointer)
len	-	abs byte exp (length)
	-	(reg)
EC	=	addrx (ending character)
	=	(reg)
	=	'quoted string'
OPT	=	EC (use ending character)
PCB	=	addrx
	=	(reg)
FORM	=	L

Default Values:

EC	no ending characters
OPT	no ending characters

Required Parameters:

dest
source
len

Programming Considerations:

The required parameter, dest, identifies the register pointing to the start of the output string address. This output address must be in a writable segment.

The required parameter, source, identifies the register pointing to the start of the input string.

The required parameter, len, gives the length (number of characters) of the input string to be moved. Its value must be less than or equal to 127.

OPT=EC signifies that "ending characters" are included within the input string. These ending characters are located at the address pointed to by EC=. OPT=EC is only required when EC= has been specified in a remote FORM=L call, and PCB= and the parameter, len, have both been specified in the current call.

EC= specifies a string of "ending characters" (within quotes) or the address of the following block: DB n; DB C'ending characters' where n is the number of ending characters. If a single quote is one of the ending characters, it should be coded as two consecutive single quotes but counted as one.

If only the three required parameters (dest, source, and len) are included with MVASCII, the number of characters specified are moved. As each character is moved, the dest and source registers are incremented to point to the location of the next character to be moved. At termination, the registers are pointing one byte past the characters moved and the condition code is set to zero (cc=0).

If ending characters are used, as each input string character is moved, it is checked against the ending characters. When the input string character matches an ending character, it is not moved; the MVASCII terminates and the condition code is set to zero (cc=0).

If the number of characters that the len parameter specifies have been moved and an expected match is not found, MVASCII terminates and the condition code is set to four (cc=4).

Example:

```
               MVASCII  3,4,5
               MVASCII  3,4,5,EC=' ,/'
                   .
                   .
                   .
               MVASCII  PCB=MOVE1
               MVASCII  ,,3,PCB=MOVE1,OPT=EC
                   .
                   .
                   .
MOVE1          MVASCII  ALPHA,BETA,5,EC=' ,/,FORM=L'
```

NOTE

Since a length was specified, the OPT=EC is required.

Error Messages:

```
MNOTE EC MUST BE REG OR ADDR WITH THIS OPTION
Return code = 1
```

2.10 PACK NUMERIC DATA (PACK)

The PACK macro converts an ASCII hexadecimal or decimal number to its equivalent binary value. PACK includes an option for skipping leading blanks in the input string.

Format:

```
    [symbol]  PACK  num[,OPT=][,PCB=][,FORM=]
```

Parameter Values:

num	-	reg (pointer to number to be packed)
OPT	=	DB (decimal - skip leading blanks)
	=	D (decimal)
	=	HB (hexadecimal - skip leading blanks)
	=	H (hexadecimal)
PCB	=	addrx
	=	(reg)
FORM	=	L

Default Values:

OPT	=	DB (decimal - skip leading blanks)

Required Parameters:

num

Programming Considerations:

The required parameter, num, is the register number containing the input string address to be packed. At the termination of PACK, register 0 (R0) contains the result and the pointer register contains the address of the byte following the last digit converted.

The valid ASCII hexadecimal numbers are 0 through 9 and A through F. The valid ASCII decimal numbers are 0 through 9. Any character, other than those specified ASCII hexadecimal or ASCII decimal numbers, causes the conversion process to stop and the nonconverted byte's address to be stored in the register.

The condition code setting is:

- A condition code of 1 indicates no characters processed; R0 is set to 0.
- A condition code of 4 indicates the number processed was too large to fit in a register. R0 contains the least-significant portion of the number processed.

Example:

```
PACK  4
STA   4,ALPHA
```

Register 4 (R4) is repositioned to point to the end of the numeric string, while the converted number is placed in R0.

Error Messages:

```
MNOTE INVALID OPTION - DECIMAL SKIP BLANKS USED
Return code = 1
```

An invalid option was specified; PACK proceeded assuming OP=DB.

2.11 PACK A FILE DESCRIPTOR (PACKFD)

PACKFD permits the user to process a file descriptor (fd) in standard operating system syntax: VOLN:FILENAME.EXT/ACCOUNT.

Format:

```
    [symbol]  PACKFD  source,dest[,OPT=][,FORMAT=]
                           [,PCB=][,FORM=]
```

Parameter Values:

source	-	reg (pointer to source string)
dest	-	addrx (destination address)
	-	(reg)
OPT	=	SYSVOLB (system volume - skip leading blanks)
	=	SYSVOL (system volume)
	=	NOVOLB (no volume - skip leading blanks)
	=	NOVOL (no volume)
	=	SPLVOLB (spool volume - skip leading blanks)
	=	SPLVOL (spool volume)
FORMAT	=	1
	=	2
PCB	=	addrx
	=	(reg)
FORM	=	L

Default Values:

OPT	=	SYSVOLB
FORMAT	=	1

Required Parameters:

source
dest

Programming Considerations:

If skip leading blanks is selected, the macro ignores all blanks from the current position of the pointer to the first nonblank. If skip leading blanks is not selected, it assumes that the fd to be converted starts at the current pointer position.

The reg parameter specifies one of the general registers that must point to the ASCII string of the unpacked fd.

The dest parameter points to a 16-byte receiving area aligned on a fullword boundary in a writable segment. The format is identical to the fd field of an FMPCB parameter block. The receiving area can be such a field.

Format 1 is normally used. If P, G, or S is specified in the unpacked fd, then a P, G, or S is returned in the packed fd, respectively. If this field is omitted in the unpacked fd, an S is then returned in the packed fd; however, P is returned for a task running under multi-terminal monitor (MTM). Any value other than P, G, or S in the account number field of an unpacked fd is treated as a syntax error.

Format 2 is used if the account number (numeric) is specified in the unpacked fd. When P, G, or S is specified or the account number field is omitted, the obtained result is the same as that result obtained in format 1. However, if a numeric value is found in the account number field of the unpacked fd, the G bit is set in the condition code and the numeric value is returned into the packed fd.

The extended fd and account number field are only meaningful in an MTM environment.

The pointer contained in register, reg, is returned pointing to the first byte that is not part of the fd.

The condition code is set on return as:

- A condition code of 0 indicates normal return.
- A condition code of 1 indicates no volume name present in input.
- A condition code of 2 indicates an account number rather than a P, G, or S appeared in account number field.
- A condition code of 4 indicates a syntax error.
- A condition code of 8 indicates no extension present in input.
- A condition code of 9 indicates no extension or volume present in input.

If a syntax error occurs, the scan of the unpacked fd terminates at the byte that caused the error. The contents of the area receiving the packed fd are filled with indeterminate code. Check the condition code to ensure that a syntax error has occurred. If the volume name, filename, or extension is fewer than 4, 8, or 3, respectively, the field is left-justified and the unused characters are set to blanks. (The operating system always sets the reserved character following the extension field to blank.)

If no volume name is provided and a "default volume" option is specified, the current default system volume name is moved into the volume name field of the packed fd. If this option is not specified, the contents of the volume name in the receiving field are left unchanged.

NOTE

> If the fd parameter is specified for file management macros (ALLOCATE, ASSIGN, RENAME, etc), a PACKFD macro is automatically issued with the option SYSVOLB and FORMAT=1.

Error Messages:

```
MNOTE INVALID OPTION - SYSVOLB USED
MNOTE INVALID FORMAT - FORMAT 1 USED
Return code = 1

MNOTE MISSING PARAMETER - NO EXPANSION
Return code = 4
```

2.12 PAUSE A TASK (PAUS)

The PAUS macro places the task in the console wait state. A message is issued to the system console. If the operator enters a CONTINUE command at the system console, the task is restarted at the next instruction.

Format:

```
    [symbol]  PAUS  [PCB=][,FORM=]
```

Parameter Values:

```
    PCB         =  addrx
                =  (reg)

    FORM        =  L
```

Default Values:

    none

Required Parameters:

    none

Programming Considerations:

Incomplete I/O requests continue to complete even when the task is in the paused state.

2.13 EXTRACT INFORMATION FROM SYSTEM TABLES (PEEK)

The PEEK macro obtains user-related information from the system pointer table (SPT) and task control block (TCB) and stores it in a corresponding location in the parameter block.

Format:

```
    [symbol]  PEEK  [OPT=][,PCB=][,FORM=]
```

Parameter Values:

OPT	=	0 returns the following information: number of logical units, maximum priority, operating system name, task name, current task status word (TSW), and task options.
		1 returns the following information: maximum blocking factor, operating system name, operating system update level, central processing unit (CPU) model numbers, system options, user account number, group account number, and system console name.
		2 returns the following information: operating system name, load volume, filename, extension, and file class.
PCB	=	addrx
	=	(reg)
FORM	=	L

Structure Generated:

PEEKS

Programming Considerations:

If no option number is specified for the OPT= parameter, the default is 0.

The parameter block must be in a writable segment. The names of the returned fields can be found in the PEEKS macro explained in Section 2.14. For OPT=0 use the TOPT. - equates. For OPT=1 use the SOPT. - equates.

2.14 GENERATE A PEEK STRUCTURE AND EQUATES (PEEKS)

The PEEKS macro generates the STRUC and EQUs required for the PEEK macro.

Format:

```
    blank  PEEKS  blank
```

Structure Generated:

```
    *
    * PEEK PARBLK
    *
    PEEKS     STRUC
    PK.OPT    DS  2                OPTIONS
    PK.NLU    DS  1                NUMBER OF LUs
    PK.MPRI   DS  1                MAXIMUM PRIORITY
    PK.OSID   DS  8                OS IDENTIFICATION
    PK.TSKID  DS  0                TASK IDENTIFIER (FORMAT 1)
    PK.UPLVL  DS  2                OS UPDATE LEVEL (FORMAT 2)
              DS  2                RESERVED (FORMAT 2)
    PK.SOPT   DS  4                SYSTEM OPTIONS (FORMAT 2)
    PK.CTSW   DS  4                CURRENT TASK STATUS WORD
    PK.TOPT   DS  2                TASK OPTIONS
    PK.STAT   DS  2                OPTIONS FROM TASK'S TCB
              ENDS
    *
```

Equates Generated:

```
    * SYSTEM OPTION EQUATES
    *
    SOPT.FPB  EQU  0            SYSTEM OPTION - SINGLE PRECISION
    SOPT.FPM  EQU  Y'80000000'                  FLOATING POINT
    SOPT.USB  EQU  1            SYSTEM OPTION-MMDDYY IF OFF
    SOPT.USM  EQU  Y'40000000'                DDMMYY IF ON
    SOPT.DIB  EQU  2            SYS. OPT.-DISPLAY TIME ON PANEL
    SOPT.DIM  EQU  Y'20000000'  DISPLAY IF ON
    SOPT.DFB  EQU  3            SYSTEM OPTION-DOUBLE PRECISION
    SOPT.DFM  EQU  Y'10000000'                FLOATING POINT
    SOPT.WCB  EQU  4
    SOPT.WCM  EQU  Y'08000000'  WCS SUPPORT
    SOPT.ALB  EQU  5
    SOPT.ALM  EQU  Y'04000000'  ALIGNMENT ERROR CHECK
```

```
SOPT.DAB   EQU  6
SOPT.DAM   EQU  Y'02000000'  SYSTEM OPTION-DIRECT ACCESS
SOPT.ITB   EQU  7
SOPT.ITM   EQU  Y'01000000'  SYSTEM OPTION-ITAM
SOPT.SPB   EQU  8
SOPT.SPM   EQU  Y'00800000'  SYSTEM OPTION-SPOOL SUPPORT
SOPT.RLB   EQU  9
SOPT.RLM   EQU  Y'00400000'  SYSTEM OPTION-ROLL SUPPORT
SOPT.TMB   EQU  10
SOPT.TMM   EQU  Y'00200000'  SYSTEM OPTION-TEMPFILES
*
* TASK OPTION EQUATES
*
TOPT.ETB   EQU  0
TOPT.ETM   EQU  X'8000'      E-TASK
TOPT.ACB   EQU  1
TOPT.ACM   EQU  X'4000'      ARITHMETIC FAULT CONTINUE
TOPT.FPB   EQU  2
TOPT.FPM   EQU  X'2000'      USING SINGLE FLOATING POINT
TOPT.MRB   EQU  3
TOPT.MRM   EQU  X'1000'      MEMORY RESIDENT
TOPT.CTB   EQU  4
TOPT.CIM   EQU  X'0800'      PREVENT SVC 6 CONTROL CALL
TOPT.CMB   EQU  5
TOPT.CMM   EQU  X'0400'      PREVENT SVC 6 COMMUNICATION CALL
TOPT.S6B   EQU  6
TOPT.S6M   EQU  X'0200'      SVC 6 CONTINUE
TOPT.DFB   EQU  7
TOPT.DFM   EQU  X'0100'      USING DOUBLE FLOATING POINT
TOPT.RLB   EQU  8
TOPT.RLM   EQU  X'0080'      ALLOW ROLL-OUT
TOPT.OVB   EQU  9
TOPT.OVM   EQU  X'0040'      USE OVERLAY
TOPT.SYB   EQU  10
TOPT.SYM   EQU  X'0020'      IN SYSTEM GROUP
TOPT.CIB   EQU  11
TOPT.CIM   EQU  X'0010'      CONSOLE I/O INTERCEPT ENABLE
                             (MTM)
TOPT.FAB   EQU  12
TOPT.FAM   EQU  X'0008'      FILE ACCOUNT PRIVILEGES
TOPT.LEB   EQU  13
TOPT.LEM   EQU  X'0004'      PREVENT E-TASK LOAD
TOPT.UVB   EQU  14
TOPT.UVM   EQU  X'0002'      UNIVERSAL
TOPT.KCB   EQU  15
TOPT.KCM   EQU  X'0001'      DO KEY CHECKS ON ASSIGN, E-TASK
```

NOTE

PEEKS is automatically generated in the expression of the PEEK macro.

2.15 RELEASE STORAGE (RELSTORE)

The RELSTORE macro releases the temporary storage in the unused portion of the task's impure segment that a previous GETSTORE macro reserved. Releasing temporary storage does not decrease the task's allocated memory size; but, it releases the unused portion of the task's impure segment. This area is between the UTOP and CTOP of a user's task.

Format:

```
    [symbol]  RELSTORE  size[,PCB=][,FORM=]
```

Parameter Values:

```
    size       -  abs address exp
               -  (reg)

    PCB        =  addrx
               =  (reg)

    FORM       =  L
```

Default Values:

none

Required Parameters:

size

Programming Considerations:

The parameter block must be in a writable segment. The size field must contain the number of bytes to be released. If the number of bytes is not specified in fullword increments, UTOP's address is adjusted by rounding down to the nearest fullword boundary. The condition code is set as:

- A condition code of 0 indicates normal termination.
- A condition code of 4 indicates the size parameter is a negative value or greater than the task's current allocation size.

2.16 SCAN A MNEMONIC TABLE (SCAN)

The SCAN macro permits the user to decode command mnemonics as the operating system command processor does.

Format:

```
    [symbol]  SCAN  TABLE=,SOURCE=,INDEX=[,PCB=][,FORM=]
```

Parameter Values:

TABLE	=	addrx
	=	(reg)
SOURCE	=	reg
INDEX	=	reg
PCB	=	addrx
	=	(reg)
FORM	=	L

Default Values:

none

Required Parameters:

TABLE
SOURCE
INDEX

Programming Considerations:

The source register must be pointing to the string to be scanned. Leading blanks are accounted for in the source. Use the SKTNB macro to ensure that this register actually points to the beginning of the string. The SKTNB macro is discussed in Section 8.13.

The TABLE parameter is the address of a command table to be recognized. This table can be built with the MNEMTBL macro. The MNEMTBL macro is detailed in Section 2.8.

The result, returned in the INDEX register, is a number that is -1 if a match was not found; or, 0 to n-1, where n is the number of mnemonics in the table, if a match is found. This number represents the matched mnemonic's position in the table, starting with zero. Thus, if a match is found on the third item in the table, the result returned in the INDEX register is 2.

The source register is returned pointing to the first nonblank character that is usually a separator following the mnemonic in the string being scanned. If a match is not found, the source register is returned unchanged. The condition code is set as:

- A condition code of 0 indicates that a match is found.
- A condition code of 4 indicates that a match has not been found.

Example:

To write a command processor that recognizes commands:

```
    COPY, DELETE, ADD, STOP

             .
             .
             .
             READ   INBLK
             LDAI   3,BUF                POINT TO BUFFER
             SKTNB  3                    POINT TO NONBLANK
             SCAN   SOURCE=3,INDEX=4,TABLE=COMMAND
             BO     CMDERR               ERROR
             SLLS   4,LADC               COMPUTE INDEX
             LDA    4,JTAB(4)            GET BRANCH ADDRESS
             BR     4                    VECTOR TO ROUTINE
             .
             .
             .
    JTAB     DAC    COPY,DELETE,ADD,STOP
    INBLK    IOPCB  ADDR=BUF,LU=5,RECL=80
    COMMAND  MNEMTBL (C,OPY),(D,ELETE),(A,DD),(S,TOP),EOT
    BUF      DS     80
             .
             .
             .
    CMDERR   EQU    *

* Routine to handle command errors
```

The COPY command is recognized as:

- C
- CO
- COP
- COPY

Other commands can be abbreviated in a similar manner.

CIPY is detected as an error and branches to CMDERR.

2.17 SET STATUS (SETSTAT)

The SETSTAT macro modifies the arithmetic fault (AF) interrupt bit and the condition code settings in the program status word (PSW). When the arithmetic fault interrupt bit setting is modified, interrupts are enabled (E) or disabled (D). When the condition code setting is modified, the current 4-bit setting is replaced with a new 4-bit setting.

Format:

```
    [symbol]  SETSTAT  [AF=][,CC=][,PCB=][,FORM=]
```

Parameter Values:

AF	=	E (arithmetic fault - enabled)
	=	D (arithmetic fault - disabled)
CC	=	abs byte exp (condition code)
	=	(reg)
PCB	=	addrx
	=	(reg)
FORM	=	L

Default Values:

AF	=	no change
CC	=	0 (zero)

Required Parameters:

none

Programming Considerations:

If no parameters are coded, the condition code is set to zero and the AF bit is not changed.

If only CC= is coded, the condition code is set; but, the arithmetic fault bit is not changed.

If only AF= is coded, the arithmetic fault bit is changed and the condition code is set to zero.

If CC= and AF= are coded, the arithmetic fault bit and condition code are changed.

If AF=E, an interrupt occurs when any of these conditions result during an arithmetic operation:

- fixed point quotient overflow
- fixed point division by zero
- floating point overflow and underflow
- floating point division by zero

If AF=D, all interrupts caused by an arithmetic fault are ignored.

2.18 UNPACK BINARY NUMBER (UNPK)

The UNPK macro converts an unsigned binary number located in a register into an ASCII hexadecimal number or an ASCII decimal number. Leading zeros can be included or replaced with blanks. The binary number is moved to register 0 (R0) for conversion.

Format:

```
    [symbol]  UNPK  num,dest[,LEN=][,OPT=][,PCB=][,FORM=]
```

Parameter Values:

```
    num         -  reg (number to be converted)

    dest        -  addrx (destination)
                -  (reg)

    LEN         =  abs byte exp (length - maximum 63)
                =  (reg)

    OPT         =  D  (decimal - leading blanks)
                =  DZ (decimal - leading zeros)
                =  H  (hexadecimal - leading blanks)
                =  HZ (hexadecimal - leading zeros)

    PCB         =  addrx
                =  (reg)

    FORM        =  L
```

Default Values:

```
    LEN         =  10 for D (decimal)
                =  8 for H (hexadecimal)
```

Required Parameters:

```
    num
    dest
```

Programming Considerations:

The required parameter, dest, must be in a writable segment. If the number to be converted is too large for dest, the most-significant digits are lost. The result is stored right-justified in dest with the left-most-significant digits filled with ASCII zeros or ASCII blanks, depending on the option. The number can be in any register; however, it is moved to register 0 (R0) before conversion.

Error Messages:

```
MNOTE INVALID OPTION - DECIMAL LEADING BLANKS USED
Return code = 1
```

2.19 WRITE TO OPERATOR--LOG MESSAGE (WTO)

The WTO macro writes a message to the operator's console or system log device. It can output a message regardless of logical unit (lu) assignments. The message can be a quoted string; or, it can be stored in memory. WTO is treated as a proceed call; consequently, the message can be modified or destroyed immediately following the macro.

Format:

```
    [symbol]  WTO  [msg][,ADDR=][,LEN=][,OPT=][,PCB=][,FORM=]
```

Parameter Values:

msg	-	'quoted string' (must be used alone or with OPT)
ADDR	=	addrx
	=	(reg)
LEN	=	abs halfword exp (must be used with ADDR)
	=	(reg)
OPT	=	F (format mode)
	=	I (image mode)
PCB	=	addrx
	=	(reg)
FORM	=	L

Default Values:

OPT	=	F (format mode)

Required Parameters:

msg

or

ADDR and LEN

Programming Considerations:

When sent to the appropriate log device, the message is formatted or in image mode. When a formatted message is sent to a device, these operations occur:

- All trailing blanks in the buffer or at the end of the message are eliminated.
- A carriage return line feed is automatically appended to the message.
- The message terminates when the end of the buffer or the message is reached, or when a carriage return is found in the message.

When a message is sent to a device in image mode, it terminates when the end of the buffer or message is reached.

When using image mode, a message with multiple lines can be sent with a single WTO macro. However, each line should include a carriage return and line feed at the end.

Error Messages:

```
MNOTE INVALID OPT - FORMAT USED
Return code = 1

MNOTE MISSING LENGTH - NO EXPANSION
Return code = 4
```

CHAPTER 3
FILE MANAGEMENT MACROS

3.1 INTRODUCTION

File management macros are those macros that manipulate files. These macros can create or delete direct access files, rename files, assign files or devices to a task's logical unit, modify the access privileges of such assignments, close the assignments, and fetch attributes.

The following sections detail the parameters for all file management macros and the formats, parameter values default values, required parameters, programming considerations, examples, and error messages for all file management macros.

Section 1.4 explains the lowercase abbreviations that appear in the parameter fields of file management macros.

3.2 PARAMETERS FOR FILE MANAGEMENT MACROS

A parameter, coded in the FMPCB macro, sets a constant into the parameter block. A parameter, coded in any other macro, replaces the value in the parameter block; but, the RESTART parameter, which cannot be coded in the FMPCB macro, can be coded in any other macro. The default value is the next instruction.

Required parameters can be coded in the FMPCB macro or in individual macros. It is more efficient to code those macros that do not change as constants in the FMPCB macro. Coding these in the individual macros generates code that stores the values in the parameter block. Refer to Section 3.17 for more details on the FMPCB macro.

- File Management Parameter Control Block (FMPCB)

  - FMPCB is specified as a file management parameter control block address or it is omitted. If it is omitted, it is constructed and filled in with remaining parameters. The FMPCB address is placed in R14.

Example:

```
CLOSE      LU=2
DELETE     PARBLK
ALLOCATE   0(2)
```

- File Descriptor (FD)

  - The FD is specified as the address of an unpacked file descriptor in standard form. A PACKFD macro packs the FD into the parameter block. PACKFD uses the default system volume with skip leading blanks. If a register is coded, it is repositioned to the byte after the last valid character of the FD.

    If a quoted string is coded in the FMPCB macro, it must be a packed file descriptor. The volume name must be specified; but, trailing blanks can be omitted. To allocate a temporary file, use FD='&'.

    Example:

```
ASSIGN     FD='CON:'
ALLOCATE   FD=(3)
ALLOCATE   FD='FILE1'
FMPCB      FD='CON'
FMPCB      FD='VOL FILE1   CALP'
ALLOCATE   FD='&',LU=2
```

- Logical Unit (LU)

  - LU represents the logical unit to which the file or device is attached.

    Example:

```
ASSIGN     LU=3
ALAS       LU=(8)
```

- Record Length (RECL)

  - RECL is the logical record length of an index file to be allocated.

Example:

```
ALLOCATE   RECL=80
ALLOCATE   RECL=(4)
```

- Number of 256-Byte Buffers for Blocking (BLKSIZE)

  - When allocating an index file, BLKSIZE is the number of 256-byte buffers allocated in the operating system. Logical records are packed into physical blocks of 256 bytes and then written to the file. BLKSIZE defaults to one for an index file in the FMPCB macro.

  Example:

```
ALLOCATE   BLKSIZE=4
     ALLOCATE   BLKSIZE=(7)
```

- Number of Index Blocks for an Index File (NDXSIZE)

  - When allocating an index file, NDXSIZE is the number of index blocks initially allocated. As the file grows, additional index blocks are automatically added. In the FMPCB macro, NDXSIZE defaults to one.

  Example:

```
ALLOCATE   NDXSIZE=4
```

- Number of Sectors for a Contiguous File (SIZE)

  - When allocating a contiguous file, SIZE is the number of sectors allocated. SIZE must be specified when allocating a contiguous file.

  Example:

```
ALLOCATE   FT=CO,SIZE=20
ALLOCATE   FT=CO,SIZE=(7)
```

- Access Privileges (AP)

  - When assigning a file, AP specifies access privileges associated with the file or device. When coding a register containing access privileges, the value must be in the low-order byte and the remainder of the register must be zero. Refer to this table:

CODE	VALUE	MEANING
SRO	X'00'	Sharable read-only
ERO	X'20'	Exclusive read-only
SWO	X'40'	Sharable write-only
EWO	X'60'	Exclusive write-only
SRW	X'80'	Sharable read-write
SREW	X'A0'	Sharable read, exclusive write
ERSW	X'C0'	Exclusive read, sharable write
ERW	X'E0'	Exclusive read-write

NOTE

An invalid code results in SRO being used.

- File Type (FT)

  - When allocating a file, FT specifies the file type to be allocated. When FT is coded in a register, the value must be the low-order byte and the remainder of the register must be zero. Refer to this table:

CODE	VALUE	MEANING
CO	X'00'	Contiguous
IN	X'02'	Index
ITAM	X'07'	ITAM buffered terminal manager

NOTE

An invalid code results in IN being used.

- ITAM Access Method (AM)

  - When allocating an integrated telecommunications access method (ITAM) terminal manager, AM specifies the line type. If AM is coded in a register, the value must be in the low-order byte and the remainder of the register must be zero. Refer to this table:

CODE	VALUE	MEANING
TL	X'00'	Terminal level
LL	X'18'	Line level

NOTE

An invalid code results in TL being used.

- Protection Keys (KEYS)

  - When assigning a file, the KEYS in the parameter block must match the keys in the directory.

- Table of Routine Addresses to Handle Errors (ERR)

  - The ERR parameter specifies the address of a table of routine addresses that handle errors returned by the file management macros. The FMERRTBL macro builds this table. The codes listed for each error message can be used in the FMERRTBL macro to provide branch addresses for each error.

- Pause on Error (PAUS)

  - On any error not specified in the FMERRTBL table, which is pointed to by the ERR parameter, the task will or will not pause after writing a message to the log device, depending on the PAUS flag.

- Location to Restart After Error (RESTART)

  - On any error not specified in the FMERRTBL table, which is pointed to by the ERR parameter, the task restarts after writing a message to the log device. If the task pauses, it continues at this address. The default is the next instruction.

3.3 ALLOCATE AND ASSIGN A FILE OR DEVICE (ALAS)

The ALAS macro is a combination of the ALLOCATE and ASSIGN macros. It allocates and assigns a file or device.

Format:

```
[symbol]  ALAS  [fmpcb][,FD=][,LU=][,AP=][,RECL=][,FT=]
                [,BLKSIZE=][,NDXSIZE=][,AM=][,KEYS=]
                [,ERR=][,RESTART=][,PAUS=][,SIZE=]
```

Parameter Values:

```
fmpcb      -  addrx
           -  (reg)

FD         =  addrx file descriptor
           =  (reg)
           =  'quoted string' (unpacked FD)

LU         =  absolute byte expression
           =  (reg)

AP         =  SRO    000xxxxx    (access privilege)
           =  ERO    001xxxxx    (access privilege)
           =  SWO    010xxxxx    (access privilege)
           =  EWO    011xxxxx    (access privilege)
           =  SRW    100xxxxx    (access privilege)
           =  SREW   101xxxxx    (access privilege)
           =  ERSW   110xxxxx    (access privilege)
           =  ERW    111xxxxx    (access privilege)
           =  (reg)

RECL       =  absolute halfword expression
           =  (reg)

FT         =  CO     xxxxx000     (contiguous file)
           =  IN     xxxxx010     (index file)
           =  ITAM   xxxxx111
           =  (reg)

BLKSIZE    =  absolute halfword expression
           =  (reg)

NDXSIZE    =  absolute halfword expression
           =  (reg)
```

AM	=	TL	xxx00xxx	(terminal level)
	=	LL	xxx11xxx	(line level)
	=	(reg)		
KEYS	=	absolute halfword expression		
	=	(reg)		
ERR	=	addrx		
	=	(reg)		
RESTART	=	addrx		
	=	(reg)		
PAUS	=	N		
SIZE	=	absolute fullword expression		
	=	(reg)		

Default Values:

FD	=	0
LU	=	0
AP	=	0
RECL	=	0
FT	=	IN
BLKSIZE	=	1 for indexed files; or else 0
NDXSIZE	=	1 for indexed files; or else 0
AM	=	TL
KEYS	=	0
ERR	=	PAUS flag if NO DEFAULT in previous FMERRTBL
	=	DEFAULT FMERRTBL of previous FMERRTBL
RESTART	=	next instruction
PAUS	=	pause if error

Required Parameters:

```
    FD
    LU
    AP
    RECL
    FT
    AM for ITAM
    KEYS
    SIZE for contiguous files
    NDXSIZE and BLKSIZE for index files
```

Programming Considerations:

Any required parameter, not specified in the ALAS macro, must be specified in the FMPCB macro. Section 3.17 details the FMPCB macro. Any specified parameter replaces the field in the parameter block.

Since the combined functions of the ALLOCATE and ASSIGN macros are performed, an error terminates without the macro completing the function. The error status cannot properly reflect the true error. For example, if ALAS fails on the ALLOCATE function, the ASSIGN function is not performed. The ALAS macro should only be used if the user is certain of no errors.

Error Messages:

```
    MNOTE INVALID ACCESS PRIV - SRO USED
    MNOTE INVALID FILE TYPE - INDEX USED
    MNOTE INVALID ACCESS METHOD - TL USED
    Return Code = 1
```

Also refer to the ALLOCATE or ASSIGN macros explained in Sections 3.4 and 3.5, respectively.

3.4 ALLOCATE A FILE (ALLOCATE)

The ALLOCATE macro makes a directory entry and reserves space on a direct access device specified in the file type (FT) parameter. (Section 3.2 explains the FT parameter.) When allocating through the Integrated Telecommunications Access Method (ITAM) buffered terminal manager, ALLOCATE reserves a memory area for a line control block.

Format:

```
[symbol]  ALLOCATE  [fmpcb][,FD=][,FT=][,RECL=][,SIZE=]
                    [,BLKSIZE=][,NDXSIZE=][,KEYS=]
                    [,ERR=][,RESTART=][,PAUS=][,LU=]
```

Parameter Values:

```
fmpcb      -  addrx
           -  (reg)

FD         =  addrx file descriptor
           =  (reg)
           =  'quoted string' (unpacked FD)

FT         =  CO     xxxxx000    (contiguous file)
           =  IN     xxxxx010    (index file)
           =  ITAM   xxxxx111
           =  (reg)

RECL       =  absolute halfword expression
           =  (reg)

SIZE       =  absolute fullword expression
           =  (reg)

BLKSIZE    =  absolute halfword expression
           =  (reg)

NDXSIZE    =  absolute halfword expression
           =  (reg)

KEYS       =  absolute halfword expression
           =  (reg)

ERR        =  addrx
           =  (reg)
```

RESTART	=	addrx
	=	(reg)
PAUS	=	N
LU	=	absolute byte expression
	=	(reg)

Default Values:

FD	=	0
FT	=	IN
RECL	=	0
SIZE	=	0
BLKSIZE	=	1 for indexed files; or else 0
NDXSIZE	=	1 for indexed files; or else 0
KEYS	=	0
ERR	=	PAUS flag if NO DEFAULT in previous FMERRTBL
	=	DEFAULT FMERRTBL of previous FMERRTBL
RESTART	=	next instruction
PAUS	=	pause if error

Required Parameters:

KEYS
RECL
FD
FT
SIZE for contiguous files
NDXSIZE and BLKSIZE for index files

NOTE

The FD parameter must specify an unpacked file descriptor because a PACKFD macro is generated.

Programming Considerations:

Any required parameter, not specified in the ALLOCATE macro, must be specified in the FMPCB macro. (Section 3.17 details the FMPCB macro.) Any specified parameter replaces the field in the parameter block.

When allocating contiguous files, a directory entry and a sector or number of sectors are reserved. When a contiguous file is allocated and the sectors are reserved, the filename, sector's starting address, read/write keys, and file type are entered into the directory. A contiguous file is not buffered.

When allocating a temporary file, the LU field must be filled in for the assign function. Temporary files are assigned when they are allocated by the ALLOCATE macro.

When allocating indexed files, only a directory entry is reserved. When an index file is allocated, the filename, number of logical records, read/write keys, and the file type are entered into the directory. Two data buffers and one index buffer are allocated in system space. Each data buffer equals the file's index block size.

When allocating a line control block, system space in memory is reserved. When a line control block is allocated and system space is reserved, the buffered terminal's filename, logical record length, and read/write keys are entered into the LCB. Two data buffers are allocated in system space. Each data buffer must equal the device's physical block size.

Error Messages:

CODE	MESSAGE
IF	VOL:FD - ILLEGAL FUNCTION - CANNOT ALLOCATE
VOL	VOL:FD - VOLUME NOT MOUNTED - CANNOT ALLOCATE
NAME	VOL:FD - EXISTS - CANNOT ALLOCATE
SIZE	VOL:FD - NO ROOM ON DISC - CANNOT ALLOCATE
PRIV	VOL:FD - PRIVILEGE ERROR - CANNOT ALLOCATE
TYPE	VOL:FD - DEVICE NOT DIRECT ACCESS - CANNOT ALLOCATE
FD	VOL:FD - INVALID FILENAME - CANNOT ALLOCATE
SYS	VOL:FD - ACCOUNT VIOLATION - CANNOT ALLOCATE
	XX - UNEXPECTED STATUS

```
MNOTE INVALID ACCESS PRIV - SRO USED
MNOTE INVALID FILE TYPE   - INDEX USED
MNOTE INVALID ACCESS METHOD - TL USED
Return code = 1
```

3.5 ASSIGN A FILE OR DEVICE (ASSIGN)

The ASSIGN macro uses an LU to establish a logical connection between the task and file or device; or, it uses a logical connection between an Integrated Telecommunications Access Method (ITAM) line and buffered terminal.

Format:

```
    [symbol]  ASSIGN  [fmpcb][,LU=][,FD=][,KEYS=][,AP=][,AM=]
                      [,ERR=][,RESTART=][,PAUS=]
```

Parameter Values:

```
    fmpcb      -  addrx
               -  (reg)

    LU         =  absolute byte expression
               =  (reg)

    FD         =  addrx
               =  (reg)
               =  'quoted string' (unpacked FD)

    KEYS       =  absolute halfword expression
               =  (reg)

    AP         =  SRO    000xxxxx    (access privilege)
               =  ERO    001xxxxx    (access privilege)
               =  SWO    010xxxxx    (access privilege)
               =  EWO    011xxxxx    (access privilege)
               =  SRW    100xxxxx    (access privilege)
               =  SREW   101xxxxx    (access privilege)
               =  ERSW   110xxxxx    (access privilege)
               =  ERW    111xxxxx    (access privilege)
               =  (reg)

    AM         =  TL     xxx00xxx    (terminal level)
               =  LL     xxx11xxx    (line level)
               =  (reg)

    ERR        =  addrx
               =  (reg)

    RESTART    =  addrx
               =  (reg)

    PAUS       =  N
```

Default Values:

LU	=	0
FD	=	0
KEYS	=	0
AP	=	SRO
AM	=	TL
ERR	=	PAUS flag if NO DEFAULT in previous FMERRTBL
	=	DEFAULT FMERRTBL of previous FMERRTBL
RESTART	=	next instruction
PAUS	=	pause if error

Required Parameters:

LU
FD
KEYS
AP
AM for ITAM

NOTE

The FD parameter must specify an unpacked file descriptor because a PACKFD macro is generated.

Programming Considerations:

Any required parameter, not specified in the ASSIGN macro, must be specified in the FMPCB macro. (Section 3.17 details the FMPCB macro.) Any specified parameter replaces the field in the parameter block.

When assigning to disk devices, the read/write keys corresponding to specified access privileges are compared to the read/write keys in the file directory entry. If the appropriate keys match, the file is assigned according to specified access privileges.

When assigning to nondirect access devices, only the access privileges are examined. If the access privileges are SWO or EWO and the user issues an ASSIGN, the file is positioned at its logical end (append mode). If an ASSIGN is not issued, the file is positioned at the beginning.

Error Messages:

```
CODE  MESSAGE


IF    LU  XXX  VOL:FD - ILLEGAL FUNCTION - CANNOT ASSIGN
LU    LU  XXX  VOL:FD - ILLEGAL LU NUMBER - CANNOT ASSIGN
VOL   LU  XXX  VOL:FD - VOLUME NOT MOUNTED - CANNOT ASSIGN
NAME  LU  XXX  VOL:FD - DOES NOT EXIST - CANNOT ASSIGN
PROT  LU  XXX  VOL:FD - PROTECTED BY KEYS - CANNOT ASSIGN
PRIV  LU  XXX  VOL:FD - PRIVILEGE ERROR - CANNOT ASSIGN
BUF   LU  XXX  VOL:FD - BUFFER ERROR - NO ROOM IN OS -
                        CANNOT ASSIGN
ASGN  LU  XXX  VOL:FD - ASSIGNED - CANNOT ASSIGN
FD    LU  XXX  VOL:FD - INVALID FILENAME - CANNOT ASSIGN
TGD   LU  XXX  VOL:FD - TRAP GENERATING DEVICE - CANNOT
                        ASSIGN
SYS   LU  XXX  VOL:FD - ACCOUNT VIOLATION - CANNOT ASSIGN
                        XX - UNEXPECTED STATUS

MNOTE INVALID ACCESS PRIV - SRO USED
MNOTE INVALID FILE TYPE - INDEX USED
MNOTE INVALID ACCESS METHOD - TL USED
Return code = 1
```

3.6 CHANGE ACCESS PRIVILEGES (CHAP)

The CHAP macro changes the current access privileges of an assigned file or device to the access privileges that the AP parameter specifies. The new access privileges must be compatible with the existing ones; if they are not compatible, the file's existing access privilege keys remain unchanged.

Format:

```
[symbol]  CHAP  [fmpcb][,LU=][,AP=][,ERR=]
                [,RESTART=][,PAUS=]
```

Parameter Values:

fmpcb	-	addrx		
	-	(reg)		
LU	=	absolute byte expression		
	=	(reg)		
AP	=	SRO	000xxxxx	(access privilege)
	=	ERO	001xxxxx	(access privilege)
	=	SWO	010xxxxx	(access privilege)
	=	EWO	011xxxxx	(access privilege)
	=	SRW	100xxxxx	(access privilege)
	=	SREW	101xxxxx	(access privilege)
	=	ERSW	110xxxxx	(access privilege)
	=	ERW	111xxxxx	(access privilege)
	=	(reg)		
ERR	=	addrx		
	=	(reg)		
RESTART	=	addrx		
	=	(reg)		
PAUS	=	N		

Default Values:

LU	=	0
AP	=	SRO
ERR	=	PAUS flag if NO DEFAULT in previous FMERRTBL
	=	DEFAULT FMERRTBL of previous FMERRTBL
RESTART	=	next instruction
PAUS	=	pause if error

Required Parameters:

LU
AP

Programming Considerations:

Any required parameter, not specified in the CHAP macro, must be specified in the FMPCB macro. (Section 3.17 details the FMPCB macro.) Any specified parameter replaces the field in the parameter block. The valid access privilege key combinations are shown in Table 3-1.

TABLE 3-1 VALID ACCESS KEY COMBINATIONS

CHANGE FROM	CHANGE TO							
	SRO	ERO	SWO	EWO	SRW	SREW	ERSW	ERW
SRO	Y	Y	N	N	N	N	N	N
ERO	Y	Y	N	N	N	N	N	N
SWO	N	N	Y	Y	N	N	N	N
EWO	N	N	Y	Y	N	N	N	N
SRW	Y	Y	Y	Y	Y	Y	Y	Y
SREW	Y	Y	Y	Y	Y	Y	Y	Y
ERSW	Y	Y	Y	Y	Y	Y	Y	Y
ERW	Y	Y	Y	Y	Y	Y	Y	Y

LEGEND

Y = VALID REQUEST
N = INVALID REQUEST

Error Messages:

```
CODE  MESSAGE


LU    LU  XXX  VOL:FD - ILLEGAL LU NUMBER - CANNOT CHANGE
                        ACCESS PRIVILEGES
PRIV  LU  XXX  VOL:FD - PRIVILEGE ERROR - CANNOT CHANGE
                        ACCESS PRIVILEGES
ASGN  LU  XXX  VOL:FD - NOT ASSIGNED - CANNOT CHANGE
                        ACCESS PRIVILEGES
      XX - UNEXPECTED STATUS

MNOTE INVALID ACCESS PRIV - SRO USED
MNOTE INVALID FILE TYPE - INDEX USED
MNOTE INVALID ACCESS METHOD - TL USED
Return code = 1
```

3.7 CHECK THE ERROR STATUS OF AN FMPCB (CHECKFM)

The CHECKFM macro generates code that checks the status after a file management function has been performed.

Format:

```
    [symbol]  CHECKFM  [fmpcb][,ERR=][,RESTART=]
```

Parameter Values:

fmpcb	-	addrx
	-	(reg)
ERR	=	addrx
	=	(reg)
RESTART	=	addrx
	=	(reg)

Default Values:

ERR	=	PAUS flag if NO DEFAULT in previous FMERRTBL
	=	DEFAULT FMERRTBL of previous FMERRTBL
RESTART	=	next instruction

NOTE

CHECKFM automatically occurs in the expansion of all file management macros.

3.8 CHECKPOINT A LOGICAL UNIT (CKPOINT)

The CKPOINT macro copies file buffered data to the indexed file or terminal buffered data to the terminal and updates the directory entries. Issuing a CKPOINT macro to a contiguous, nondirect access device or unbuffered file has the same effect as a WAITIO macro. Section 4.16 explains the WAITIO macro.

Format:

```
    [symbol]  CKPOINT  [fmpcb][,LU=][,AM=][,ERR=][,RESTART=]
                       [,PAUS=]
```

Parameter Values:

fmpcb	-	addrx		
	-	(reg)		
LU	=	absolute byte expression		
	=	(reg)		
AM	=	TL	xxx00xxx	(terminal level)
	=	LL	xxx11xxx	(line level)
	=	(reg)		
ERR	=	addrx		
	=	(reg)		
RESTART	=	addrx		
	=	(reg)		
PAUS	=	N		

Default Values:

LU	=	0
AM	=	TL
RESTART	=	next instruction
ERR	=	PAUS flag if NO DEFAULT in previous FMERRTBL
	=	DEFAULT FMERRTBL of previous FMERRTBL
PAUS	=	pause if error

Required Parameters:

LU
AM for ITAM

Programming Considerations:

Any required parameter, not specified in the CKPOINT macro, must be specified in the FMPCB macro. (Section 3.17 details the FMPCB macro.) Any specified parameter replaces the field in the parameter block.

After issuing a CKPOINT macro, the file pointer is not repositioned to the beginning of the file as in a CLOSE macro. CKPOINT should be used after a large amount or a considerably important amount of data is saved to a buffered file because it preserves the data by copying it to the file. If a system failure occurs and data exists in the file buffers, all data up to the last CLOSE or CKPOINT can be recovered; any data appended after the last CLOSE or CKPOINT is lost.

Error Messages:

CODE	MESSAGE
LU	LU XXX VOL:FD - ILLEGAL LU NUMBER - CANNOT CHECKPOINT
ASGN	LU XXX VOL:FD - NOT ASSIGNED - CANNOT CHECKPOINT
	XX - UNEXPECTED STATUS

3.9 CLOSE A LOGICAL UNIT AND DELETE A FILE (CLDE)

The CLDE macro closes an LU and deletes a file. It performs the combined functions of the CLOSE and DELETE macros.

Format:

```
    [symbol]  CLDE   [fmpcb][,FD=][,LU=][,KEYS=][,ERR=]
                     [,RESTART=][,PAUS=]
```

Parameter Values:

fmpcb	-	addrx
	-	(reg)
FD	=	addrx file descriptor
	=	(reg)
	=	'quoted string' (unpacked FD)
LU	=	absolute byte expression
	=	(reg)
KEYS	=	absolute halfword expression
	=	(reg)
ERR	=	addrx
	=	(reg)
RESTART	=	addrx
	=	(reg)
PAUS	=	N

Default Values:

FD	=	0
LU	=	0
KEYS	=	0

ERR	=	PAUS flag if NO DEFAULT in previous FMERRTBL
	=	DEFAULT FMERRTBL of previous FMERRTBL
RESTART	=	next instruction
PAUS	=	pause if error

Required Parameters:

FD
LU
KEYS

Programming Considerations:

Any required parameter, not specified in the CLDE macro, must be specified in the FMPCB macro. (Section 3.17 details the FMPCB macro.) Any specified parameter replaces the field in the parameter block.

Since the combined functions of the CLOSE and DELETE macros are performed, an error terminates the macro without completing the function. The error status cannot properly reflect the true error. For example, if CLDE fails on the close function, the delete function is not performed. The CLDE macro should only be used if the user is certain no errors exist.

Error Messages:

Refer to the error messages given for the CLOSE or DELETE macros explained in Sections 3.10 and 3.11, respectively.

3.10 CLOSE A LOGICAL UNIT (CLOSE)

The CLOSE macro breaks the logical connection between the task and file or between the device, or ITAM line, and terminal by closing the currently assigned LU.

Format:

```
    [symbol]  CLOSE  [fmpcb][,LU=][,ERR=][,RESTART=][,PAUS=]
```

Parameter Values:

fmpcb	-	addrx
	-	(reg)
LU	=	absolute byte expression
	=	(reg)
ERR	=	addrx
	=	(reg)
RESTART	=	addrx
	=	(reg)
PAUS	=	N

Default Values:

LU	=	0
ERR	=	PAUS flag if NO DEFAULT in previous FMERRTBL
	=	DEFAULT FMERRTBL of previous FMERRTBL
RESTART	=	next instruction
PAUS	=	pause if error

Required Parameters:

LU

Programming Considerations:

Any required parameter, not specified in the CLOSE macro, must be specified in the FMPCB macro. (Section 3.17 details the FMPCB macro.) Any specified parameter replaces the field in the parameter block.

When the LU is closed, all data in file buffers or terminal buffers are copied to the user's file.

Error Messages:

CODE	MESSAGE
LU	LU XXX - ILLEGAL LU NUMBER - CANNOT CLOSE
ASGN	LU XXX - NOT ASSIGNED - CANNOT CLOSE
	XX - UNEXPECTED STATUS

3.11 DELETE A FILE (DELETE)

The DELETE macro removes the file directory entry and releases the reserved space of a currently unassigned file on a direct access device. When deleting through the ITAM buffered terminal manager, a currently unassigned Line Control Block (LCB) is removed from memory.

Format:

```
    [symbol]  DELETE  [fmpcb][,FD=][,KEYS=][,ERR=][,RESTART=]
                      [,PAUS=]
```

Parameter Values:

fmpcb	-	addrx
	-	(reg)
FD	=	addrx
	=	(reg)
	=	'quoted string' (unpacked FD)
KEYS	=	absolute halfword expression
	=	(reg)
ERR	=	addrx
	=	(reg)
RESTART	=	addrx
	=	(reg)
PAUS	=	N

Default Values:

FD	=	0
KEYS	=	0
ERR	=	PAUS flag if NO DEFAULT in previous FMERRTBL
	=	DEFAULT FMERRTBL of previous FMERRTBL
RESTART	=	next instruction
PAUS	=	pause if error

Required Parameters:

FD
KEYS

NOTE

The FD parameter must specify an unpacked file descriptor because a PACKFD macro is generated.

Programming Considerations:

Any required parameter, not specified in the DELETE macro, must be specified in the FMPCB macro. (Section 3.17 details the FMPCB macro.) Any specified parameter replaces the field in the parameter block.

If the contents of the parameter block's volume name, filename, extension, and read/write keys fields match the fields in the file directory entry, the file is deleted. If the logical terminal's name matches the name in the LCB, the LCB is deleted.

Error Messages:

CODE	MESSAGE
VOL	VOL:FD - VOLUME NOT MOUNTED - CANNOT DELETE
NAME	VOL:FD - DOES NOT EXIST - CANNOT DELETE
PROT	VOL:FD - PROTECTED BY KEYS - CANNOT DELETE
PRIV	VOL:FD - PRIVILEGE ERROR - CANNOT DELETE
ASGN	VOL:FD - ASSIGNED - CANNOT DELETE
TYPE	VOL:FD - DEVICE NOT DIRECT ACCESS - CANNOT DELETE
FD	VOL:FD - INVALID FILENAME - CANNOT DELETE
TGD	VOL:FD - TRAP GENERATING DEVICE - CANNOT DELETE
SYS	VOL:FD - ACCOUNT VIOLATION - CANNOT DELETE
	XX - UNEXPECTED STATUS

3.12 GENERATE A FILE DESCRIPTOR STRUCTURE (FDS)

The FDS macro generates the structure for a packed file descriptor.

Format:

```
    blank  FDS  blank
```

Structure Generated:

```
    *           FILE DESCRIPTOR
    *
    FDS         STRUC
    FD.VOL      DS      4               VOLUME
    FD.FNAME    DS      8               FILENAME
    FD.EXT      DS      3               EXTENSIONS
    FD.ACT      DS      1               PRIVATE,GROUP,SYSTEM
                ENDS
```

3.13 FETCH ATTRIBUTES OF A FILE OR DEVICE ASSIGNED TO AN LU (FETATR)

The FETATR macro sends to the parameter block the physical attributes of the file or device currently assigned to the specified LU. The parameter block must be in a writable segment.

Format:

```
    [symbol]  FETATR  [fmpcb][,LU=][,ERR=][,RESTART=][,PAUS=]
```

Parameter Values:

fmpcb	-	addrx
	-	(reg)
LU	=	absolute byte expression
	=	(reg)
ERR	=	addrx
	=	(reg)
RESTART	=	addrx
	=	(reg)
PAUS	=	N

Default Values:

LU	=	0
ERR	=	PAUS flag if NO DEFAULT in previous FMMERTBL
	=	DEFAULT FMERRTBL of previous FMERRTBL
RESTART	=	next instruction
PAUS	=	pause if error

Required Parameters:

LU

Programming Considerations:

Any required parameter, not specified in the ALLOCATE macro, must be specified in the FMPCB macro. (Section 3.17 details the FMPCB macro.) Any specified parameter replaces the field in the parameter block.

The FM.DC field receives a hexadecimal number indicating the file or device type. Refer to the OS/32 System Planning and Configuration Guide for a list of the devices and their codes.

The FM.DATB field receives a hexadecimal number indicating certain file or device attributes. See the FMPCBS macro, Section 3.18, for the code and equates.

The FM.RECL field receives the logical record length of the file or physical record length of the device assigned to the specified LU (e.g., 80-byte records for card readers and 120 or 132-byte records for line printers). If the device has variable length records, a zero value is returned to this field (e.g., magnetic tape). However, variable length record devices are normally used as a fixed record length device. For direct access devices, a record length, which is the file's logical record length established at allocation time.

The FM.VOL, FM.FNAME, FM.EXT, and FM.ACT each receive the volume name, filename, extension, and file class, respectively. For a nondirect access device, the device mnemonic is sent to the FM.VOL field and the remaining fields are filled with blanks.

The FM.SIZE field receives an unsigned hexadecimal number indicating the current size of a direct access file which is the number of logical records in an index file and the number of sectors in a contiguous file.

All other fields remain unchanged.

Error Messages:

CODE	MESSAGE
LU	LU XXX - ILLEGAL LU NUMBER - CANNOT FETCH ATTRIBUTES
ASGN	LU XXX - NOT ASSIGNED - CANNOT FETCH ATTRIBUTES
	XX - UNEXPECTED STATUS

3.14 GENERATE THE SUBROUTINE TO CHECK THE STATUS OF AN FMPCB (FMERR)

The FMERR macro generates the subroutine to check the error status after the completion of a file management function. Refer to Section 1.7, Error Handling and Recovery, for a description of these functions.

Format:

```
    blank  FMERR  blank
```

NOTE

The subroutine is only generated on the first call of this macro. Subsequent calls do not generate another copy of the subroutine. FMERR is called by all file management macros.

3.15 FETCH RETURN ADDRESS IN A USER ROUTINE FOR FILE MANAGEMENT ERRORS (FMERRET)

In a user-defined routine to handle file management errors, a return to the instruction following the macro call that caused the error is obtained by using the FMERRET macro. The optional register is the register where the address is returned. The user routine can save this address before issuing any other file management macros.

Format:

```
    [symbol]  FMERRET  [reg]
```

Parameter Values:

```
    reg        -  register expression
```

Default Values:

```
    reg        -  15
```

Example:

If a program wants to allocate a file that already exists, the FMERRTBL macro specifies a return to delete the file and retries the ALLOCATE. At the end of this routine, the user might want to continue after the original ALLOCATE:

```
        ALLOCATE  NEWFILE,ERR=DELERR,FD=(3),FT=IN,
                  RECL=80

*  R3 points to the unpacked file descriptor

        ASSIGN      NEWFILE,LU=4        ASSIGN IT
           .
           .
           .
DELERR  FMERRTBL    PAUS=N,NAME=DEL
```

```
*  if name error occurs, branch to DEL

DEL      EQU        *
         FMERRET      12          GET RETURN ADDRESS
         STA          12,RETSAV
         LDAR         12,14       HOLD ADDRESS OF NEWFILE
                                  WHICH IS IN R14
         DELETE       (12)
         ALLOCATE     (12)
         LDA          12,RETSAV
         BR           12          RETURN TO ASSIGN
           .
           .
           .
NEWFILE  FMPCB
RETSAV   DAS         1
```

3.16 GENERATE A TABLE OF ADDRESSES FOR FM ERROR HANDLING (FMERRTBL)

The FMERRTBL macro generates a table of branch addresses to user-written routines to handle errors returned by the file management macros. Refer to Section 1.7, Error Handling and Recovery, for a description of these functions.

Format:

```
    [symbol]  FMERRTBL  [default][,IF=][,LU=][,VOL=][,NAME=]
                        [,SIZE=][,PROT=][,PRIV=][,BUF=]
                        [,ASGN=][,TYPE=][,FD=][,TGD=][,SYS=]
                        [,IO=][,PAUS=]
```

Parameter Values:

default	-	DEFAULT	(use this FMERRTBL as the default for all FMPCBs)
IF	=	addr	(illegal function)
LU	=	addr	(logical unit)
VOL	=	addr	(volume)
NAME	=	addr	(name)
SIZE	=	addr	(size)
PROT	=	addr	(protection)
PRIV	=	addr	(privilege)
BUF	=	addr	(buffer)
ASGN	=	addr	(assignment)
TYPE	=	addr	(type)
FD	=	addr	(file descriptor)
TGD	=	addr	(trap-generating device)
SYS	=	addr	(system)

IO	=	addr	(input/output)
PAUS	=	N	(no pause)

Default Values:

IF	=	no entry in table
LU	=	no entry in table
VOL	=	no entry in table
NAME	=	no entry in table
SIZE	=	no entry in table
PROT	=	no entry in table
PRIV	=	no entry in table
BUF	=	no entry in table
ASGN	=	no entry in table
TYPE	=	no entry in table
FD	=	no entry in table
TGD	=	no entry in table
SYS	=	no entry in table
IO	=	no entry in table
PAUS	=	pause if error

NOTE

If the DEFAULT parameter is specified, the FMERRTBL macro is the default for all FMPCB macros.

3.17 GENERATE A FILE MANAGEMENT PARAMETER CONTROL BLOCK (FMPCB)

The FMPCB macro constructs the parameter block for file management macros. It can be constructed alone or as part of the expansion of other file management macros.

Format:

```
    symbol  FMPCB  [AP=][,AM=][,FT=][,LU=][,KEYS=][,RECL=]
                   [,FD=][,NDXSIZE=][,BLKSIZE=][,SIZE=]
                   [,ERR=][,PAUS=]
```

Parameter Values:

AP	=	SRO	000xxxxx	(access privilege)
	=	ERO	001xxxxx	(access privilege)
	=	SWO	010xxxxx	(access privilege)
	=	EWO	011xxxxx	(access privilege)
	=	SRW	100xxxxx	(access privilege)
	=	SREW	101xxxxx	(access privilege)
	=	ERSW	110xxxxx	(access privilege)
	=	ERW	111xxxxx	(access privilege)
AM	=	TL	xxx00xxx	(terminal level)
	=	LL	xxx11xxx	(line level)
FT	=	CO	xxxxx000	(contiguous file)
	=	IN	xxxxx010	(index file)
	=	ITAM	xxxxx111	
LU	=	absolute byte expression		
KEYS	=	absolute halfword expression		
RECL	=	absolute halfword expression		
FD	=	'quoted string"		(packed FD)
SIZE	=	absolute fullword expression		
NDXSIZE	=	absolute halfword expression		

BLKSIZE	=	absolute halfword expression
ERR	=	addr
PAUS	=	N

NOTE

The FD parameter must specify a packed file descriptor because a PACKFD macro is not generated.

Default Values:

AP	=	SRO
AM	=	TL
FT	=	IN
LU	=	0
KEYS	=	0
RECL	=	0
FD	=	0
NDXSIZE	=	1 for indexed files; or else 0
BLKSIZE	=	1 for indexed files; or else 0
SIZE	=	0
ERR	=	PAUS flag if NO DEFAULT in previous FMERRTBL
	=	DEFAULT FMERRTBL of previous FMERRTBL
PAUS	=	pause if error

3.18 GENERATE AN FMPCB STRUCTURE AND EQUATES (FMPCBS)

The FMPCBS macro generates the structure and equates for the FMPCB parameter block.

Format:

```
    blank   FMPCBS   blank
```

Structure Generated:

```
* FM        PARBLK
*
FMPCBS      STRUC
FM.FC       DS      0       FUNCTION CODE
FM.FUN      DS      1       FUNCTION CODE
FM.DC       DS      0       DEVICE CODE (FETATR)
FM.MOD      DS      1       MODIFIER
FM.STAT     DS      1       STATUS
FM.LU       DS      1       LOGICAL UNIT
FM.DATB     DS      0       DEVICE ATTRIBUTES (FETATR)
FM.KEYS     DS      0       KEYS
FM.WRKY     DS      1       WRITE KEY
FM.RDKY     DS      1       READ KEY
FM.RECL     DS      2       LOGICAL RECORD LENGTH
FM.FD       DS      0       FILE DESCRIPTOR
FM.VOL      DS      4       VOLUME
FM.FNAME    DS      8       FILENAME FIELD
FM.EXT      DS      3       EXTENSION
FM.ACT      DS      1       PRIVATE, GROUP, SYSTEM
FM.SIZE     DS      0       FILE SIZE
FM.NDXSI    DS      2       INDEX BLOCK SIZE
FM.BLKSI    DS      2       DATA BLOCK SIZE
FM.PAUS     DS      4       PAUSE FLAG
FM.ERR      DAS     1       ERROR ADDRESS
FM.RESTA    DAS     1       RESTART ADDRESS
            ENDS
```

Equates Generated:

```
*  DEVICE ATTRIBUTES EQUATES
*
DATB.INB  EQU   0          INTERACTIVE DEVICE
DATB.INM  EQU   X'8000'
DATB.RDB  EQU   1          SUPPORTS READ
DATB.RDM  EQU   X'4000'
DATB.WRB  EQU   2          SUPPORTS WRITE
DATB.WRM  EQU   X'2000'
DATB.BIB  EQU   3          SUPPORTS BINARY
DATB.BIM  EQU   X'1000'
DATB.WAB  EQU   4          SUPPORTS WAIT I/O
DATB.WAM  EQU   X'0800'
DATB.RNB  EQU   5          SUPPORTS RANDOM
DATB.RNM  EQU   X'0400'
DATB.UPB  EQU   6          SUPPORTS UNCONDITIONAL PROCEED
DATB.UPM  EQU   X'0200'
DATB.IMB  EQU   7          SUPPORTS IMAGE
DATB.IMM  EQU   X'0100'
DATB.HIB  EQU   8          SUPPORTS HALT I/O
DATB.HIM  EQU   X'0080'
DATB.RWB  EQU   9          SUPPORTS REWIND
DATB.RWM  EQU   X'0040'
DATB.BRB  EQU   10         SUPPORTS BACKSPACE RECORD
DATB.BRM  EQU   X'0020'
DATB.FRB  EQU   11         SUPPORTS FORWARD SPACE RECORD
DATB.FRM  EQU   X'0010'
DATB.WFB  EQU   12         SUPPORTS WRITE FILEMARK
DATB.WFM  EQU   X'0008'
DATB.FFB  EQU   13         SUPPORTS FORWARD SPACE FILEMARK
DATB.FFM  EQU   X'0004'
DATB.BFB  EQU   14         SUPPORTS BACKSPACE FILEMARK
DATB.BFM  EQU   X'0002'
*
* DEVICE CODE EQUATES
*
DC.CO     EQU   0          CONTIGUOUS FILE
DC.IN     EQU   2          INDEXED FILE
DC.NUL    EQU   255        NULL DEVICE
```

NOTE

FMPCBS is automatically generated in the expansion of any file management macro.

3.19 RETURN THE RELATIVE RECORD ADDRESS OF THE NEXT SEQUENTIAL RECORD (NOTE)

The NOTE macro returns the relative record address of the next sequential record.

Format:

```
[label]  NOTE  [iopcb][,LU=][,RECNUMB=]
               [,RESTART=][,PAUS=][,ERR]
```

Parameter Values:

iopcb	- addrx (reg)
LU	= absolute byte expression = (reg)
RECNUMB	= record number = (reg)
PAUS	= N
ERR	= addrx = (reg)

Default Values:

iopcb	=
LU	= 0
RECNUMB	=
RESTART	= next instruction
PAUS	= pause if error
ERR	=

Programming Considerations:

A 256-byte sector is defined as one record for a contiguous file, for which relative record is interpreted as relative sector. The first record of a file is record 0. A NOTE issued to a logical unit (lu) that has had no I/O since being assigned will return a relative record address of 0.

If a NOTE is issued and a contiguous file is positioned at EOM or an indexed file is positioned at EOF, an EOM or EOF status is returned.

A NOTE always returns the record or sector address immediately following the last record referenced, or 0 if the file has not been referenced or has just been rewound. If a NOTE is issued and the returned record address is subsequently used in a POINT request, the latter function will position the file to the exact position it occupied when the NOTE request was issued. See POINT macro, Section 3.20.

3.20 REPOSITION A FILE TO A SPECIFIED RELATIVE RECORD ADDRESS (POINT)

The POINT macro repositions a file to a specified relative record address that is specified in the SVC 1 record address field.

Format:

```
    [label]  POINT  [iopcb][,LU=][,RECNUMB=]
                    [,RESTART=][,PAUS=][,ERR=]
```

Parameter Values:

iopcb	- addrx (reg)
LU	= absolute byte expression = (reg)
RECNUMB	= record number = (reg)
PAUS	= N
ERR	= addrx (reg)

Default Values:

iopcb	=
LU	= 0
RECNUMB	=
RESTART	= next instruction
ERR	=

Programming Considerations:

A POINT request specifying the relative record address 0 is equivalent to a rewind. If a POINT request specifies a record position beyond EOF for an indexed file or beyond EOM for a contiguous file, and EOM status is returned. The relative record position is set to one position beyond the last record for indexed files and one position beyond the last sector for contiguous files.

If a NOTE request is issued and the returned record address is subsequently used in a point request, the latter function will position the file to the exact position it occupied when the note request was issued.

3.21 RENAME A FILE ASSIGNED TO A LOGICAL UNIT (RENAME)

The RENAME macro changes a currently assigned filename and extension to the filename and extension specified in the FD parameter. The filename must be on a direct access device and assigned with ERW access privileges.

Format:

```
    [symbol]  RENAME  [fmpcb][,LU=][,FD=][,ERR=][,RESTART=]
                      [,PAUS=]
```

Parameter Values:

fmpcb	-	addrx
	-	(reg)
FD	=	addrx
	=	(reg)
	=	'quoted string' (unpacked FD)
LU	=	absolute byte expression
	=	(reg)
ERR	=	addrx
	=	(reg)
RESTART	=	addrx
	=	(reg)
PAUS	=	N

Default Values:

LU	=	0
FD	=	0
ERR	=	PAUS flag if NO DEFAULT in previous FMERRTBL
	=	DEFAULT FMERRTBL of previous FMERRTBL
RESTART	=	next instruction
PAUS	=	pause if error

Required Parameters:

FD
LU

NOTE

The FD parameter must specify an unpacked file descriptor because a PACKFD macro is generated.

Programming Considerations:

Any required parameter, not specified in the RENAME macro, must be specified in the FMPCB macro. (Section 3.17 details the FMPCB macro.) Any specified parameter replaces the field in the parameter block.

The RENAME macro ignores the volume name field. The filename and extension replace the current filename and extension in the directory.

Error Messages:

CODE	MESSAGE
LU	LU XXX VOL:FD - ILLEGAL LU NUMBER - CANNOT RENAME
NAME	LU XXX VOL:FD - EXISTS - CANNOT RENAME
PROT	LU XXX VOL:FD - PROTECTED BY KEYS - CANNOT RENAME
PRIV	LU XXX VOL:FD - PRIVILEGE ERROR - CANNOT RENAME
ASGN	LU XXX VOL:FD - NOT ASSIGNED - CANNOT RENAME
	LU XXX VOL:FD - DEVICE NOT DIRECT ACCESS - CANNOT RENAME
FD	LU XXX VOL:FD - INVALID FILENAME - CANNOT RENAME
SYS	LU XXX VOL:FD - ACCOUNT VIOLATION - CANNOT RENAME
	XX - UNEXPECTED STATUS

3.22 REPROTECT A FILE ASSIGNED TO A LOGICAL UNIT BY CHANGING THE KEYS (REPROT)

The REPROT macro changes the read/write protection keys of a currently assigned file to the contents of the KEYS parameter. The file must be on a direct access device and assigned with ERW access privileges.

Format:

```
    [symbol]  REPROT  [fmpcb][,LU=][,KEYS=][,ERR=][,RESTART=]
                      [,PAUS=]
```

Parameter Values:

fmpcb	-	addrx
	-	(reg)
LU	=	absolute byte expression
	=	(reg)
KEYS	=	absolute halfword expression
	=	(reg)
ERR	=	addrx
	=	(reg)
RESTART	=	addrx
	=	(reg)
PAUS	=	N

Default Values:

LU	=	0
KEYS	=	0
ERR	=	PAUS flag if NO DEFAULT in previous FMERRTBL
	=	DEFAULT FMERRTBL of previous FMERRTBL
RESTART	=	next instruction
PAUS	=	pause if error

Required Parameters:

LU
KEYS

Programming Considerations:

Any required parameter not specified in the REPROT macro, must be specified in the FMPCB macro. (Section 3.17 details the FMPCB macro.) Any specified parameter replaces the field in the parameter block.

When issuing the REPROT macro, the specified read/write keys replace the current read/write keys of a specified file in the device's directory.

Error Messages:

CODE	MESSAGE
LU	LU XXX VOL:FD - ILLEGAL LU NUMBER - CANNOT REPROTECT
PRIV	LU XXX VOL:FD - PRIVILEGE ERROR - CANNOT REPROTECT
ASGN	LU XXX VOL:FD - NOT ASSIGNED - CANNOT REPROTECT
TYPE	LU XXX VOL:FD - DEVICE NOT DIRECT ACCESS - CANNOT REPROTECT
SYS	LU XXX VOL:FD - ACCOUNT VIOLATION - CANNOT REPROTECT
	XX - UNEXPECTED STATUS

CHAPTER 4
INPUT/OUTPUT MACROS

4.1 INTRODUCTION

Input/output (I/O) macros enable a task to sequentially or randomly read and write records while maintaining full control over waiting for I/O to complete or to proceed and manipulate filemarks.

The following sections detail the parameters associated with input/output macros. The formats, parameter values, default values, required parameters, programming considerations, examples, and error messages are also supplied for each I/O macro.

Section 1.4, Parameter Field Value Mnemonics, explains the lowercase abbreviations that appear in the parameter field.

4.2 PARAMETERS FOR I/O MACROS

A parameter, coded in the input/output parameter control block (IOPCB) macro, sets a constant into the parameter block; an omitted parameter sets a zero value for the field. A parameter, coded in any other macro, replaces the value in the parameter block. Two exceptions are record length (RECL) and location to restart after error (RESTART). RECL computes the ending address and it is not part of the parameter block passed to the operating system. RESTART defaults to retry the I/O SVC.

The required parameters can be coded in the IOPCB macro or in individual macros. It is more efficient to code those parameters that do not change as constants in the IOPCB macro. Coding these parameters in individual macros results in generating code to store values in the parameter block. Refer to Section 4.11 for a detailed explanation of the IOPCB macro.

The following paragraphs detail the parameters for input/output macros.

- Input/Output Parameter Control Block (IOPCB)

  - The input/output parameter control block is specified as an address of an IOPCB macro. If omitted, it is constructed and filled in with remaining parameters. The IOPCB address is placed in register 14 (R14).

Example:

```
                    READ   PARBLK
                    WRITE  PARBLK,ADDR=ALPHA
                    REWIND LU=2
                          .
                          .
                          .
          PARBLK    IOPCB  LU=2,RECL=80,ADDR=BETA
```

- Logical Unit (LU)

  - LU is the logical unit where the I/O operation occurs. This LU must be assigned prior to any I/O operation.

  Example:

```
    BFILE  LU=4
    WRITE  PARBLK,LU=2
```

- Address of Data to Be Transferred (ADDR)

  - ADDR is the I/O buffer address that sends or receives the data being transferred. It is used only for READ and WRITE macros. To specify the amount of data to be transferred, refer to the RECL or ENDADDR parameters.

  Example:

```
    READ   ADDR=ALPHA,RECL=80
    WRITE  ADDR=BETA,ENDADDR=BETA+79
```

- Record Length (RECL)

  - RECL is the actual number of bytes to be transferred in a READ or WRITE macro.

  Example:

```
    READ   ADDR=ALPHA,LU=3,RECL=132
    WRITE  ADDR=BETA,LU=5,RECL=(8)
```

- Address of the Last Byte to Be Transferred (ENDADDR)

  - The ENDADDR parameter is the actual address of the last byte to be transferred. If RECL is specified, this field is computed as ADDR+RECL-1 and is automatically set. This address must be greater than or equal to ADDR.

- Actual Number of Bytes Transferred (TRANS)

  - TRANS is the actual number of bytes that a READ or WRITE macro transfers. If an error occurs during the data transfer, this field is modified with indeterminate data. It can be addressed as IO.TRANS(14) or IOPCB+IO.TRANS.

- Options Used for READ and WRITE (OPT)

  If options other than the default options are needed, specify them in every READ or WRITE macro. The options are:

  - ASCII (A) or Binary (B)

    The default for the ASCII or binary option is ASCII. If image (I) is coded, this option is ignored.

  - Wait (W) or Proceed (P)

    The default for wait and proceed modes is wait. In wait mode, the task stops execution, initiates the data transfer, and waits until I/O completion. If the device is not busy in the proceed mode, the I/O is initiated and returned to the calling task. If the device is busy, the request is queued and control is or is not returned, depending on the options CP and UP, to the calling task. If the IOQ option is specified in the LTSW macro and I/O has been completed, an item is added to the task queue.

  - Sequential (S) or Random (R) Access

    In sequential and random access, the default is sequential access. With sequential access, the next logical record is accessed. With random access, the record in the RECNUMB field is accessed.

- Conditional Proceed (CP) or Unconditional Proceed (UP)

  With conditional or unconditional proceeds, the default is CP. If, after a proceed request, the device is busy and the total number of requests exceeds the maximum, CP puts the task in a wait state. Once the request has been queued or initiated, the task resumes execution. If UP is coded and the device is busy, the task resumes execution at the UPEXIT address. UPEXIT must be coded in this case.

- Format (F) or Image (I) Mode

  With format and image modes, the default is format (F) mode. In format mode, the data being transferred is formatted according to the ASCII (A) or binary (B) options. In image mode, the data is not formatted. The data does not contain any control characters (carriage returns, line feeds) that the user must supply for inclusion.

- Extended Options for ITAM (XOPT)

  The XOPT parameter specifies the extended options for integrated telecommunications access method (ITAM) requests.

- Random Record Number (RECNUMB)

  RECNUMB specifies the number of the next logical record to be accessed. It is only used for READ and WRITE macros. If RECNUMB is specified in READ or WRITE macros, OPT=R does not have to be specified. If it is specified in only the IOPCB macro, the OPT=R must be coded for random access.

- End of File (EOF)

  The EOF parameter specifies the address to go to if an EOF condition arises on an indexed file. It can be coded as part of a macro or incorporated as part of the IOERRTBL macro.

- End of Medium (EOM)

  The EOM parameter specifies the address to go to if an EOM condition arises on a contiguous file. It can be coded as part of a macro or it can be incorporated as part of the IOERRTBL macro.

- End of File or End of Medium (END)

  The END parameter can be used in place of the EOF and EOM parameters; it detects either condition. It is most useful when the program goes to the same address in both cases. It can be coded as part of a macro or incorporated as part of the IOERRTBL macro.

- Table of Routines to Handle Errors (ERR)

  The ERR parameter specifies the address of an address table of routines that handles errors returned by I/O macros. The IOERRTBL macro builds this table. The codes listed for each error message can be used in the IOERRTBL macro to provide branch addresses for each error.

- Pause on Error (PAUS)

  On any error not specified in the IOERRTBL table (pointed to by ERR), the task does or does not pause after writing a message to the log device.

- Location at which to Restart After Error (RESTART)

  On any error not specified in the IOERRTBL table (pointed to by ERR), the task restarts after writing a message to the log device. If the task pauses, it continues at this address. The default is to retry the I/O operation.

4.3 BACKWARD TO FILEMARK ON A FILE OR DEVICE (BFILE)

The BFILE macro backspaces the device or file assigned to the LU over one filemark. For an indexed file, this backspacing is equivalent to a rewind. For a contiguous file or magnetic tape, the effect is to position to the end of the previous file.

Format:

```
    [symbol]  BFILE  [iopcb][,LU=]
```

Parameter Values:

iopcb	-	addrx
	-	(reg)
LU	=	absolute byte expression
	=	(reg)

Default Values:

LU	=	0

Required Parameters:

LU

Programming Considerations:

If positioned at the beginning of a file, BFILE has no effect. BFILE to an indexed file has the same effect as a REWIND because indexed files do not recognize filemarks.

To reposition to the beginning of a file on magnetic tape or on a contiguous file after reading a filemark, the issuing of two BFILEs is required. The first BFILE positions the tape to the end of the file before the filemark. The second BFILE positions it to the beginning of the file if it is the first file on the tape. If the file is not the first on the tape, the second BFILE positions over the beginning filemark to the end of the previous file. To position to the beginning of the desired file, issue a FFILE to position past the filemark. Section 4.5 details the FFILE macro.

NOTE

BFILE is treated as a proceed call. Refer to Section 4.16 for a detailed explanation of the WAITIO macro.

4.4 BACKSPACE ONE RECORD (BREC)

The BREC macro backspaces an LU to the previous record. If the LU is at the beginning of the file, backspacing does not occur.

Format:

```
    [symbol]  BREC  [iopcb][,LU=]
```

Parameter Values:

iopcb	-	addrx
	-	(reg)
LU	=	absolute byte expression
	=	(reg)

Default Values:

LU	=	0

Required Parameters:

LU

NOTE

BREC is treated as a proceed call. Refer to Section 4.16 for a detailed description of the WAITIO macro.

4.5 FORWARD TO FILEMARK ON A FILE OR DEVICE (FFILE)

The FFILE macro forward spaces over one filemark on the device or file assigned to the LU. For an indexed file, the file is positioned at the end of the file. For a contiguous file or magnetic tape, the file is positioned after the filemark at the beginning of the next file.

Format:

```
    [symbol]  FFILE  [iopcb][,LU=]
```

Parameter Values:

iopcb	-	addrx
	-	(reg)
LU	=	absolute byte expression
	=	(reg)

Default Values:

LU	=	0

Required Parameters:

LU

Programming Considerations:

To position at the end of an indexed file in order to append, issue a FFILE. To position at the end of a contiguous file or magnetic tape, issue a FFILE followed by a BFILE to position back over the filemark. Section 4.3 explains the BFILE macro.

NOTE

FFILE is treated as a proceed call. Refer to the WAITIO macro discussed in Section 4.16.

4.6 FORWARD TO NEXT RECORD ON A FILE OR DEVICE (FREC)

The FREC macro forward spaces an LU to the next logical record. Spacing does not occur if the LU is positioned at the end of a file or device.

Format:

```
    [symbol]  FREC  [iopcb][,LU=]
```

Parameter Values:

iopcb	-	addrx
	-	(reg)
LU	=	absolute byte expression
	=	(reg)

Default Values:

LU	=	0

Required Parameters:

LU

NOTE

FREC is treated as a proceed call. Refer to the WAITIO macro discussed in Section 4.16.

4.7 HALT AN INPUT/OUTPUT PROCEED REQUEST (HALTIO)

The HALTIO macro cancels a previously issued proceed I/O request. (This action is useful on an interactive device.) If a HALTIO macro is not used, an outstanding request must be satisfied before any other I/O can be started on that LU.

Format:

```
    [symbol] HALTIO [iopcb][,LU=][,ERR=][,RESTART=][,PAUS=]
```

Parameter Values:

iopcb	-	addrx
	-	(reg)
LU	=	absolute byte expression
	=	(reg)
ERR	=	addrx
	=	(reg)
RESTART	=	addrx
	=	(reg)
PAUS	=	N

Default Values:

LU	=	0
ERR	=	PAUS flag if no default in previous IOERRTBL
RESTART	=	0
PAUS	=	pause if error

Required Parameters:

LU

Programming Considerations:

When a HALTIO macro is issued to an LU, any previous I/O proceed requests, whether in progress or queued, are cancelled. When I/O is terminated, the task that issued the I/O proceed request takes a trap (if enabled); the parameter block address that issued the I/O proceed request is placed on the task queue; and the I/O operation status (data transfer or command function) is returned to the status fields of both parameter blocks. The time of the actual termination is asynchronous to when the HALTIO macro is issued.

When an I/O request is issued to an LU and a previous I/O proceed request exists for the same LU, the second request and any subsequent requests to that LU cannot be serviced until the previous I/O request has been completed. When issuing a HALTIO macro, the first I/O request is cancelled, allowing I/O requests issued after the cancellation to be started on the device.

If the QIO statement was specified at task establishment time and at least one I/O request to a specified LU is on the user I/O queue, executing a HALTIO macro cancels any I/O to that specified LU already in progress and all requests to that specified LU on the user I/O queue.

These devices support the HALTIO macro:

- card reader
- Carousel
- CRT
- cassette
- Owl 1100 CRT
- Owl 1200 CRT
- paper tape reader/punch
- printer
- Teletype keyboard/printer

The system returns status in the HALT and PROCEED blocks as follows:

1. In the HALT block status IO.STAT(14):

   X'00' indicates that the requested I/O termination has been scheduled.

   X'81' indicates that an LU has not been assigned.

   X'82' indicates that I/O is not ongoing for this LU.

   The device number is placed in the IO.DN(14) field.

2. The PROCEED block occurs when I/O actually terminates. X'82' is returned in IO.STAT(14).

   When a proceed I/O call is requested, the status field (a halfword) is initialized to a positive value (1). The user can sense (poll) this status to determine I/O completion.

   Example:

```
BLK      IOPCB    ADDR=BUF,RECL=80,LU=1
BUF      DS       80
         .
         .
         .
         READ     BLK,OPT=P

*R14 contains the address of BLK
*IO.STAT has been preset to plus 1

         .
         .
         .
         B         TEST
IODONE   EQU       *
         .
         .
         .
```

4.8 GENERATE THE SUBROUTINE THAT CHECKS THE STATUS OF AN IOPCB (IOERR)

The IOERR macro generates the subroutine that checks the error status after completing an I/O function. Refer to Section 1.7, Error Handling and Recovery, for a description of these functions.

Format:

```
    blank      IOERR      blank
```

NOTE

The subroutine is only generated on the initial call of this macro. Subsequent calls do not generate another copy of the subroutine. IOERR is called by all nonproceed I/O macro calls.

4.9 FETCH RETURN ADDRESS IN A USER ROUTINE FOR INPUT/OUTPUT ERRORS (IOERRET)

The IOERRET macro, used in a user-defined routine to handle I/O errors, fetches the return address and enables the user to return to the instruction following the macro that caused the error. The optional register is the register where the address is returned. The user routine can save this address before issuing any other I/O macro.

Format:

```
    blank IOERRET blank
```

Parameter Values:

reg	-	register expression

Default Value:

reg	-	15

4.10 GENERATE A TABLE OF ADDRESSES FOR INPUT/OUTPUT ERROR USER HANDLING ROUTINES (IOERRTBL)

The IOERRTBL macro generates a table of branch addresses to user-written routines. This table handles errors that I/O macros return. Refer to Section 1.7, Error Handling and Recovery, for a description of these functions.

Format:

```
[symbol]  IOERRTBL  [default][,IF=][,LU=][,DU=][,EOM=]
                    [,EOF=][,UERR=][,RERR=][,UNERR=]
                    [,PAUS=][,END=]
```

Parameter Values:

default	-	DEFAULT (use this IOERRTBL as the default for all I/O macros)
IF	=	addr (illegal function)
LU	=	addr (illegal or unassigned LU)
DU	=	addr (device unavailable)
EOM	=	addr (end of medium)
EOF	=	addr (end of file)
UERR	=	addr (unrecoverable error)
RERR	=	addr (parity or recoverable error)
UNERR	=	addr (unknown error)
PAUS	=	N (no pause)
END	=	addr (end of file or medium)

Default Values:

IF	=	no entry in table
LU	=	no entry in table
DU	=	no entry in table
EOM	=	no entry in table
EOF	=	no entry in table
END	=	no entry in table
UERR	=	no entry in table
RERR	=	no entry in table
UNERR	=	no entry in table
PAUS	=	pause if error

4.11 GENERATE AN INPUT/OUTPUT PARAMETER CONTROL BLOCK (IOPCB)

The IOPCB macro constructs the parameter block for I/O macros. It can be constructed alone or as part of the expansion of other I/O macros.

Format:

```
[symbol]  IOPCB  [FUN=][,LU=][,STAT=][,DN=][,ADDR=]
                 [,ENDADDR=][,RECNUMB=][,TRANS=]
                 [,XOPT=][,RECL=][,RESTART=][,PAUS=]
                 [,ERR=]
```

Parameter Values:

FUN	=	absolute byte expression*
LU	=	absolute byte expression
STAT	=	absolute byte expression*
DN	=	absolute byte expression*
ADDR	=	relocatable address expression
ENDADDR	=	relocatable address expression
RECNUMB	=	absolute fullword expression
TRANS	=	absolute address expression*
XOPT	=	absolute fullword expression
RECL	=	absolute address expression
RESTART	=	relocatable address expression*
PAUS	=	N
ERR	=	addr

* These parameters are usually not needed because macros or the operating system set the fields.

Default Values:

FUN	=	0
LU	=	0
STAT	=	0
DN	=	0
ADDR	=	0
ENDADDR	=	ADDR+RECL-1 if both are specified
	=	0 otherwise
RECNUMB	=	0
TRANS	=	0
XOPT	=	0
RECL	=	0
RESTART	=	0
PAUS	=	pause if error
ERR	=	PAUS flag if no default in previous IOERRTBL
	=	default IOERRTBL of previous IOERRTBL

4.12 GENERATE AN IOPCB STRUCTURE (IOPCBS)

The IOPCBS macro generates the STRUCs and equates for the IOPCB parameter block.

Format:

```
    blank   IOPCBS   blank
```

Structure Generated:

```
*
IOPCBS    STRUC
IO.FUN    DS       0         FUNCTION CODE
IO.FC     DS       1         FUNCTION CODE
IO.LU     DS       1         LOGICAL UNIT
IO.DINDS  DS       0         DEVICE INDEPENDENT STATUS
IO.STAT   DS       1         STATUS
IO.DDEPS  DS       0         DEVICE DEPENDENT STATUS
IO.DN     DS       1         DEVICE NAME
IO.ADDR   DS       ADC       STARTING ADDR
IO.ENDAD  DS       ADC       ENDING ADDR
IO.RNDAD  DS       4         RANDOM ADDR
IO.TRANS  DS       ADC       TRANSFER LENGTH
IO.XOPT   DS       4         ITAM REQUESTS
IO.RECL   DS       ADC       RECORD LENGTH
IO.PAUS   DS       4         PAUSE FLAG
IO.ERR    DAS      1         ERROR TABLE POINTER
IO.RESTA  DS       ADC       RESTART ADDRESS
          ENDS
```

NOTE

IOPCBS is automatically generated in any I/O macro expansion.

4.13 READ A LOGICAL RECORD (READ)

The READ macro accesses the next logical record according to the specified options. If RECNUMB or OPT=R is coded, the next random record is accessed. Section 4.2 summarizes the parameters for I/O macros.

Format:

```
[symbol]  READ  [iopcb][,LU=][,ADDR=][,RECNUMB=]
                [,RECL=][,ENDADDR=][,EOF=][,EOM=]
                [,OPT=][,ERR=][,END=][,RESTART=]
                [,UPEXIT=][,PAUS=][,XOPT=]
```

Parameter Values:

```
iopcb     -  addrx
          -  (reg)

LU        =  absolute byte expression
          =  (reg)

ADDR      =  addrx
          =  (reg)

RECL      =  absolute address expression does not change
             IO.RECL
          =  (reg)

ENDADDR   =  addrx
          =  (reg)

EOF       =  addrx

EOM       =  addrx

OPT       =  B  (binary)                  specified in
          =  P  (proceed)                 any order
          =  I  (image)                   enclosed in
          =  UP (unconditional proceed)   parentheses
          =  R  (random)

RECNUMB   =  absolute fullword expression
          =  (reg)

ERR       =  addrx
```

END	=	addrx
RESTART	=	addrx
UPEXIT	=	addrx
XOPT	=	absolute fullword expression
	=	(reg)
PAUS	=	N

Default Values:

LU	=	0
ADDR	=	0
RECNUMB	=	0
RECL	=	0
ENDADDR	=	ADDR+RECL-1 if both are specified
	=	0 otherwise
XOPT	=	0
ERR	=	PAUS flag if no default in previous IOERRTBL
	=	default IOERRTBL of previous IOERRTBL
RESTART	=	retry the READ
PAUS	=	pause if error
END	=	addrx

Required Parameters:

ADDR
LU
RECL or ENDADDR

NOTE

On a proceed call, the status is set to positive one for polling. RECNUMB= or OPT=R, or both, cause a random READ.

Error Messages:

CODE	MESSAGE
IF	I/O ERROR C0XX LU X - ILLEGAL FUNCTION
DU	I/O ERROR A0XX LU X - DEVICE UNAVAILABLE
EOM	I/O ERROR 90XX LU X - END OF MEDIUM
EOF	I/O ERROR 88XX LU X - END OF FILE
UERR	I/O ERROR 84XX LU X - UNRECOVERABLE ERROR
RERR	I/O ERROR 82XX LU X - PARITY OR RECOVERABLE ERROR
LU	I/O ERROR 81XX LU X - ILLEGAL OR UNASSIGNED LU

4.14 REWIND A FILE OR DEVICE (REWIND)

The REWIND macro rewinds the file or device assigned to the LU. The file or device is repositioned to its beginning whether or not any filemarks are found.

Format:

```
    [symbol]  REWIND  [iopcb][,LU=]
```

Parameter Values:

```
    iopcb     -  addrx
              -  (reg)

    LU        =  absolute byte expression
              =  (reg)
```

Default Values:

```
    LU        =  0
```

Required Parameters:

```
    LU
```

NOTE

REWIND is treated as a proceed call. Refer to the WAITIO macro detailed in Section 4.16.

4.15 TEST FOR INPUT/OUTPUT COMPLETION (TESTIO)

The TESTIO macro tests for I/O completion to a specified LU. If a previous I/O proceed request or queued I/O proceed request does exist, the condition code is set to X'F'. However, if no outstanding I/O proceed request exists, the condition code is set to X'0'.

Format:

```
    [symbol]  TESTIO  [iopcb][,LU=]
```

Parameter Values:

iopcb	-	addrx
	-	(reg)
LU	=	absolute byte expression
	=	(reg)

Default Values:

LU	=	0

4.16 WAIT FOR INPUT/OUTPUT COMPLETION (WAITIO)

The WAITIO macro puts the task in a wait state until all previous I/O proceed requests to the specified LU, which are in progress or currently on the I/O queue, are serviced and all I/O has been completed. Task execution then resumes. If an I/O proceed request is not on the I/O queue, user-control is returned and task execution continues. When I/O is completed, the previous I/O proceed request status is returned to the status fields of the parameter block that issued the I/O proceed request.

Format:

```
    [symbol]  WAITIO  [iopcb][,IOPCB=][,EOF=][,EOM=][,END=]
                      [,RESTART=][,ERR=][,PAUS=]
```

Parameter Values:

```
    iopcb      -  addrx   (WAIT block)
               -  (reg)

    IOPCB      =  addrx   (PROCEED block)
               =  (reg)

    EOF        =  addrx
               =  (reg)

    EOM        =  addrx
               =  (reg)

    END        =  addrx
               =  (reg)

    RESTART    =  addrx
               =  (reg)

    ERR        =  addrx
               =  (reg)

    PAUS       =  N
```

Default Values:

RESTART	=	0
ERR	=	PAUS flag if no default in previous IOERRTBL
	=	default IOERRTBL of previous IOERRTBL
PAUS	=	pause if error

Required Parameters:

IOPCB	=	PROCEED block

Programming Considerations:

The LU number from the PROCEED block is placed in the WAIT block and the wait state is entered. The status is returned to the PROCEED block and that status is checked after the task resumes execution.

If RESTART is omitted, the restart address is the same as whatever address is in the PROCEED parameter block. The default is to retry the proceed I/O request.

Error Messages:

CODE	MESSAGE
IF	I/O ERROR C0XX LU X - ILLEGAL FUNCTION
DU	I/O ERROR A0XX LU X - DEVICE UNAVAILABLE
EOM	I/O ERROR 90XX LU X - END OF MEDIUM
EOF	I/O ERROR 88XX LU X - END OF FILE
UERR	I/O ERROR 84XX LU X - UNRECOVERABLE ERROR
RERR	I/O ERROR 82XX LU X - PARITY OR RECOVERABLE ERROR
LU	I/O ERROR 81XX LU X - ILLEGAL OR UNASSIGNED
UNERR	I/O ERROR XXXX LU X - UNKNOWN ERROR

4.17 WRITE FILEMARK (WFM)

The WFM macro writes a filemark to the file or device assigned to the specified LU. If it is an indexed file, no action occurs.

Format:

```
    [symbol]  WFM  [iopcb][,LU=]
```

Parameter Values:

```
    iopcb      -  addrx
               -  (reg)

    LU         =  absolute byte expression
               =  (reg)
```

Default Values:

```
    LU         =  0
```

Required Parameters:

LU

NOTE

WFM is treated as a proceed call. Refer to the WAITIO macro discussed in Section 4.16.

4.18 WRITE A LOGICAL RECORD (WRITE)

The WRITE macro writes the next logical record according to the specified options. If RECNUMB or OPT=R is coded, the next random record is written. Refer to Section 4.1, Summary of Parameters for Input/Output Macros, for a description of these functions.

Format:

```
[symbol]  WRITE  [iopcb][,LU=][,ADDR=][,RECNUMB=]
                 [,RECL=][,ENDADDR=][,EOF=][,EOM=]
                 [,OPT=][,ERR=][,END=][,RESTART=]
                 [,UPEXIT=][,PAUS=][,XOPT=]
```

Parameter Values:

```
iopcb     -  addrx
          -  (reg)

LU        =  absolute byte expression
          =  (reg)

ADDR      =  addrx
          =  (reg)

RECL      =  absolute address expression does not change
             IO.RECL
          =  (reg)

ENDADDR   =  addrx
          =  (reg)

EOF       =  addrx

EOM       =  addrx

OPT       =  B  (binary)                   (specified in
          =  P  (proceed)                  )any order
          =  I  (image)                    )enclosed in
          =  UP (unconditional proceed)    (parentheses
          =  R  (random)

RECNUMB   =  absolute fullword expression
          =  (reg)

ERR       =  addrx
```

RESTART	=	addrx
UPEXIT	=	addrx
XOPT	=	absolute fullword expression
	=	(reg)
PAUS	=	N

Default Values:

RESTART	=	retry the WRITE
FUN	=	0
LU	=	0
STAT	=	0
DN	=	0
ADDR	=	0
ENDADDR	=	ADDR+RECL-1 if both are specified
	=	0 otherwise
RECNUMB	=	0
TRANS	=	0
XOPT	=	0
RECL	=	0
RESTART	=	0
PAUS	=	pause if error
ERR	=	PAUS flag if no default in previous IOERRTBL
	=	default IOERRTBL of previous IOERRTBL

NOTE

On a proceed call, the status is set to positive one for polling; RECNUMB= or OPT=R, or both, cause a random WRITE.

Error Messages:

CODE	MESSAGE
IF	I/O ERROR C0XX LU X - ILLEGAL FUNCTION
DU	I/O ERROR A0XX LU X - DEVICE UNAVAILABLE
EOM	I/O ERROR 90XX LU X - END OF MEDIUM
EOF	I/O ERROR 88XX LU X - END OF FILE
UERR	I/O ERROR 84XX LU X - UNRECOVERABLE ERROR
RERR	I/O ERROR 82XX LU X - PARITY OR RECOVERABLE ERROR
LU	I/O ERROR 81XX LU X - ILLEGAL OR UNASSIGNED
UNERR	I/O ERROR XXXX LU X - UNKNOWN ERROR

CHAPTER 5
TASK MANAGEMENT MACROS

5.1 INTRODUCTION

Task management macros manipulate tasks. Through these macros, foreground tasks can extract control over and can communicate with other tasks.

The formats, parameter values, default values, required parameters, programming considerations, examples, and error messages are supplied for each task management macro.

Section 1.4, Parameter Field Value Mnemonics, explains the lowercase abbreviations that appear in the parameter field of task management macros.

5.2 CANCEL A TASK (CANCEL)

The CANCEL macro cancels a task; if it is nonresident, it removes the task from memory.

Format:

```
    [symbol]  CANCEL  [tmpcb][,TASKID=][,DIR=][,OPT=]
```

Parameter Values:

tmpcb	-	addrx (address or pointer to PCB)
	-	(reg)
TASKID	=	addrx (address or pointer to TASKID)
	=	(reg)
DIR	=	OT (direction - other task)
	=	SD (direction - self-directed)
OPT	=	S (save in memory)
	=	D (delete from memory)

Default Values:

tmpcb	-	TMPCB built automatically
TASKID	=	no change
DIR	=	OT (other task)
OPT	=	S

Programming Considerations:

If tmpcb is specified, the function is set according to DIR=. If a TASKID is specified, it is moved into the tmpcb; if TASKID is not specified, it is assumed to be in the tmpcb. The TASKID must be left-justified in an 8-byte field padded with blanks and fullword boundary aligned. If a tmpcb is not specified, it is automatically built and set as previously stated. The TMPCB macro can build the tmpcb. Refer to Section 5.25 for an explanation of the TMPCB macro.

R14 points to the tmpcb; R15 modifies it. Neither register can be used for addressing. Task execution is halted. If the task is resident and OPT=S, the task remains in memory and all task LUs are checkpointed, not closed.

If the task is nonresident and OPT=S, the task is removed from memory and all LUs are closed.

If the task is resident and OPT=D, the task is made nonresident and removed from memory, and all LUs are closed.

If the task is nonresident and OPT=D, the task is removed from memory and all LUs are closed.

5.3 CHANGE PRIORITY (CHPRIO)

The CHPRIO macro changes the priority of a directed task.

Format:

```
    [symbol]  CHPRIO  [tmpcb][,TASKID=][,DIR=][,PRI=]
```

Parameter Values:

tmpcb	-	addrx (address or pointer to PCB)
	-	(reg)
TASKID	=	addrx (address or pointer to TASKID)
	=	(reg)
DIR	=	OT (direction - other task)
	=	SD (direction - self-directed)
PRI	=	absolute byte expression
	=	(reg) - new priority

Default Values:

tmpcb	-	TMPCB built automatically
TASKID	=	no change
DIR	=	OT (other task)
PRI	=	no change

Programming Considerations:

If tmpcb is specified, the function is set according to DIR. If a TASKID is specified, it is moved into the tmpcb; if it is not specified, it is assumed to be in the tmpcb. The TASKID must be left-justified in an 8-byte field padded with blanks and fullword boundary aligned. If a tmpcb is not specified, it is automatically built and set as previously stated. The TMPCB macro can build the tmpcb.

R14 points to the tmpcb; R15 modifies it. Neither register can be used for addressing.

The CHPRIO macro changes the directed task's current priority to the user-specified priority, PRI.

5.4 CHECK THE STATUS OF A TASK (CKTASK)

The CKTASK macro checks the status of the directed task.

Format:

```
    [symbol]  CKTASK  [tmpcb][,TASKID=][,DIR=]
```

Parameter Values:

tmpcb	-	addrx (address or pointer to PCB)
	-	(reg)
TASKID	=	addrx (address or pointer to TASKID)
	=	(reg)
DIR	=	OT (direction - other task)
	=	SD (direction - self-directed)

Default Values:

tmpcb	-	TMPCB built automatically
TASKID	=	no change
DIR	=	OT (other task)

Programming Considerations:

If tmpcb is specified, the function is set according to DIR. If a TASKID is specified, it is moved into the tmpcb; if it is not specified, it is assumed to be in the tmpcb. The TASKID must be left-justified in an 8-byte field padded with blanks and fullword boundary aligned. If a tmpcb is not specified, it is automatically built and set as previously stated. The TMPCB macro can build the tmpcb. R14 points to the tmpcb; R15 modifies it. Neither register can be used for addressing.

The CKTASK macro checks the directed task status. These fields are set in the calling task's parameter block:

TM.TST	wait status
TM.RPI	current priority
TM.STA	error status

These fields are also set on any other task-directed macro. The CKTASK macro provides no other functions.

5.5 CONNECT A TRAP GENERATING DEVICE TO A TASK (CONNECT)

The CONNECT macro connects the trap-generating device that DMN specifies to the directed task. CONNECT does not enable traps. Refer to the THAW macro discussed in Section 5.24.

Format:

```
    [symbol]  CONNECT  [tmpcb][,TASKID=][,DIR=]
                       [,DMN=][,PARM=]
```

Parameter Values:

tmpcb	-	addrx (address or pointer to PCB)
	-	(reg)
TASKID	=	addrx (address or pointer to TASKID)
	=	(reg)
DIR	=	OT (direction - other task)
	=	SD (direction - self-directed)
DMN	=	addrx (address or pointer to 4-byte device mnemonic)
	=	(reg)
PARM	=	absolute address expression
	=	(reg) - register containing parameter

Default Values:

tmpcb	-	TMPCB built automatically
TASKID	=	no change
DIR	=	OT (other task)
DMN	=	no change
PARM	=	no change

Programming Considerations:

If tmpcb is specified, the function is set according to DIR. If a TASKID is specified, it is moved into the tmpcb; if it is not specified, it is assumed to be in the tmpcb. The TASKID must be left-justified in an 8-byte field padded with blanks and fullword boundary aligned. If tmpcb is not specified, it is automatically built and set as previously stated. The TMPCB macro can build the tmpcb.

R14 points to the tmpcb; R15 modifies it. Neither register can be used for addressing.

Before the connection is made, these conditions must exist:

- DMN must be a trap-generating device.
- DMN must not currently be connected to the directed task or any other task. It can be connected to only one task at a time; however, a task can be connected to more than one trap-generating device at the same time.
- The directed task must set up the UDL with a SETUDL macro with the DIQ code specified in the TSKN option and then enable traps with a LTSW macro.

  Example:

  ```
  SETUDL TSKN=(DIQ,addrx),TSKQ=addrx

  LTSW  DIQ,TSKE
  ```

5.6 DISABLE INTERRUPTS ON A TRAP-GENERATING DEVICE (FREEZE)

The FREEZE macro disables interrupts on DMN that are connected to the directed task. The system first ensures that the trap-generating device and directed task are connected. It then disables interrupts. When the FREEZE macro disables interrupts, the trap-generating device remains connected; but, all generated interrupts are lost. If interrupts are already disabled, FREEZE has no effect.

Format:

```
    [symbol]  FREEZE  [tmpcb],[,TASKID=][,DIR=][,DMN=]
```

Parameter Values:

tmpcb	-	addrx (address or pointer to PCB)
	-	(reg)
TASKID	=	addrx (address or pointer to TASKID)
	=	(reg)
DIR	=	OT (direction - other task)
	=	SD (direction - self-directed)
DMN	=	addrx (address or pointer to 4-byte device mnemonic)
	=	(reg)

Default Values:

tmpcb	-	TMPCB built automatically
TASKID	=	no change
DIR	=	OT (other task)
DMN	=	no change

Programming Considerations:

If tmpcb is specified, the function is set according to DIR. If a TASKID is specified, it is moved into the tmpcb; if it is not specified, it is assumed to be in the tmpcb. The TASKID must be left-justified in an 8-byte field padded with blanks and fullword boundary aligned. If a tmpcb is not specified, it is automatically built and set as previously stated. The TMPCB macro can build the tmpcb.

R14 points to the tmpcb; R15 modifies it. Neither register can be used for addressing.

5.7 LOAD A TASK INTO MEMORY (LOAD)

The LOAD macro loads a task into memory. It does not start the task. Refer to the START macro discussed in Section 5.22.

Format:

```
    [symbol]  LOAD  [tmpcb][,TASKID=][,DIR=][,LU=]
                    [,OPT=][,SIZE=]
```

Parameter Values:

tmpcb	-	addrx (address or pointer to PCB)
	-	(reg)
TASKID	=	addrx (address or pointer to TASKID)
	=	(reg)
DIR	=	OT (direction - other task)
LU	=	absolute byte expression
	=	(reg) - logical unit
OPT	=	CM (intertask communication)
	=	RP (subtask reporting)
	=	SZ (segment size increment)
	=	PR (load and proceed)
	=	ET (prevent E-task load)
	=	CT (intertask control)
	=	RL (roll)
	=	NO (no option)
	=	(reg) - register containing the options
SIZE	=	absolute address expression
	=	(reg) - increment size

Default Values:

tmpcb	-	TMPCB built automatically
TASKID	=	no change
DIR	=	OT (other task)
LU	=	no change
OPT	=	no change
SIZE	=	no change

Programming Considerations:

If tmpcb is specified while the directed task is being loaded, the function is set according to DIR. If a TASKID is specified, it is moved into the tmpcb; if it is not specified, it is assumed to be in the tmpcb. The TASKID must be left-justified in an 8-byte field padded with blanks and fullword boundary aligned. If a tmpcb is not specified, it is automatically built and set as previously stated. The TMPCB macro can build the tmpcb.

R14 points to the tmpcb; R15 modifies it. Neither register can be used for addressing.

Before issuing the LOAD macro, the directed task must be assigned to the LU with an ASSIGN macro. The LU must be positioned to the first byte of the task's LIB. When the LOAD macro is executed, the directed task is loaded from the specified LU into a memory area large enough to hold the task. If that area does not exist and the roll option is specified, the directed task is rolled out to a file on the roll volume and is placed in a wait state. While the directed task is being loaded, the calling task is placed in a wait state. When the directed task is loaded, its task name becomes the name specified in the TASKID parameter or the name specified in the task name field of the parameter block.

The calling task is released from the wait state and the LU is positioned to the byte following the loaded task. If the same task is to be reloaded with the same assigned LU, the LU must be rewound by using the REWIND macro prior to each subsequent load.

If these error conditions occur, LOAD is rejected and an error code is stored in the parameter block's error status field:

- The receiving task is already loaded into memory.
- The specified task name is invalid.
- The macro is self-directed.
- The system does not have a large enough memory area to hold the receiving task and it does not support the roll option.
- The requested memory size specified where the task is to be loaded, is larger than the system's total memory space.
- The directed task is a background task. (Background tasks can only be loaded from the system console.)
- The LU is not positioned to LIB, or LIB is invalid.
- The following options can be specified in any order enclosed in parentheses. If only one option is coded, the parentheses must be omitted:

CM	indicates that the loaded directed task can execute communications functions.
RP	indicates that the calling task becomes a monitor task and the directed task becomes a subtask, causing the subtask to report all status changes during execution to the monitor task through task traps.
SZ	indicates that the task's impure segment size is increased by adding the number of bytes the SIZE parameter specifies.
PR	indicates that the calling task continues executing while the directed task is being loaded. If the latest LTSW macro specifies the LODQ option, a trap to the calling task occurs when loading is completed.
ET	indicates that the directed task cannot be an E-task.
CT	indicates that the directed task loaded into memory can issue SVC 6 control functions.
NO	indicates that no options are desired.

5.8 LOAD A TASK STATUS WORD (LTSW)

The LTSW macro sets or replaces the current task status word (TSW) located in the task's TCB with a new user-specified TSW.

Format:

```
    [symbol]  LTSW  [option,...,option][,CC=][,LOC=][,PCB=]
                    [,FORM=]
```

Parameter Values:

option	-	(reg)
	-	absolute fullword expression
	-	WT (trap wait)
	-	PWRE (power restore trap enable)
	-	ARFE (arithmetic fault trap enable)
	-	S14E (SVC 14 trap enable)
	-	TSKE (task queue service trap enable)
	-	MAFE (memory access fault trap enable)
	-	IITE (illegal instruction trap enable)
	-	SUQ (enable subtask queue entries for subtask state change)
	-	DIQ (enable task queue on device interrupt)
	-	TCQ (enable task queue entry on task call)
	-	TMQ (enable task queue entry on task message)
	-	LODQ (enable task queue entry on completion of load and proceed)
	-	IOQ (enable task queue entry on I/O completion)
	-	TMCQ (enable task queue entry on time out completion)
	-	ITQ (enable task queue entry on SVC 15 buffer transfer command execution, termination or halt I/O)
	-	TETS (enable trap event service routines)
CC	=	condition code (absolute expression less than 16)
LOC	=	addrx (transfer location)
	=	(reg)

PCB	=	addrx
	=	(reg)
FORM	=	L

Default Values:

option	-	bits not set
LOC	=	0

Programming Considerations:

If specified as codes, the options are specified as positional parameters in any order. They cannot be used with (reg) or an absolute fullword expression. The condition code (CC) can only be specified with the codes. If CC is specified without any codes, all interrupts are disabled; that is, all code bits are reset to zero. The condition code cannot be specified if the codes are specified in (reg) or as an absolute expression.

If FORM=L is specified, the parameter block is built according to the options. IF LOC= is omitted, the parameter block is set to zero.

If PCB= is specified, an existing parameter block is assumed. If specified, the options set new options, regardless of previous options in the existing block. If specified, LOC= replaces the previous transfer location; if it is not specified, the existing transfer location is used. The new TSW is loaded.

If neither PCB= nor FORM=L are specified, a parameter block is built according to the options, LOC= and FORM=L, and the new TSW is loaded.

5.9 MAKE A TASK NONRESIDENT (MAKNRES)

The MAKNRES macro makes the directed task nonresident regardless of the options specified at Link time. Once nonresident, the task can be rolled if the system supports the roll option.

Format:

```
    [symbol]  MAKNRES  [tmpcb][,TASKID=][,DIR=]
```

Parameter Values:

tmpcb	-	addrx (address or pointer to PCB)
	-	(reg)
TASKID	=	addrx (address or pointer to TASKID)
	=	(reg)
DIR	=	OT (direction - other task)
	=	SD (direction - self-directed)

Default Values:

tmpcb	-	TMPCB built automatically
TASKID	=	no change
DIR	=	OT (other task)

Programming Considerations:

If tmpcb is specified, the function is set according to DIR. If a TASKID is specified, it is moved into the tmpcb; if it is not specified, it is assumed to be in the tmpcb. The TASKID must be left-justified in an 8-byte field padded with blanks and fullword boundary aligned. If a tmpcb is not specified, it is automatically built and set as previously stated. The TMPCB macro can build the tmpcb.

R14 points to the tmpcb; R15 modifies it. Neither register can be used for addressing.

5.10 MAKE A TASK NONROLLABLE (MAKNROLL)

The MAKNROLL macro restricts the directed task from being rolled.

Format:

```
    [symbol]  MAKNROLL  [tmpcb][,TASKID=][,DIR=]
```

Parameter Values:

tmpcb	-	addrx (address or pointer to PCB)
	-	(reg)
TASKID	=	addrx (address or pointer to TASKID)
	=	(reg)
DIR	=	OT (direction - other task)
	=	SD (direction - self-directed)

Default Values:

tmpcb	-	TMPCB built automatically
TASKID	=	no change
DIR	=	OT (other task)

Programming Considerations:

If tmpcb is specified, the function is set according to DIR. If a TASKID is specified, it is moved into the tmpcb; if it is not specified, it is assumed to be in the tmpcb. The TASKID must be left-justified in an 8-byte field padded with blanks and fullword boundary aligned. If a tmpcb is not specified, it is automatically built and set as previously stated. The TMPCB macro can build the tmpcb.

R14 points to the tmpcb; R15 modifies it. Neither register can be used for addressing.

5.11 MAKE A TASK RESIDENT (MAKRES)

The MAKRES macro makes the directed task resident regardless of what options were specified at Link time. Once resident, the task cannot be rolled.

Format:

```
    [symbol]  MAKRES  [tmpcb][,TASKID=][,DIR=]
```

Parameter Values:

tmpcb	-	addrx (address or pointer to PCB)
	-	(reg)
TASKID	=	addrx (address or pointer to TASKID)
	=	(reg)
DIR	=	OT (direction - other task)
	=	SD (direction - self-directed)

Default Values:

tmpcb	-	TMPCB built automatically
TASKID	=	no change
DIR	=	OT (other task)

Programming Considerations:

If tmpcb is specified, the function is set according to DIR. If a TASKID is specified, it is moved into the tmpcb; if it is not specified, it is assumed to be in the tmpcb. The TASKID must be left-justified in an 8-byte field padded with blanks and fullword boundary aligned. If a tmpcb is not specified, it is automatically built and set as previously stated. The TMPCB macro can build the tmpcb.

R14 points to the tmpcb; R15 modifies it. Neither register can be used for addressing.

5.12 MAKE A TASK ROLLABLE (MAKROLL)

The MAKROLL macro makes the directed task rollable. However, if resident, the task is not rolled.

Format:

```
    [symbol]  MAKROLL  [tmpcb][,TASKID=][,DIR=]
```

Parameter Values:

tmpcb	-	addrx (address or pointer to PCB)
	-	(reg)
TASKID	=	addrx (address or pointer to TASKID)
	=	(reg)
DIR	=	OT (direction - other task)
	=	SD (direction - self-directed)

Default Values:

tmpcb	-	TMPCB built automatically
TASKID	=	no change
DIR	=	OT (other task)

Programming Considerations:

If tmpcb is specified, the function is set according to DIR. If a TASKID is specified, it is moved into the tmpcb; if it is not specified, it is assumed to be in the tmpcb. The TASKID must be left-justified in an 8-byte field padded with blanks and fullword boundary aligned. If a tmpcb is not specified, it is automatically built and set as previously stated. The TMPCB macro can build the tmpcb.

R14 points to the tmpcb; R15 modifies it. Neither register can be used for addressing.

5.13 BUILD A MESSAGE RING OR CHAIN OF BUFFERS (MSGRING)

The MSGRING macro builds a ring or chain of 76-byte buffers and sets the link addresses. In a ring buffer, the link address of the last buffer points to the first buffer. In a chain buffer, the link address of the last buffer is set to 0. Any number of buffers can be built.

Format:

```
    [symbol]  MSGRING  [number][,code][,LEN=]
```

Parameter Values:

number	-	integer constant (number of 76-byte buffers)
code	-	R (ring buffers)
	-	C (chain buffers)
LEN	=	integer constant (length of buffer plus link)

Default Values:

code	-	R
number	-	1
LEN	=	76 (for 32-bit assemblies)
	=	74 (for 16-bit assemblies)

5.14 ADD A PARAMETER TO THE TASK QUEUE (QUEPARM)

The QUEPARM macro adds a user-specified parameter to the directed task's task queue. The directed task must set up the UDL to receive a parameter with a SETUDL macro and enable traps with a LTSW macro.

Format:

```
    [symbol]  QUEPARM  [tmpcb][,TASKID=][,DIR=][,PARM=]
```

Parameter Values:

tmpcb	-	addrx (address or pointer to PCB)
	-	(reg)
TASKID	=	addrx (address or pointer to TASKID)
	=	(reg)
DIR	=	OT (direction - other task)
	=	SD (direction - self-directed)
PARM	=	absolute address expression
	=	(reg) - register containing parameter

Default Values:

tmpcb	-	TMPCB built automatically
TASKID	=	no change
DIR	=	OT (other task)
PARM	=	no change

Programming Considerations:

If tmpcb is specified, the function is set according to DIR. If a TASKID is specified, it is moved into the tmpcb; if it is not specified, it is assumed to be in the tmpcb. The TASKID must be left-justified in an 8-byte field padded with blanks and fullword boundary aligned. If a tmpcb is not specified, it is automatically built and set as previously stated. The TMPCB macro can build the tmpcb.

R14 points to the tmpcb; R15 modifies it. Neither register can be used for addressing.

Example:

```
    SETUDL TSKN=(TCQ,addrx),TSKQ=addrx
        .
        .
        .
    LTSW   TCQ,TSKE
```

5.15 RECEIVE A LOGICAL UNIT FROM A TASK (RECVLU)

The RECVLU macro transfers the LU currently assigned to the directed task to the calling task and then closes the LU assigned to the directed task. The calling task's LU must not be assigned. The directed task must be in a dormant or paused wait state or suspended by a SUSPEND macro.

Format:

```
[symbol]  RECVLU  [tmpcb][,TASKID=][,DIR=][,CLU=][,DLU=]
```

Parameter Values:

tmpcb	-	addrx (address or pointer to PCB)
	-	(reg)
TASKID	=	addrx (address or pointer to TASKID)
	=	(reg)
DIR	=	OT (direction - other task)
	=	SD (direction - self-directed)
CLU	=	absolute byte expression
	=	(reg) - calling LU
DLU	=	absolute byte expression
	=	(reg) - directed LU

Default Values:

tmpcb	-	TMPCB built automatically
TASKID	=	no change
DIR	=	OT (other task)
CLU	=	no change
DLU	=	no change

Programming Considerations:

If tmpcb is specified, the function is set according to DIR. If a TASKID is specified, it is moved into the tmpcb; if it is not specified, it is assumed to be in the tmpcb. The TASKID must be left-justified in an 8-byte field padded with blanks and fullword boundary aligned. If a tmpcb is not specified, it is automatically built and set as previously stated. The TMPCB macro can build the tmpcb.

R14 points to the tmpcb; R15 modifies it. Neither register can be used for addressing.

5.16 RELEASE A TASK (RELEASE)

The RELEASE macro releases the directed task, currently suspended by a previous SUSPEND macro, by taking it out of the task wait state. Once released, the directed task continues to execute. If the task is not in another wait state, executing occurs with the instruction that follows the instruction executed before the task was suspended.

Format:

```
    [symbol]  RELEASE  [tmpcb][,TASKID=][,DIR=]
```

Parameter Values:

tmpcb	-	addrx (address or pointer to PCB)
	-	(reg)
TASKID	=	addrx (address or pointer to TASKID)
	=	(reg)
DIR	=	OT (direction - other task)
	=	SD (direction - self-directed)

Default Values:

tmpcb	-	TMPCB built automatically
TASKID	=	no change
DIR	=	OT (other task)

Programming Considerations:

If tmpcb is specified, the function is set according to DIR. If a TASKID is specified, it is moved into the tmpcb; if it is not specified, it is assumed to be in the tmpcb. The TASKID must be left-justified in an 8-byte field padded with blanks and fullword boundary aligned. If a tmpcb is not specified, it is automatically built and set as previously stated. The TMPCB macro can build the tmpcb.

R14 points to the tmpcb; R15 modifies it. Neither register can be used for addressing.

5.17 RUN A TASK (RUN)

The RUN macro is a combination of the LOAD and START macros; it loads a task into memory and causes that loaded task to be executed. Both functions are performed with a single call. Refer to the LOAD macro explained in Section 5.7 and the START macro explained in Section 5.22.

Format:

```
    [symbol]  RUN  [tmpcb]  [,TASKID=][,DIR=][,LU=][,OPT=]
                            [,SIZE=][,TOD=][,INT=][,SAD=]
                            [,SOP=]
```

Parameter Values:

tmpcb	-	addrx (address or pointer to PCB)
	-	(reg) - address or pointer to PCB
TASKID	=	addrx (address or pointer to TASKID)
	=	(reg)
DIR	=	OT (direction - other task)
	=	SD (direction - self-directed)
LU	=	absolute byte expression
	=	(reg) - logical unit
OPT	=	S (start option)
	=	D (delay start)
	=	CM (intertask communication)
	=	RP (subtask reporting)
	=	SZ (segment size increment)
	=	PR (load and proceed)
	=	ET (prevent E-task load)
	=	CT (intertask control)
	=	RL (roll)
	=	NO (no options)
	=	(reg) - register containing options
SIZE	=	absolute address
	=	(reg) - increment size
TOD	=	time expression (time of day to start)
	=	(reg) - register containing the time of day in seconds from midnight. See Section 6.5.

INT	=	time expression (interval of delay start)
	=	(reg) - register containing interval in milliseconds of delay to start
SAD	=	addrx (task starting address)
	=	(reg) - register containing the starting address
SOP	=	addrx (address of start options field)
	=	(reg) - pointer to start options field

Default Values:

tmpcb	=	TMPCB built automatically
TASKID	=	no change
DIR	=	OT (other task)
OPT	=	start immediate (no load options)
TOD	=	no change
SIZE	=	no change
TOD	=	no change
INT	=	no change
SAD	=	no change
SOP	=	no change

Programming Considerations:

If tmpcb is specified while the directed task is being loaded, the function is set according to DIR. If a TASKID is specified, it is moved into the tmpcb; if it is not specified, it is assumed to be in the tmpcb. The TASKID must be left-justified in an 8-byte field padded with blanks and fullword boundary aligned. If tmpcb is not specified, it is automatically built and set as previously stated. The TMPCB macro can build the tmpcb.

R14 points to the tmpcb; R15 modifies it. Neither register can be used for addressing.

The OPT parameter must specify S, or D, or both. If specified, S and D must be enclosed in parentheses and separated by a comma in either order. To leave the parameter block (set by a TMPCB macro) unchanged, specify a null parameter (OPT=,). If OPT is omitted, an immediate start is requested. If OPT=S is specified, SOP must be coded or a valid start option address must be specified in the parameter block. If OPT=D is specified, TOD or INT, but not both, must be specified. See the GENTIME macro (Section 6.4) for a definition of a time-expression. Refer to the LOAD and START macros, Sections 5.7 and 5.22, respectively, for a more detailed explanation.

5.18 SEND A LOGICAL UNIT TO A TASK (SENDLU)

The SENDLU macro sends to the directed task the LU currently assigned to the calling task and then closes the LU assigned to the calling task. The directed task must not have the LU currently assigned and must be in the dormant or paused wait state or suspended by a SUSPEND macro.

Format:

```
    [symbol]  SENDLU  [tmpcb][,TASKID=][,DIR=][,CLU=][,DLU=]
```

Parameter Values:

tmpcb	-	addrx (address or pointer to PCB)
	-	(reg)
TASKID	=	addrx (address or pointer to TASKID)
	=	(reg)
DIR	=	OT (direction - other task)
	=	SD (direction - self-directed)
CLU	=	absolute byte expression
	=	(reg) - calling LU
DLU	=	absolute byte expression
	=	(reg) - directed LU

Default Values:

tmpcb	-	TMPCB built automatically
TASKID	=	no change
DIR	=	OT (other task)
CLU	=	no change
DLU	=	no change

Programming Considerations:

If tmpcb is specified, the function is set according to DIR. If a TASKID is specified, it is moved into the tmpcb; if it is not specified, it is assumed to be in the tmpcb. The TASKID must be left-justified in an 8-byte field padded with blanks and fullword boundary aligned. If a tmpcb is not specified, it is automatically built and set as previously stated. The TMPCB macro can build the tmpcb.

R14 points to the tmpcb; R15 modifies it. Neither register can be used for addressing.

5.19 SEND A MESSAGE (SENDMSG)

The SENDMSG macro sends a message from the calling task to the directed task by transferring the message to the directed task's message buffer and by putting an item on the directed task's task queue. The message must be 64 bytes long and fullword boundary aligned. Before sending the message to the directed task, the system appends the calling task's 8-byte TASKID to the beginning of the message. The message is sent in binary format and image mode.

Format:

```
    [symbol]  SENDMSG  [tmpcb][,TASKID=][,DIR=][,MSG=]
```

Parameter Values:

tmpcb	-	addrx (address or pointer to PCB)
	-	(reg)
TASKID	=	addrx (address or pointer to TASKID)
	=	(reg)
DIR	=	OT (direction - other task)
	=	SD (direction - self-directed)
MSG	=	addrx (address or pointer to message buffer)
	=	(reg) - address or pointer to message buffer

Default Values:

tmpcb	-	TMPCB built automatically
TASKID	=	no change
DIR	=	OT (other task)
MSG	=	no change

Programming Considerations:

If tmpcb is specified, the function is set according to DIR. If a TASKID is specified, it is moved into the tmpcb; if it is not specified, it is assumed to be in the tmpcb. The TASKID must be left-justified in an 8-byte field padded with blanks and fullword boundary aligned. If a tmpcb is not specified, it is automatically built and set as previously stated. The TMPCB macro can build the tmpcb.

R14 points to the tmpcb; R15 modifies it. Neither register can be used for addressing.

Example:

The directed task must accept a message by setting up the UDL with a SETUDL macro and enable message traps with a LTSW macro:

```
SETUDL TSKN=(PMQ,addrx),TSKQ=addrx
    .
    .
    .
LTSW  PMQ,TSKE
```

5.20 INITIALIZE OR MODIFY A USER DEDICATED LOCATION (SETUDL)

The SETUDL macro dynamically sets or modifies the user dedicated locations (UDLs) with addresses of user-supplied trap routines and new task status words to service the various traps.

Format:

```
    [symbol]  SETUDL  [TSKQ=][,MSGR=][,PWRN=][,ARFN=]
                      [,S14N=][,TSKN=][,MAFN=][,IITN=]
```

Parameter Values:

```
    TSKQ  = addrx (task queue address)
          = (reg) - pointer to task queue
          = integer constant (task queue size)

    MSGR  = addrx (message ring address)
          = (reg) - pointer to message ring
          = integer constant (number of 76-byte buffers
            in message ring)

    PWRN  = addrx (power restore address; new TSW)
          = (code,...,code,addrx) power restore new TSW

    ARFN  = addrx (arithmetic fault address; new TSW)
          = (code,...,code,addrx) arithmetic fault new TSW

    S14N  = addrx (SVC 14 address; new TSW)
          = (code,...,code,addrx) SVC 14 new TSW

    TSKN  = addrx (task queue address; service routine TSW)
          = (code,...,code,addrx) - task queue service
            routine TSW

    MAFN  = addrx (memory access fault address; new TSW)
          = (code,...,code,addrx) - memory access fault new
            TSW

    IITN  = addrx (illegal instruction address; new TSW)
          = (code,...,code,addrx) - illegal instruction new
            TSW
```

where code is any of these states:

WT	trap wait
PWRE	power restore trap enable
ARFE	arithmetic fault trap enable
S14E	SVC 14 trap enable
TSKE	task queue service trap enable
MAFE	memory access fault trap enable
IITE	illegal instruction trap enable
SUQ	enable subtask queue entry
DIQ	enable device interrupt task queue entry
TCQ	enable task call task queue entry
TMQ	enable task message task queue entry
LODQ	enable completion of load and proceed task queue entry
IOQ	enable I/O completion task queue entry
TMCQ	enable time out completion task queue entry
ITQ	enable SVC 15 buffer transfer, termination or wait I/O task queue entry

Programming Considerations:

The SETUDL macro can initialize or modify the UDL. If the operand of the TSKQ parameter is an address or register pointer, assume a DLIST assembler instruction built the actual queue. The address of the DLIST assembler instruction is stored in the UDL. If an integer constant is specified, DLIST is automatically generated and branched around. The DLIST address is stored in the UDL. The program can access this address by:

```
    LDA    R1,UDL.TSKQ
    RTL    R2,0(R1)
```

If the parameter of the MSGR pointer specifies an address or a register pointer, assume that a MSGRING macro built the message ring. That value is stored in the UDL. If an integer constant is specified, that number of buffers is built into a ring, branched around, and the first buffer's address is stored in the UDL. The buffers' link fields are set into a ring.

If the operand of any PWRN, ARFN, S14N, TSKN, MAFN, IITN parameter is an address, assume the list form of the LTSW macro built the new TSW and that the TSW is stored in the UDL. If the parameter is specified as (code,...code,addrx), a TSW is built in R14 and R15 and stored in the UDL.

5.21 SIMULATE AN INTERRUPT ON A TRAP-GENERATING DEVICE (SIMINT)

The SIMINT macro simulates an interrupt on a specified trap-generating device connected to the directed task. The THAW macro must have been issued to the directed task. If interrupts are disabled, there is no effect. The system ensures that the trap-generating device and the directed task are connected. It then simulates interrupts.

Format:

```
    [symbol]  SIMINT  [tmpcb][,TASKID=][,DIR=][,DMN=]
```

Parameter Values:

tmpcb	-	addrx (address or pointer to PCB)
	-	(reg)
TASKID	=	addrx (address or pointer to TASKID)
	=	(reg)
DIR	=	OT (direction - other task)
	=	SD (direction - self-directed)
DMN	=	addrx (address or pointer to 4-byte device mnemonic)
	=	(reg) - address or pointer to 4-byte device mnemonic

Default Values:

tmpcb	-	TMPCB built automatically
TASKID	=	no change
DIR	=	OT (other task)
DMN	=	no change

Programming Considerations:

If tmpcb is specified, the function is set according to DIR. If a TASKID is specified, it is moved into the tmpcb; if it is not specified, it is assumed to be in the tmpcb. The TASKID must be left-justified in an 8-byte field padded with blanks and fullword boundary aligned. If a tmpcb is not specified, it is automatically built and set as previously stated.

R14 points to the tmpcb; R15 modifies it. Neither register can be used for addressing.

5.22 START EXECUTION OF A TASK (START)

The START macro causes a task, which has been loaded into memory, to be executed. Options enable a delayed start, a start at a specific time of day or after an interval, a start at a specific address, and the passing of start options to the task.

Format:

```
[symbol]  START  [tmpcb][,TASKID=][,DIR=][,OPT=]
                 [,TOD=][,INT=][,SAD=][,SOP=]
```

Parameter Values:

tmpcb	-	addrx (address or pointer to PCB)
	-	(reg)
TASKID	=	addrx (address or pointer to TASKID)
	=	(reg)
DIR	=	OT (direction - other task)
	=	SD (direction - self-directed)
OPT	=	S (start options)
	=	D (delay start)
TOD	=	time expression (time of day to start)
	=	(reg) - register containing time of day in seconds from midnight. See Section 6.5.
INT	=	time expression (interval of delay to start)
	=	(reg) - register containing interval in milliseconds of delay to start. See Section 6.5.
SAD	=	addrx - (task starting address)
	=	(reg) - register containing starting address
SOP	=	addrx - address or pointer to start options field
	=	(reg) - address or pointer to start options field

Default Values:

tmpcb	-	TMPCB built automatically
TASKID	=	no change
DIR	=	OT (other task)
OPT	=	start immediate
TOD	=	no change
INT	=	no change
SAD	=	no change
SOP	=	no change

Programming Considerations:

If tmpcb is specified, the function is set according to DIR. If a TASKID is specified, it is moved into the tmpcb; if it is not specified, it is assumed to be in the tmpcb. The TASKID must be left-justified in an 8-byte field padded with blanks and fullword boundary aligned. If a tmpcb is not specified, it is automatically built and set as previously stated. The TMPCB macro can build the tmpcb.

R14 points to the tmpcb; R15 modifies it. Neither register can be used for addressing.

The OPT parameter must specify S, or D, or both. If specified, S and D must be enclosed in parentheses and separated by a comma in either order. To leave the parameter block (which was set by a TMPCB macro, or some other macro) unchanged, specify a null parameter (OPT=,). If OPT is omitted, an immediate start is requested. If OPT=S is specified, SOP must be coded or a valid start option address must be specified in the parameter block. If OPT=D is specified, TOD or INT, but not both, must be specified. See the GENTIME macro (Section 6.4) for a definition of a time-expression.

- start immediate - no start options

  - The OPT parameter must be omitted. The SAD parameter specifies the directed task starting address. If the starting address is 0, the task is started at the address established at Link time. |

- start immediate with start options

  - Start options, optionally specified in certain languages and utility programs at execution time, are also included as run-time information when the directed task starts execution. When the start function is executed, start options located at the address specified in the parameter block are stored into the directed task's user top of program (UTOP) area. If sufficient memory is not available between UTOP and core top of memory (CTOP), the macro is rejected and an error code is stored in the parameter block's error status field. The task should then be reloaded into a larger segment using the SIZE parameter of the LOAD macro. Refer to Section 5.7 for a detailed explanation of the LOAD macro.

  - The user-specified start options must be loaded on a fullword boundary. The maximum length of start options are defined at SYSGEN time through the CMDLENGTH option. If the start options' length is greater than that length specified at SYSGEN time or a carriage return is present within start options, only those characters up to the maximum number or the carriage return are stored in the task's UTOP area.

NOTE

The start options field address is also the message buffer field address in the parameter block. The contents of this field are always assumed to be the start option address when the start function is specified.

- delayed start

  - The directed task starts execution after a user-specified interval elapses. The interval can be specified as time of day (TOD) or an interval in milliseconds (INT). If neither TOD or INT are specified, the interval is assumed to be in the parameter block.

  - Before the start function can be executed for the directed task, bytes 192 through 251 of the UDL must be reserved for the delayed start function's use.

  - When the start function is executed, the directed task is immediately placed into a time wait state. When the interval elapses, the directed task starts execution.

- delay start function with start options

  - When this function is specified, the directed task starts execution after a user-specified interval elapses. This interval, which is located in the parameter block's increment of time and count fields, can be specified as time of day or interval timing interval.

  - Before this start function can be executed for the directed task, bytes 192 through 251 of the UDL must be reserved for the delay start function's use.

  - When this start function is executed, the start options, located at the address specified in the parameter block, are stored into the directed task's UTOP area and the directed task is immediately placed into a time wait state. If sufficient memory is not available between UTOP and CTOP, this call is rejected and an error code is stored in the parameter block's error status field. The task should then be reloaded into a larger segment using the SIZE parameter of the LOAD macro. Refer to Section 5.7 detailing the LOAD macro.

  - The user-specified start options must be located on a fullword boundary. The maximum length of the start options is defined at SYSGEN time through the CMDLENGTH option. If the length of the start options is greater than that length specified at SYSGEN time or a carriage return is present within the start options, only those characters up to the maximum number or the carriage return are stored in the task's UTOP area. Since the start options' field address is also the message buffer field address in the parameter block, this field's contents are always assumed to be the start options address when the start function is specified. When the user-specified interval elapses, the directed task starts execution.

5.23 PLACE A TASK IN THE WAIT STATE (SUSPEND)

The SUSPEND macro places the directed task in the task wait state. The directed task remains in the wait state until another task releases it. If the task is self-directed, it causes the calling task to suspend itself. To release the calling task from the task wait state, another task must be available to subsequently release it.

Format:

```
    [symbol]  SUSPEND  [tmpcb][,TASKID=][,DIR=]
```

Parameter Values:

tmpcb	-	addrx (address or pointer to PCB)
	-	(reg)
TASKID	=	addrx (address or pointer to TASKID)
	=	(reg)
DIR	=	OT (direction - other task)
	=	SD (direction - self-directed)

Default Values:

tmpcb	-	TMPCB built automatically
TASKID	=	no change
DIR	=	OT (other task)

Programming Considerations:

If tmpcb is specified, the function is set according to DIR. If a TASKID is specified, it is moved into the tmpcb; if it is not specified, it is assumed to be in the tmpcb. The TASKID must be left-justified in an 8-byte field padded with blanks and fullword boundary aligned. If a tmpcb is not specified, it is automatically built and set as previously stated. The TMPCB macro can build the tmpcb.

R14 points to the tmpcb; R15 modifies it. Neither register can be used for addressing.

5.24 ENABLE INTERRUPTS ON A CONNECTED TRAP-GENERATING DEVICE (THAW)

The THAW macro ensures that the trap-generating device and the directed task are connected. Refer to the CONNECT macro detailed in Section 5.5. The THAW macro then enables interrupts. The UDL can be established by using the SETUDL macro. Interrupts are disabled when the directed task terminates or if an UNCONN or FREEZE macro is directed to the task. If a THAW macro is issued when interrupts are already enabled, the macro has no effect.

Format:

```
    [symbol]  THAW  [tmpcb][,TASKID=][,DIR=][,DMN=][,PARM=]
```

Parameter Values:

tmpcb	-	addrx (address or pointer to PCB)
	-	(reg)
TASKID	=	addrx (address or pointer to TASKID)
	=	(reg)
DIR	=	OT (direction - other task)
	=	SD (direction - self-directed)
DMN	=	addrx - address or pointer to 4-byte device mnemonic
PARM	=	absolute address expression
	=	(reg) - register containing parameter

Default Values:

tmpcb	-	TMPCB built automatically
TASKID	=	no change
DIR	=	OT (other task)
DMN	=	no change
PARM	=	no change

Programming Considerations:

If tmpcb is specified, the function is set according to DIR. If a TASKID is specified, it is moved into the tmpcb; if it is not specified, it is assumed to be in the tmpcb. The TASKID must be left-justified in an 8-byte field padded with blanks and fullword boundary aligned. If a tmpcb is not specified, it is automatically built and set as previously stated. The TMPCB macro can build the tmpcb.

R14 points to the tmpcb; R15 modifies it. Neither register can be used for addressing.

5.25 CONSTRUCT A TASK PARAMETER CONTROL BLOCK (TMPCB)

The TMPCB macro constructs the parameter block for task management macros. It can be constructed alone or as part of the expansion of other task management macros.

Format:

```
[symbol]  TMPCB  [TASKID=][,DIR=][,OPT=][,PRI=][,DMN=]
                 [,PARM=][,LU=][,SIZE=][,MSG=][,SAD=]
                 [,TOD=][,INT=][,CLU=][,DLU=][,SOP=]
```

Parameter Values:

TASKID	=	'quoted string' (the task's name specified as 1- to 8-alphanumeric characters, the first of which must be a letter)
DIR	=	OT (other task)
	=	SD (direction - self-directed)
OPT	=	CM (intertask communication)
	=	RP (subtask reporting)
	=	SZ (segment size increment)
	=	PR (load and proceed)
	=	ET (prevent E-task load)
	=	CT (intertask control)
	=	RL (roll)
	=	S (start options for LOAD)
	=	D (delayed start)
PRI	=	abs byte exp (priority)
DMN	=	'quoted string' (one- to four-character device mnemonic)
PARM	=	addr (address of parameter to be queued)
LU	=	abs byte exp (logical unit number)
SIZE	=	abs addr exp (size increment)
MSG	=	addr (message buffer address)
SAD	=	addr (start address)

TOD	= time expression (time of day to start)
INT	= time expression (increment of time to start)
CLU	= abs byte exp (calling LU)
DLU	= abs byte exp (directed LU)
SOP	= addr (address of start options)

5.26 EXIT TRAP EVENT SERVICE ROUTINE (TEXIT)

A task event service routine is executed as a result of a task event trap. To terminate the execution of a task event service routine, issue the TEXIT macro.

Format:

```
[symbol]   TEXIT   [PCB=][FORM=]
```

Parameter Values:

PCB	=	addrx (address or pointer to parameter control block)
	=	(reg) - address or pointer to parameter control block
FORM	=	L (list form - only build PCB)

Programming Considerations:

When the TEXIT macro is issued, the following sequence takes place:

- TSW location that was saved at the time of interruption is restored.
- Registers are restored according to the selected Link options NONE, ALL, TEQSAVE.

5.27 GENERATE A USER DEDICATED LOCATION STRUCTURE AND EQUATES (UDLS)

The UDLS macro generates the STRUC and EQUs for user dedicated locations (UDL).

Format:

```
    blank     UDLS     blank
```

Structure Generated:

```
    UDLS       STRUC
    UDL.CTOP   DS    4            CTOP
    UDL.UTOP   DS    4            UTOP
    UDL.UBOT   DS    4            UBOT
               DS    4            RESERVED
    UDL.TSKQ   DS    4            A(TASK QUEUE)
               DS    4
    UDL.MSGR   DS    4            A(MESSAGE RING)
    UDL.SV14   DS    4            A(SVC 14 ARG)
               DS    16           RESERVED
    UDL.PWRO   DS    8            POWER RESTORATION OLD TSW
    UDL.PWRN   DS    8            POWER RESTORATION NEW TSW
    UDL.ARFO   DS    8            ARITHMETIC FAULT OLD TSW
    UDL.ARFN   DS    8            ARITHMETIC FAULT NEW TSW
               DS    8
               DS    8
    UDL.S14O   DS    8            SVC 14 OLD TSW
    UDL.S14N   DS    8            SVC 14 NEW TSW
    UDL.TSKO   DS    8            TASK QUEUE SERVICE OLD TSW
    UDL.TSKN   DS    8            TASK QUEUE SERVICE NEW TSW
    UDL.MAFO   DS    8            MEMORY ACCESS FAULT OLD TSW
    UDL.MAFN   DS    8            MEMORY ACCESS FAULT NEW TSW
    UDL.IITO   DS    8            ILLEGAL INSTRUCTION OLD TSW
    UDL.IITN   DS    8            ILLEGAL INSTRUCTION NEW TSW
               DS    16*2         RESERVED
    UDL.AIDS   DS    64           RESERVED FOR AIDS
               ENDS
```

Equates Generated:

```
*
*   TASK STATUS WORD EQUATES
*
TSW.WTM   EQU    Y'80000000'   TRAP WAIT
TSW.WTB   EQU    0
TSW.PWRM  EQU    Y'40000000'   POWER RESTORATION TRAP ENABLE
TSW.PWRB  EQU    1
TSW.AFM   EQU    Y'20000000'   ARITH FAULT TRAP ENABLE
TSW.AFB   EQU    2
TSW.S14M  EQU    Y'10000000'   SVC 14 TRAP ENABLE
TSW.S14B  EQU    3
TSW.TSKM  EQU    Y'08000000'   TASK QUEUE SERVICE TRAP ENABLE
TSW.TSKB  EQU    4
TSW.MAFM  EQU    Y'04000000'   MEMORY ACCESS FAULT TRAP ENABLE
TSW.MAFB  EQU    5
TSW.IITM  EQU    Y'02000000'   ILLEGAL INSTRUCTION TRAP ENABLE
TSW.IITB  EQU    6
TSW.SUQM  EQU    Y'00010000'   SUBTASK QUEUE ENTRY ENABLE
TSW.SUQB  EQU    15
TSW.DIQM  EQU    Y'00008000'   QUEUE ENTRY DEVICE INTERRUPT
TSW.DIQB  EQU    16
TSW.TCM   EQU    Y'00004000'   QUEUE ENTRY TASK CALL
TSW.TCB   EQU    17
TSW.PMM   EQU    Y'00001000'   QUEUE ENTRY PEER TASK MESSAGE
TSW.PMB   EQU    19
TSW.LODM  EQU    Y'00000800'   LOAD PROCEED QUEUE ENTRY ENABLE
TSW.LODB  EQU    20
TSW.IOM   EQU    Y'00000400'   QUEUE ENTRY I/O PROCEED
                               TERMINATION
TSW.IOB   EQU    21
TSW.TMCM  EQU    Y'00000200'   QUEUE ENTRY TIMEOUT COMPLETION
TSW.TMCB  EQU    22
TSW.ITM   EQU    Y'00000100'   ITAM BIT
TSW.ITB   EQU    23
TSW.LOC   EQU    4             DISPLACEMENT OF LOC FULLWORD
```

NOTE

UDLS is automatically generated in the expansion of the FETPTR macro.

5.28 DISCONNECT A TRAP-GENERATING DEVICE (UNCONN)

The UNCONN macro disconnects a specified DMN that is connected to the directed task. The system first ensures that the trap-generating device and directed task are connected. It then disables all interrupts and disconnects the device from the directed task. The device can now be connected to another task.

Format:

```
    [symbol]  UNCONN  [tmpcb][,TASKID=][,DIR=][,DMN=]
```

Parameter Values:

tmpcb	-	addrx (address or pointer to PCB)
	-	(reg)
TASKID	=	addrx (address or pointer to TASKID)
	=	(reg)
DIR	=	OT (direction - other task)
	=	SD (direction - self-directed)
DMN	=	addrx (address or pointer to 4-byte device mnemonic)
	=	(reg)

Default Values:

tmpcb	-	TMPCB built automatically
TASKID	=	no change
DIR	=	OT (other task)
DMN	=	no change

Programming Considerations:

If tmpcb is specified, the function is set according to DIR. If a TASKID is specified, it is moved into the tmpcb; if it is not specified, it is assumed to be in the tmpcb. The TASKID must be left-justified in an 8-byte field padded with blanks. No boundary alignment is required. If a tmpcb is not specified, it is automatically built and set as previously stated. The TMPCB macro can build the tmpcb.

R14 points to the tmpcb; R15 modifies it. Neither register can be used for addressing.

CHAPTER 6
TIMER MANAGEMENT MACROS

6.1 INTRODUCTION

Timer management macros can cancel time interval requests, schedule traps cyclically at different times, read the remaining time for an interval to elapse, generate a time interval, build a table of time intervals, schedule an interrupt by adding a parameter to a task queue when a specified interval has elapsed, and wait for a specific interval to elapse.

The formats, parameter values, default values, required parameters, programming considerations, examples, and error messages are supplied for each timer management macro.

Section 1.4, Parameter Field Value Mnemonics, explains the lowercase abbreviations that appear in the parameter fields of timer management macros.

6.2 CANCEL TIME INTERVAL REQUEST (CANTIME)

The CANTIME macro cancels all previous interval requests that match the increment of time specified in OPT and the parameter located in PARM. (PARM is the parameter associated with the interval to be cancelled.) If the interval to be cancelled is part of a cyclic group, the entire time cycle is cancelled.

Format:

```
    [symbol]  CANTIME  [PARM=][,OPT=][,PCB=][,FORM=]
```

Parameter Values:

```
    PARM       =  absexp
               =  (reg)

    OPT        =  TOD (code to indicate time-of-day interval
                  to be cancelled)
               =  INT (code to indicate interval to be cancelled)

    PCB        =  addrx  (address or pointer to parameter
                  control block)
               =  (reg) - address or pointer to parameter
                  control block

    FORM       =  L (list form - only build PCB)
```

The possible condition codes are:

- A condition code of 0 indicates normal termination.
- A condition code of 4 indicates that no previous interval request exists that matches the provided parameter.

Example:

```
         CYCTIME  NUMBINT=3,TABLE=ALPHA,OPT=INT
           .
           .
           .
         CANTIME PARM=2,OPT=INT  CANCEL THE GROUP

    ALPHA TIMETBL (10,1),(20,2),(30,3),OPT=INT
```

6.3 SCHEDULE TRAPS CYCLICALLY AT DIFFERENT TIMES (CYCTIME)

The CYCTIME macro repetitively adds items to the calling task's queue at user-defined intervals within a specific time cycle until the task terminates or issues a CANTIME macro specifying any parameter in the table. The user-defined intervals within a specific time cycle must be specified as all TOD intervals or all INT intervals.

Format:

```
    [symbol]   CYCTIME   [NUMBINT=][,TABLE=][,OPT=][,PCB=]
                         [,FORM=]
```

Parameter Values:

NUMBINT	=	abs halfword exp (number of intervals defined in the table)
	=	(reg) - register containing number of intervals defined in the table
TABLE	=	addrx (address or pointer to the table of intervals)
	=	(reg) - address or pointer to the table of intervals
OPT	=	TOD (table of intervals in seconds from midnight)
	=	INT (table of intervals in milliseconds from now)
PCB	=	addrx (address or pointer to parameter control block)
	=	(reg) - address or pointer to parameter control block
FORM	=	L (list form - only build PCB)

Programming Considerations:

The table can be built with the TIMETBL macro. The option in the TIMETBL macro must agree with the option in the CYCTIME macro; that is, both must be TOD or both must be INT. Section 6.6 details the TIMETBL macro.

None of the intervals can be zero. If the intervals are specified as TOD, each interval must minimally be one greater than the preceding interval.

The time cycle, in which the user-defined intervals must occur, differs for time-of-day intervals and interval timing intervals. The cycle for time-of-day intervals ranges from the day on which the first interval occurs through and including the day on which the last interval occurs. The time cycle is the sum of days on which the intervals occur. The time period for interval timing is the sum of intervals in the table.

Example:

```
    TODTABLE  TIMETBL (15:00:00,1),(1:15:00:00,2),
                      (2:15:00:00,3),(2:16:00:00,4),OPT=TOD

         The intervals are:

         15:00 hours of current day
         15:00 hours of second day
         15:00 hours of third day
         16:00 hours of third day
         15:00 hours of fourth day
                    .
                    .
                    .
         The time cycle is three days.

    INTTABLE  TIMETBL  (:18,1),(:36,2),OPT=INT

         The first interval is 18 seconds or 18000 ms.
         The second interval is 36 seconds or 36000 ms.
         The third interval is 18 seconds or 18000 ms.
                    .
                    .
                    .
         The time cycle is 54 seconds or 54000 ms.
```

6.4 GENERATE A GENTIME INTERVAL (GENTIME)

The GENTIME macro converts a time expression to seconds from midnight or milliseconds from now; but, it does not convert a time expression to both. If INT is specified and OPT is omitted, bits 0 through 3 of the word generated are set to 0001. If INT is specified and OPT is not omitted, bits 0 through 3 are not set. The macro generates a CAL DC instruction at the symbol.

Format:

```
    [symbol]  GENTIME  [TOD=][,INT=][,OPT=]
```

Parameter Values:

TOD	=	time expression (specifies the time of day)
INT	=	time expression (specifies the time interval)
OPT	=	any character (to prevent bits 0 through 3 from being set)

Programming Considerations:

The time expression is expressed as:

```
    DAY:HOUR:MINUTE:SECOND.FRACTION-OF-SECOND
```

The expression is evaluated from right to left. If TOD is specified, the macro converts it to seconds from midnight and the fraction-of-second is ignored. If INT is specified, the time is converted to milliseconds. |

Example:

```
GENTIME      INT=.005           5MS
GENTIME      INT=.05           50MS
GENTIME      INT=.5           500MS
GENTIME      INT=1.5         1500MS
GENTIME      INT=2:0.5     120500MS
GENTIME      TOD=.5        MIDNIGHT
GENTIME      TOD=25        25 SECONDS AFTER MIDNIGHT
GENTIME      TOD=1:0:0     1 AM
GENTIME      TOD=60:       1 AM
GENTIME      TOD=2::       2 AM
GENTIME      TOD=120:      2 AM
GENTIME      TOD=3:::      3 DAYS FROM MIDNIGHT
```

6.5 READ TIME REMAINING FOR AN INTERVAL TO ELAPSE (READTIME)

The READTIME macro reads the current remaining time associated with the parameter in PARM. PARM is the parameter associated with the desired interval when the interval was started. The time is returned in 4 bytes past the beginning of the PCB that R14 points to.

Format:

```
    [symbol]  READTIME [PARM=][,OPT=][,PCB=][,FORM=]
```

Parameter Values:

PARM	=	absexp
	=	(reg)
OPT	=	TOD (time of day returned in seconds from midnight)
	=	INT (time returned in milliseconds from now)
PCB	=	addrx (address or pointer to parameter control block)
	=	(reg) - address or pointer to parameter control block
FORM	=	L (list form - only build PCB)

Programming Considerations:

If the interval was scheduled with a CYCTIME macro and more than one interval in the table had the same parameter associated with it, the current time during the desired interval cannot be the time that is read. Each interval should have a unique parameter associated with it. The condition code is set as:

- A condition code of 0 indicates normal termination.
- A condition code of 4 indicates that no interval is associated with PARM.

Example:

```
    TRAPTIME    INT=4,PARM=1       Four second interval
    STA         15,HOLD            Hold pointer to PCB
      .
      .
      .
    LDA         5,HOLD             Recover pointer to PCB
    READTIME    PCB=(5),PARM=1     Read the time
      .
      .
      .
```

Since the same parameter block used in the TRAPTIME macro reads the time, it is returned into the parameter block in the same form.

6.6 BUILD A TABLE OF TIME INTERVALS FOR CYCTIME (TIMETBL)

The TIMETBL macro builds a table of time intervals and associated parameters for use with the CYCTIME macro. Refer to the GENTIME macro, Section 6.4, for a definition of a time expression.

Format:

```
    [symbol]  TIMETBL  (interval,parm)...,OPT=
```

Parameter Values:

interval	-	time expression (the interval of time)
parm	-	absolute expression (the parameter associated with the interval)
OPT	=	TOD (interval is time of day)
	=	INT (interval in milliseconds)

6.7 SCHEDULE AN INTERRUPT BY ADDING A PARAMETER TO THE TASK QUEUE WHEN A SPECIFIED INTERVAL HAS ELAPSED (TRAPTIME)

The TRAPTIME macro concurrently sets up a timer interval with the task's subsequent execution. An item is then added to the calling task's task queue when the user-specified interval has elapsed.

Format:

```
[symbol]  TRAPTIME  [TOD=][,INT=][,PARM=][,PCB=]
                    [,FORM=]
```

Parameter Values:

TOD	=	time expression (time of day to schedule interrupt)
	=	(reg) - register containing the time of day in day in seconds from midnight to schedule interrupt
INT	=	time expression (interval of time to wait)
	=	(reg) - register containing interval in milliseconds
PARM	=	absexp (parameter that is added to the task queue)
	=	(reg) - parameter that is added to the task queue
PCB	=	addrx (address or pointer to parameter control block)
	=	(reg) - address or pointer to parameter control block
FORM	=	L (list form - only build PCB)

Programming Considerations:

Before executing the TRAPTIME macro, these steps must be performed so traps are serviced as they occur:

1. Define a trap-service routine.
2. Initialize the UDL with the SETUDL macro:

```
SETUDL TSKN=(TMCQ,addrx),TSKQ=addrx
```

3. Initialize the current TSW with the LTSW macro:

```
LTSW TSKE,TMCQ
```

After the interval has started and the condition code is set, the task can continue processing or enter a trap wait state. The possible condition codes are:

- A condition code of 0 indicates that the interval has started; normal termination.
- A condition code of 4 indicates a sufficient amount of system space is unavailable.

6.8 WAIT FOR A SPECIFIC INTERVAL TO ELAPSE (WAITTIME)

The WAITTIME macro places the task in a wait state until the specified time-of-day or interval has elapsed. Refer to the GENTIME macro, Section 6.4, for the definition of a time expression.

Format:

```
    [symbol]  WAITTIME  [TOD=][,INT=][,PCB=][,FORM=]
```

Parameter Values:

```
    TOD        =  time expression
               =  (reg)

    INT        =  time expression
               =  (reg)

    PCB        =  addrx
               =  (reg)

    FORM       =  L (list form - only build PCB)
```

Programming Considerations:

Time of day (TOD) is the time when a task, currently in a wait state, resumes execution. If TOD is specified, the time is converted to seconds from midnight and the fraction-of-second is ignored. If INT is specified, the time is converted to milliseconds from the time the WAITTIME macro was executed. PCB is the address or pointer to the parameter control block.

Examples:

```
    WAITTIME INT=5            is converted to 5000ms.

    WAITTIME INT=.005         is converted to 5ms.
```

CHAPTER 7
MODEL 3200MPS SYSTEM MACROS

7.1 INTRODUCTION

The Perkin-Elmer Multiprocessing System (Model 3200MPS System) consists of a central processing unit (CPU), up to nine auxiliary processing units (APUs), and a number of logical processing units (LPUs) used to assign (map) tasks to the CPU or an APU.

Tasks can run on an APU or on the CPU and must be able to communicate with, obtain status information about, and have control over other tasks in the Model 3200MPS System. Communications between and control of tasks running on the CPU or the APU are performed by supervisor call (SVC) 6 and the new SVC 13. The Model 3200MPS System macros detailed in this chapter make the SVC services available to a task and ensure the proper timing and use of these services. These macros are applicable to the Model 3200MPS System only. They are divided into the following categories:

- Support macros. Included in this category are the APPCB, APPERTBL, APPERR, APPERRET, and APSTRUC macros.
- Information macros. In this group are the FETLPU and APUSTAT macros.
- A macro used to map the LPU to the APU. This is the APUMAP macro.
- Task control macros that include the APUCNTL and the REQUEUE macros.
- Task direction macros. This category consists of the SETCPU and SETLPU macros.
- Task timer macros that include the CRTIMERS, RESETIME, STARTIME, GETIME, READTCNT, and STOPTIME macros.

7.1.1 Chapter Organization

This chapter explains the functions of the Model 3200MPS System macros and details their format, parameters, and functional details. Examples and explanations for using each of the macros are included.

Each macro is presented in an easily-readable format that in all cases shows the user-supplied name or symbol (in lower case) in the NAME field, the operation in the OPERATION field, and the parameters in the OPERAND field. When a parameter is enclosed in brackets, it is optional. An operand entered without brackets is a required operand. The operands are separated by commas and can continue on more than one line.

Example:

NAME	OPERATION	OPERAND
symbol	APUSTAT	APN=n [,BUF=n] [,LEN=n]

In this example, symbol designates a user-supplied name and APUSTAT is the macro name. The APN parameter is required. The BUF and LEN parameters are optional, shown by their enclosing brackets. Each parameter except the first one must be entered with its preceding comma. The value of the character to be entered after the keyword equal sign (,BUF=n) is explained under the Parameters: heading in each macro section. The keyword parameters are nonpositional.

7.2 SUPPORT MACROS

The Model 3200MPS System support macros build a parameter block for use by the other macros; generate tables of branch addresses to user-written error routines; handle input/output (I/O) errors; enable a task to return to the instruction following a macro that caused an error; generate subroutines that check error status after completing an I/O function; and generate structures and define equates for the appcb parameter block built by the APPCB macro or built automatically by other macro calls.

Support macros are called automatically from the information, mapping, and control macros, or the user can directly call the support macros and, thereby, override the timing and default values set by the other macros. The support macros are:

- APPCB
- APPERTBL
- APPERR
- APPERRET
- APSTRUC

7.2.1 APPCB (Build APU Parameter Block) Macro

The APPCB macro builds the appcb parameter block for use by the APU information, mapping, and task control macros and fills the parameter block fields with the appropriate information. This macro also can be called automatically from the FETLPU, APUSTAT, APUMAP, APUCNTL, and REQUEUE macros.

Format:

```
     NAME     |   OPERATION   |              OPERAND
-----------------------------------------------------------------
    symbol    |   APPCB       | [FUN=] [,OPT=n] [,DOPT=n]
              |               |        [,APN=n] [,ERR=n] [,BUF=n]
              |               |        [,LEN=n] [,USE=n]
```

Parameters:

FUN=	This optional parameter sets up the function field in the appcb parameter block. A user can enter any of the following four function field designators with this parameter:

1. LPMT.FUN (entered as FUN=LPMT.FUN) returns the logical processor mapping table (LPMT) in the user buffer. The LPMT lists which LPUs are mapped to which APUs.
2. STAT.FUN returns APU task status information.
3. MAP.FUN performs the LPU/APU mapping functions.
4. CTRL.FUN performs the APU control functions.

OPT=	n specifies the mapping or control function to be performed. The available options are any byte expression.

Example:

```
OPT=X'FF'
```

DOPT=	n specifies the command to be issued or the LPU number to be mapped to.
APN=	n specifies the number of the processor for which status information is being requested. The range is from 0 through 9, with 0 always designating the CPU.
HSTAT=	n specifies in bytes the data to be sent to the APU for a link check in one's complement of the APU status byte.
ERR=	n specifies the return error code.
BUF=	n specifies the address of the user buffer.
LEN=	n specifies in bytes the maximum length of the user buffer.
USE=	n specifies the length in bytes of the buffer actually used.

Functional Details:

If the APSTRUC macro (Section 7.2.5.) that generates structures and defines equates for the appcb parameter block was not expanded by the time the APPCB macro is called, the APPCB macro expands the APSTRUC macro and uses the symbolic names generated for the parameter block.

If the paramater block is generated by the macro using it, the parameter block will be pointed to by the register defined by the ENVIRON macro (See Section 8.4) as the PCBREG parameter immediately following the macro execution. The default value for this register is the contents of Register 14 (R14).

If the APPCB macro is expanded in a pure section of code, the macro allocates the required storage space at the next impure location that is fullword aligned, constructs the parameter block, and returns to the pure segment.

Example:

In this example, the APPCB macro builds a parameter block to obtain the status for APU1 and uses the storage space called BUFFER. The storage space (BUFFER) is 128 bytes long.

```
    PARMBLK APPCB APN=1, BUF=BUFFER, LEN=128
```

7.2.2 APPERTBL (Build APU Error Recovery Table) Macro

The APPERTBL macro builds a table of branch addresses to user-written error recovery routines that handle specific errors returned by the APU task control macros.

The value the user gives each parameter with this macro specifies the branch address of the error recovery subroutine written by the user.

Format:

```
  NAME   |  OPERATION   |                OPERAND
---------------------------------------------------------------
symbol   |   APPERTBL   | [,BAE=n] [,BNW=n] [,IBS=n]
                        | [,ITO=n] [,PNG=n] [,IAN=n]
                        | [,ILN=n] [,IOS=n] [,COP=n]
                        | [,ONX=n] [,DIS=n] [,NTS=n]
                        | [,NMR=n] [,NDS=n] [,ENE=n]
                        | [,NOF=n] [,NET=n] [,EIX=n]
                        | [,NTQ=n] [,ELSE=n]
```

Parameters:

BAE=	n specifies the address of a routine to remedy errors if the data buffer is not fullword aligned. This and all of the following parameters are optional with the APPERTBL macro. If none of these parameters are entered, the default error routine address table (@ERR.TAB) is built.
BNW=	n specifies the address of a routine to remedy errors if the data buffer is not located in a writable segment.
IBS=	n specifies the address of a routine to remedy errors if the data buffer is not long enough (Insufficient Buffer Space).
ITO=	n specifies the address of a routine to remedy errors if task options prevent granting of privileges (Illegal Task Options).
PNG=	n specifies the address of a routine to remedy errors if the recovery address for Privileges was Not Granted.

IAN=	n specifies the address of a routine to remedy errors if an Invalid APU Number was specified.
ILN=	n specifies the address of a routine to remedy errors if an Invalid LPU Number was specified.
IOS=	n specifies the address of a routine to remedy errors if an Invalid Option byte was Specified.
COP=	n specifies the address of a routine to remedy errors if requested privileges are currently owned by another task (Currently Owned Privileges).
ONX=	n specifies the address of a routine to remedy error if the APU cannot be marked On-EXclusive from the On state.
DIS=	n specifies the address of a routine to remedy errors if a request was denied because the APU was Disabled.
NTS=	n specifies the address of a routine to remedy errors if status cannot be returned because the queue is locked.
NMR=	n specifies the address of a routine to remedy errors if No Mapping Rights were granted.
NDS=	n specifies the address of a routine to remedy errors if the APU is Not in the Disabled State.
ENE=	n specifies the address of a routine to remedy Errors if the the APU is Not Enabled.
NOF=	n specifies the address of a routine to remedy errors if the APU is Not marked Off.
NET=	n specifies the address of a routine to remedy errors if the exclusive task is not in the system, or if the APU is not marked On.
EIX=	n specifies the address of a routine to remedy errors encountered during command transmission.
NTQ=	n specifies the address of a routine to remedy errors if the preemptive task was not on the ready queue.
ELSE=	n specifies the address of a routine to remedy any errors other than those specified above.

Functional Details:

The APPERTBL macro is automatically expanded by the APPERR macro. See Section 7.2.3. The user can call the APPERTBL macro prior to calling the APPERR macro, and the APPERR macro will not expand the APPERTBL macro. Once the APPERTBL macro is expanded, any macro needing the error recovery routine address table will refer to it. To use a user-defined error routine address table, the APPERTBL macro must be expanded by the user before it is expanded by the first occurrance of the APPERR macro. The name of the default error routine address table is @ERR.TAB.

The ELSE keyword parameter provides a way for the user to handle all errors except for the ones whose entry points were identified by the standard keyword parameters. If none of the keyword parameters (including ELSE) is specified, the default error recovery subroutine is executed.

The user can specify the error conditions with user-written recovery procedures by including the keyword for the specific error followed by the error recovery entry point. The macro starts execution at these points when the error occurs. Whenever the ELSE parameter is specified, the ELSE handler must be able to handle all error conditions except those specifically referred to by the keyword parameters used.

Example:

This is an example of a user-written routine to handle four different types of errors: the condition of the APU ready queue is locked (NTS); the status buffer is not fullword aligned (BAE); the status buffer is not long enough (IBS); the requested rights are currently owned by another task (COP).

```
    TABLENAME APPERTBL NTS=RETRY1MS,BAE=ALIGNBUF,IBS=SHORTBUF
                       ,COP=RETRY1MS
```

7.2.3 APPERR (APU Error Recovery) Macro

The APPERR macro is an error linkage macro that references branch addresses in the error table built by the APPERTBL macro. The APPERR macro generates default error recovery and error linkage procedures to the user-written subroutine if that subroutine was specified.

Format:

NAME	OPERATION	OPERAND
symbol	APPERR	

Parameters:

There are no optional or required parameters for this macro.

Functional Details:

This macro is expanded automatically by the first call to the FETLPU, APUSTAT, APUMAP, APUCNTL, or REQUEUE macro. After the first expansion of this macro, all subsequent macro calls generate a linkage to the subroutine.

If the APPERTBL macro was not expanded by the user, it is expanded using the default values. The default error recovery procedure used by the system will release control or mapping rights to an APU after the function completes. The RELEASE=N parameter with the APUMAP and APUCNTR macros has no effect on the default error recovery procedure.

The default error procedure for any error is to display an error message. The task is paused when an error is encountered. To restore all of the user registers and continue executing the user code at the line following the macro call producing the error, enter the operator CONTINUE command.

Table 7-1 shows the error messages produced by the default System Error Recovery Subroutine and the conditions causing the errors.

TABLE 7-1 ERROR RECOVERY SUBROUTINE DESCRIPTIONS

ERROR	REASONS FOR ERROR
BAE	Data buffer not aligned on fullword boundary.
BNW	Data buffer not located in writable segment.
IBS	Insufficient space in data buffer.
ITO	Link option prohibits granting requested privilege.
PNG	Task was not granted the requested privilege.
IAN	APU number greater than the maximum allowed.
ILN	LPU number greater than the maximum allowed.
IOS	An invalid option specified for the function.
COP	Requested option was specified for this function.
ONX	APU cannot be marked On-Exclusive from the On state.
DIS	Function rejected, APU is in disabled state.
NTS	Access to APU ready queue not obtainable.
NMR	No mapping rights for LPU currently mapped to the APU.
NDS	Cannot enable the APU more than once.
ENE	APU could not pass the power up link check.
NOF	Disable the APU from the Off state only.
NET	Exclusive task is not in the system. APU is not on.
EIX	Error encountered in transmission of command.
NTQ	The preemptive task was not on the ready queue.
ELSE	All other errors except those specified.

Examples:

Since there are no parameters that must be entered with this macro, simply entering the macro name causes the error linkage function to occur:

```
    APPERR
```

7.2.4 APPERRET (APU Error Return) Macro

The APPERRET macro is the return error linkage macro that permits the user to recover from an error and return to the user-level code and continue the main program at the instruction following the APU macro that caused the error.

Format:

NAME	OPERATION	OPERAND
symbol	APPERRET	

Parameters:

There are no required or optional parameters for this macro.

Functional Details:

While in the user-defined error recovery procedure, a user can use any of the general purpose registers without affecting the normal execution of the main program, with the exception of the link register. The link register is defined by the SCRREG parameter of the ENVIRON macro. See Section 8.4. The default value is R15.

NOTE

The return register points back to the error recovery subroutine, NOT back to the instruction following the macro.

Examples:

This is an example of a user-written routine to recover from a buffer alignment error. The algorithm used is to round the buffer address up to the next fullword location and decrease the length of the buffer by the alignment factor. Assume the buffer address to be in location STATBUF, and the buffer length to be in LENGTH.

```
ALIGNBUF EQU    *
     L      R1,STATBUF       GET ADDRESS OF STATUS BUFF
     LR     R2,R1            SAVE THE ADDRESS
     AIS    R1,3             ROUND UP TO NEXT FULLWORD
     NHI    R1,X'FFFC'
     ST     R1,STATBUF       SAVE THE NEW BUFFER ADDRESS
     NHI    R2,3             SAVE THE ODD BITS
     SIS    R2,4             SUBTRACT ALIGNMENT FACTOR
     L      R1,LENGTH        GET THE BUFFER LENGTH
     AR     R1,R2            ADD NEGATIVE VALUE TO
                             LENGTH
     ST     R1,LENGTH        SAVE THE SHORTER VALUE
     APPERRET                RETURN TO USER TO TRY
                             AGAIN
```

This is an example of a user-written routine to ignore an error, which is 'status buffer too short in length for the data to be returned'.

```
SHORTBUF EQU    *
         LIS    R0,0            SET CONDITION CODE TO ZERO
         APPERRET               RETURN TO CAL ERROR
```

7.2.5 APSTRUC (Control and Mapping Structures) Macro

The APSTRUC macro generates structures and defines equates for the appcb parameter block used by the FETLPU, APUSTAT, APUMAP and APUCNTL macros.

Format:

NAME	OPERATION	OPERAND
symbol	APSTRUC	

Parameters:

There are no required or optional parameters for this macro.

Functional Details:

This macro is automatically invoked by the first APPCB macro call. Once this macro is expanded, it will not be expanded again.

Examples:

These are examples of the structures that are defined by this macro:

```
SVC13.STRUC                 SVC 13 PARAMETER BLOCK

SV13.OPT DS     1           OPTIONS
SV13.FUN DS     1           FUNCTION CODE
SV13.DOP DS     1           DIRECTIVE OPTION
SV13.APN DS     1           APU NUMBER
SV13.APS DS     2           APU STATUS
SV13.ERR DS     2           ERROR STATUS
SV13.BUF DS     4           DATA BUFFER START ADDRESS
SV13.USE DS     2           BUFFER USED
SV13.LEN DS     2           LENGTH OF BUFFER

 THE SVC13 FUNCTION 1 BUFFER DEFINITION STRUCTURE

F1B.       STRUC            STRUCTURE OF  FUNCTION 1 BUFFER
```

```
F1B.APUN DS     1          1-BYTE FOR  APU NUMBER
F1B.LPUN DS     1          1-BYTE FOR  LPU NUMBER
F1B.NTSK DS     2          2 BYTES FOR NUMBER OF READ TASKS
F1B.STAT DS     2          2 BYTES APU STATUS
F1B.OPTS DS     2          2 BYTES OF OPTIONS
F1B.CURT DS     8          NAME OF CURRENTLY ACTIVE TASK
F1B.CNTL DS     8          NAME OF TASK WITH CONTROL RIGHTS
F1B.MAPT DS     8          NAME OF TASK WITH MAPPING RIGHTS
F1B.RDYQUEUE DS     8      FIRST TASK NAME ON READY QUEUE

        APU STATE MNEMONICS DEFINITION

APS.DISA EQU    0          APU IS DISABLED
APS.MOFF EQU    1          APU IS MARKED OFF
APS.MONX EQU    2          APU IS MARKED ON EXCLUSIVE
APS.MON  EQU    3          APU IS MARKED ON NORMALLY
APS.WATX EQU    6          APU WAITING FOR EXCLUSIVE TASK
APS.WAIT EQU    7          APU IS WAITING FOR NORMAL TASK

        APU MAPPING MNEMONICS DEFINITION

MAP.GMP  EQU    X'80'      GAIN MAPPING PRIVILEGES
MAP.MOX  EQU    X'40'      MARK APU ON, EXCLUSIVE
MAP.MON  EQU    X'20'      MARK APU ON
MAP.MAP  EQU    X'10'      MAP APU INTO LPMT AT LPU"X"
MAP.REM  EQU    X'08'      REMOPVE ALL REFERENCES TO APU
MAP.MOF  EQU    X'02'      MARK APU OFF
MAP.RMP  EQU    X'01'      RELEASE MAPPING PRIVILEGES

        APU CONTROL OPTIONS MNEMONICS DEFINITION

COPT.GCR EQU    X'80'      GAIN APU CONTROL RIGHTS
COPT.ENA EQU    X'40'      ENABLE APU
COPT.ECF EQU    X'08'      EXECUTE CONTROL FUNCTION
COPT.DIS EQU    X'02'      DISABLE MARKED OFF APU
COPT.RCR EQU    X'01'      RELEASE CONTROL RIGHTS

        APU CONTROL COMMAND MNEMONICS

CCMD.STR EQU    X'01'      START THE APU
CCMD.SST EQU    X'02'      SINGLE STEP THE APU
CCMD.TTH EQU    X'04'      TRANSFER TASK TO CPU
CCMD.RPF EQU    X'07'      RELOAD POWER FAIL IMAGE
CCMD.SPF EQU    X'08'      STORE POWER FAIL IMAGE
CCMD.RTS EQU    X'80'      RTSM CHECK
CCMD.RES EQU    X'83'      RESCHEDULE ON APU
CCMD.STO EQU    X'85'      STOP THE APU
CCMD.STA EQU    X'86'      SEND APU STATUS
CCMD.CHK EQU    X'8A'      CHECK POINT THE CURRENT TASK

         SVC 13 FUNCTION CODES

LPMT.FUN EQU    X'00'
STAT.FUN EQU    X'01'
MAP.FUN  EQU    X'02'
CTRL.FUN EQU    X'03'
```

7.3 INFORMATION MACROS

Tasks in a Model 3200MPS System must be able to obtain status information about the LPUs and APUs in the system. This information is available to the tasks through SVC 13. By use of the APU information macros that utilize the SVC services, LPU and APU information is available to all tasks in the system. The information macros are FETLPU and APUSTAT. They allow a task to gain information regarding the:

- Maximum number of LPUs and APUs
- Logical Processor Mapping Table (LPMT)
- Number of LPUs mapped for the selected APU
- Number of tasks in the ready queue for the selected APU
- Status of the selected APU
- Task names associated with the selected APU, which include:
    - the active, control, and mapping tasks
    - what task the selected APU is waiting for
    - APU number
    - all ready tasks in queue order
    - task options

7.3.1 FETLPU (Fetch LPU Map) Macro

The FETLPU macro returns APU and LPU information to the requesting task and stores the information in a user-specified buffer. This information consists of:

- Maximum number of APUs and LPUs
- A copy of the Logical Processor Mapping Table (LPMT)

The LPMT contains one entry for each LPU number, starting at LPU0 and ending with LPUn, where n represents the maximum number of LPUs that can be configured in the system at system generation. See the System Generation/32 (SYSGEN/32) Reference Manual. By convention, LPU0 always represents the CPU

Format:

NAME	OPERATION	OPERAND
symbol	FETLPU	[appcb] [,BUF=n] [,LEN=n]

Parameters:

appcb	specifies the address or pointer to the appcb parameter block or to a register that will contain the address of the parameter block.
BUF=	n specifies the starting address of the user-specified buffer to receive the requested data. The buffer length can be variable, but must begin on a fullword boundry and be located in the task's writable segment. This parameter can be supplied in the FETLPU macro or in the APPCB macro if the parameter block is expanded separately. The buffer address can also be passed to the FETLPU macro in a register in the format BUF=(R4). A data buffer is not required for mapping or control macros.

LEN=	n specifies a decimal number that expresses the maximum usable length of the data buffer in bytes. This parameter can be supplied in the FETLPU macro or in the APPCB macro if the parameter block is expanded separately. The length can also be passed to the FETLPU macro in a register in the format LEN=(R1). The maximum length is 65,535 bytes, and the minimum is 4.

NOTE

The maximum required buffer length can be calculated as follows:

LEN = ((#LPUs + 1) * TABLEWIDTH) + 4

#LPUs = The maximum number of LPUs in the system. TABLEWIDTH always equals 1.

The maximum buffer length ever required is:

LEN = ((255 + 1) * 1) + 4 = 260 bytes

When this call is successfully completed, the following parameter block fields are defined:

- SV13.ERR - This error status field can contain any of the following values:
  - Zero means no error, successful completion.
  - BAE means a buffer alignment error, because the starting address of the data buffer is not fullword aligned.
  - BNW means a buffer not writable error because the data buffer is in a nonwritable program segment.
  - IBS means insufficient buffer space error because the data buffer is not large enough to contain all the data.

The data buffer field shows the data that was returned to the buffer in bytes in the format shown in Figure 7-1.

0(0) (Reserved)	1(1) Maximum APU Number	2(2) Maximum LPU Number	3(3) LPMT Table Width
4(4) Copy of Logical Processor Mapping Table (LPMT) . . .			

Figure 7-1 Buffer Data Returned for FETLPU Macro

Maximum APU Number	This field contains a binary number with a value that can range from 0 through 9, representing the total number of APUs that can be in the current system configuration.
Maximum LPU Number	This field contains a binary number with a value that can range from 1 to 255, representing the total number of LPUs that can be in the current system configuration.
LPMT Table Width	This field is for future expansion. It now always has a value of one.
LPU1 through LPUn	These fields contain the APU number that the LPU is mapped to. The value can range from 0 through 9, the maximum APU number. A value of zero always signifies that the LPU is mapped to the CPU. The LPU number is determined by the byte position in the data buffer and can be calculated by the following: LPU# = Byte Position - 4 The first byte in the buffer starts at zero.

Functional Details:

If the appcb parameter is not entered with this macro, the appcb parameter block is automatically built and the required parameters are assigned to the appropriate fields.

If any required parameter is not specified in the FETLPU macro, it must have been previously specified in the APPCB macro. Any required parameter specified with an existing appcb parameter block replaces the old value in that field.

The buffer address (BUF) must begin on a fullword boundary and be located in the task's writable segment.

Examples:

In the following example, the user wants to access the maximum LPU number, the maximum APU number, and the LPMT.

```
        ----
        ----
        ----
        FETLPU   , BUF=BUFFER,LEN=260
        LB         R1,BUFFER+1            R1=Maximum APU Number
        LB         R2,BUFFER+2            R2=Maximum LPU Number
        LA         R3,BUFFER+4            R3=Address of LPMT
        ----
        ----
        ----
        ALIGN 4
 BUFFER DS     260
```

NOTE

If an error occurs, it is handled by the default error recovery procedure, the APPERR macro.

In this example, the user wants to use a named parameter block for this FETLPU macro call.

```
PROGRAM
          ----
          ----
          ----
          FETLPU     PARMBLK
          LB         R1,BUFFER+1           R1=Maximum APU Number
          ----
          ----
          ----
 PARMBLK  APPCB      FUN=MAP.FUN,BUF=BUFFER,LEN=260
          ALIGN      4
 BUFFER   DS         260
```

7.3.2 APUSTAT (Fetch APU Status) Macro

The APUSTAT macro allows a task to access APU status information by returning information for the specified APU to the requesting task. The returned status information is stored in the user-specified buffer and consists of the:

- APU number, state, and options
- Number of tasks in the APU ready queue
- Number of LPUs mapped to the APU
- Name of the task having control and mapping rights of the APU
- Name of the task currently active on the APU
- Name of the task the APU is waiting for
- List of tasks in the APU ready queue in order of execution

Format:

NAME	OPERATION	OPERAND
symbol	APUSTAT	[appcb] [,APN=n] [,BUF=n] [,LEN=n]

Parameters:

appcb	Specifies the address or pointer to the appcb parameter block or to a register that will contain the address of the parameter block.
APN=	n specifies the APU number about which information is desired. The number can be supplied in this macro or in the APPCB macro if the parameter block is built separately. The APU number can also be passed to the APUSTAT macro in bits 24 through 31 of a register. The format for this is APN=(R4).

BUF=	n specifies the starting address of the user-specified data buffer or register. This buffer can be variable in length, but must begin on a fullword boundary and be located in the task's writable segment. This parameter can be supplied in the APUSTAT macro or in the APPCB macro if the parameter block is expaned separately. The buffer address can also be passed to the APUSTAT macro in a register in the format BUF=(R2). A data buffer is not required for mapping or control macros. This is a required parameter.
LEN=	n specifies a decimal number that expresses the maximum usable length of the data buffer in bytes. This parameter can be supplied in the APUSTAT macro or in the APPCB macro if the parameter block is expanded separately. The length can also be passed to the APUSTAT macro in a register in the format LEN=(R9). The maximum length is 65,535 bytes and the minimum is 8. This is a required parameter.

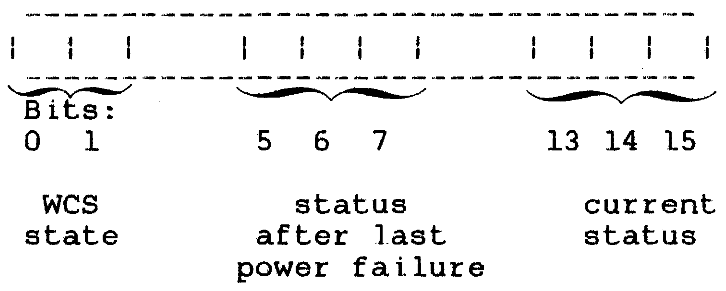
The data buffer field receives the status information in the format shown in Figure 7-2.

```
 --------------------------------------------------------
|0(0)            |1(1)            | 2(2)                 |
|  APU number    |  Number of     |    Number of tasks   |
|                |  LPUs mapped   |                      |
|--------------------------------------------------------|
|4(4)                       |                            |
|       APU.STAT            |         APU.OPTS           |
|--------------------------------------------------------|
|8(8)                                                    |
|                  Active task name:                     |
|-----------             or                 ------------|
|12(C)             Waiting task name                     |
|                                                        |
|--------------------------------------------------------|
|16(10)                                                  |
|                                                        |
|------------     Control task name         ------------|
|20(14)                                                  |
|                                                        |
|--------------------------------------------------------|
|24(18)                                                  |
|                                                        |
|------------     Mapping task name         ------------|
|28(c)                                                   |
|                                                        |
|--------------------------------------------------------|
|32(20)                                                  |
|                 Ready task names (1)                   |
|------------           or                  ------------|
|                 Exclusive task name                    |
|                                                        |
|--------------------------------------------------------|
|                         .                              |
|                         .                              |
|                         .                              |
|--------------------------------------------------------|
|                 Ready  task  names (N)                 |
|                         .                              |
|-------------            .                 ------------|
|                         .                              |
|                                                        |
 --------------------------------------------------------
```

Figure 7-2 Data Buffer Format for APUSTAT Macro

Fields:

APU number	is a 1-byte field containing the decimal number of the APU to which the status information applies. The number can be from 0 through 9, with 0 designating the CPU.
Number of LPUs mapped	is a 1-byte field containing the number of LPMT entries that are mapped to the specified APU.
Number of tasks	is a 2-byte field containing the number of waiting tasks in the specified APU ready queue. The active task, if there is one, is not included in this count.
APU.STAT	is a 2-byte field containing bit settings that indicate the current software status of the APU, the status after the last power fail, and whether or not the writable control store (WCS) is initialized or loaded for the specified APU. The specific bits used are identified below.



If bit 0 is set, the WCS has been initialized. If bit 1 is set, the WCS has been loaded.

The current status and status after last power failure may contain one of the following settings:

7 = APU on, and waiting for task
6 = APU on exclusive, and waiting for task
3 = APU on
2 = APU on, exclusive
1 = APU off
0 = APU disabled

APU.OPTS	is a 16-bit field specifying any special APU configuration options. If bit 0 of the halfword is set, the APU has no WCS support. If bit 1 is set, the APU has no floating point support. If bit 2 is set, the APU will stop and wait for a task; i.e., trap block wait convention. Other bits are reserved for future options.
Active task name	is an 8-byte field containing the name of the currently active task. It may also contain the waiting task name if the APU is stopped and waiting for a task. The task name is left-justified in the field and padded with blanks, if necessary. If there is no currently active or waiting task, the field is entirely filled with blanks.
Control task name	is an 8-byte field containing the name of the task that has been granted control rights over this APU. If no control task exists, the field is entirely filled with blanks.
Mapping task name	is an 8-byte field containing the name of the task that has been granted mapping rights over this APU. If no mapping task exists, the field is entirely filled with blanks.
Ready task names	is a variable length table of 8-byte fields containing the name of each task in this APU ready queue. The currently active task, if any, does not appear in this table. The number of entries in this table is given in the number of tasks field. The order of entries corresponds to the order of the tasks on the ready queue. If the current status setting for the APU is set to X'2' marked On-Exclusive the APU ready queue is always empty. The ready task names field contains the name of the task that has exclusive rights to the APU.

Examples:

This is an example of a user-written routine to get APU status. The routine could be part of the main body of code or a subroutine. Assume that the data buffer address is in location STATBUF; the buffer length is in location LENGTH; and the APU number is passed in R1:

```
    GETSTAT  EQU     *
             LIS    R3,2         INITIALIZE RETRY COUNTER
             ST     R3,RETRYCNT
             L      R2,STATBUF   GET THE BUFFER ADDRESS
             L      R3,LENGTH    GET THE BUFFER LENGTH
    RETRY    APUSTAT APN=1,BUF=(R2),LEN=(R3)
             BP      RETRY       SOFT ERROR TRY AGAIN
             BZ     CONTINUE     GOOD RETURN
```

If the condition code came back as a minus, there was an unrecoverable error and the main routine must handle this situation.

```
             WTO     'UNRECOVERABLE ERROR TRYING TO
                      GET APU STATUS'
             EOT      RC=1
```

This is the logic path for no errors:

```
    CONTINUE EQU     *
             EOT     RC=0
             END
```

In this example, the user requests the APU status for APU 1:

```
         ----
         ----
         APUSTAT     ,APN=1,BUF=BUFFER,LEN=112
         LHL          R0,SV13.APS(R14)        R0=APU H/S Status
         LA           R1,BUFFER               R1=Data Buffer Address
         LB           R2,F1B.APUN(R1)         R2=APU Number
         LB           R3,F1B.LPUN(R1)         R3= # LPUs Mapped
         LH           R4,F1B.NTSK(R1)         R4= # of Tasks
         LH           R5,F1B.STAT(R1)         R5=APU Mapping State
         LH           R6,F1B.OPTS(R1)         R6=APU Options
         LA           R7,F1B.CURT(R1)         R7=Current Task Address
         LA           R8,F1B.CNTL(R1)         R8=Controlling Task
                                                 Address
         LA           R9,F1B.MAPT(R1)         R9=Mapping Task Address
         LA           RA,F1B.RDYQ(R1)         RA=Address of Task on
                                                 the Ready Queue
         ----
         ----
         ----
         ALIGN        4
 BUFFER  DS           112
```

7.4 MAP LPU TO APU MACROS

Tasks in a Model 3200MPS System can request the mapping privileges of a specified APU through the APUMAP macro. If no other task was granted that privilege, and if permitted by the task's established options, mapping privileges are granted to the requesting task, giving the task the right to:

- Mark the APU on or off
- Map the APU into the LPMT
- Remove all references to the APU from the LPMT
- Request and release mapping privileges

Via the LPMT, an LPU can be mapped to one, and only one, APU. However, an APU can be mapped to more than one LPU.

Figure 7-3 shows the valid paths for the SVC call.

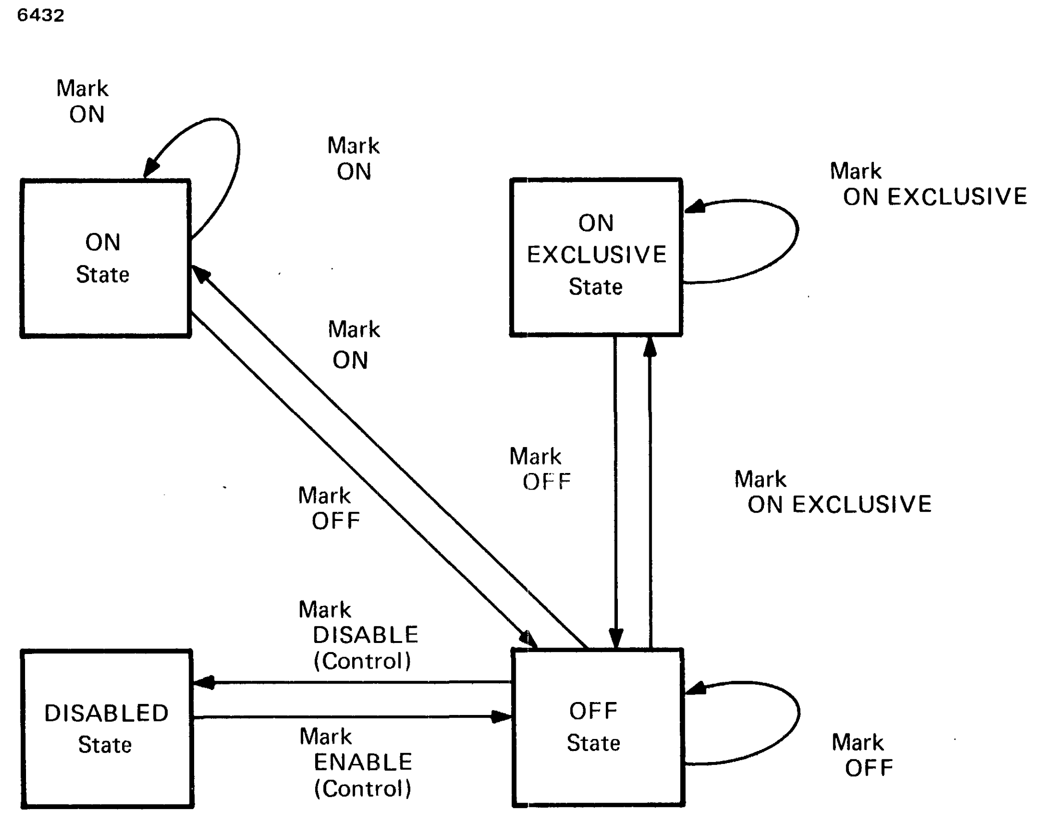


Figure 7-3 Valid Paths for an SVC Call

7.4.1 APUMAP (APU Mapping) Macro

The APUMAP macro is used by a task to gain mapping privileges to and perform mapping functions on the specified APU, and to release or not release the mapping privileges of a specified APU after completion of this macro function.

Format:

NAME	OPERATION	OPERAND
symbol	APUMAP	[appcb] [,APN=n] [,LPN=n]
		[,MAPFN= [mapfn1,...,mapfnn]]
		[,RELEASE=(Y/N)] [,TASKID=]

Parameters:

appcb	is the address or pointer to the appcb parameter control block (PCB). If this parameter is omitted, the parameter block will be built automatically.
APN=	n specifies the number of the APU that the mapping request is directed to or specifies a register. This is a required parameter if MLPU is specified in the MAPFN parameter.
LPN=	n specifies the LPU number or a register. If MLPU is specified, this parameter is required.
MAPFN=	name specifies the name of the APU for which mapping privileges are requested. A user can choose any one of the following four designators with this parameter:

1. APUON indicates that the specified APU is to be marked On.

2. MLPU means map the specified APU into the LPMT, which will contain the mapping arrangement between the LPU and the APU, at the LPU number specified with the LPN parameter. The LPMT contains one entry for each LPU number.

3. APUOFF means mark the specified APU off.

4. REMAPU means remove all references to the APU from the LPMT.

5. APUEXCL means mark the specified APU on exclusively to the task whose name must be specified with the TASKID parameter.

RELEASE=	Y/N indicates yes or no. Yes means the task is to release mapping privileges after completion of this macro function. No indicates mapping privileges are not to be released after completion of this macro function. Yes is the default parameter.
TASKID=	specifies the address of the buffer containing the task name to which this APU is to be marked on exclusively. It must be fullword aligned and 8 bytes long. The task name must be left justified, and any remaining bytes (if task name is less than 8 characters) must be filled with blanks.

NOTE

A buffer filled with all blanks means mark the APU on exclusively to the task issuing the macro.

Upon completion of the APUMAP macro, the following parameter block fields are defined:

- SV13.ERR - error status

byte 1	contains the bit position of the option being executed when the error occurred.
byte 2	contains the error code indicating the type of error. It will contain one of the following:

- Zero means no errors.
- BAE means the buffer is not fullword aligned.
- IBS means the buffer size is too small.
- ITO means an error, task options prevent granting of privilege.

- PNG means the task has not been granted privileges required to perform the option.
- IAN means an invalid APU number.
- ILN means an invalid LPU number.
- IOS means an invalid option specified.
- COP means an error, privilege is currently owned by another task.
- ONX means the APU cannot be marked On-exclusive from the On state.
- DIS means the APU is not enabled (disable state).
- NTS means the APU queue access is denied.
- NMR means the task has not been granted mapping rights over the APU that is currently mapped to the specified LPU.
- NET means APU could not be marked On-Exclusive because the exclusive task could not be found in the system.

Functional Details:

If appcb is not specified, the parameter block is automatically built and the required information is assigned to the appropriate fields. If appcb is specified, all the required information not specified with the macro is assumed to be in the parameter block.

To successfully execute this macro the task must be established with the link APU mapping option. See the OS/32 Link Reference Manual.

If more than one mapping function is specified, each must be enclosed in parentheses and separated by commas.

The value of N for the RELEASE parameter, will allow a task to retain mapping privileges to the specified APU after the macro call has completed. All other tasks in the system will be prohibited from sending mapping commands to that APU, including the command processor, until the mapping privileges are released by the task holding them. The controlling task releases mapping privileges by reentering the APUMAP macro and specifying Yes with the RELEASE parameter.

Examples:

In this example, the APUMAP macro maps all tasks directed to LPU1 to execute on APU1:

```
    APUMAP ,APN=1 ,LPN=1 ,MAPFN=MLPU
```

This example shows how the APUMAP macro marks APU2 into the OFF state:

```
    APUMAP ,APN=2 ,MAPFN=APUOFF
```

7.5 TASK CONTROL MACROS

A task in a Model 3200MPS System must be able to control other tasks in the system. A task can request the control privilege of a specified APU. If no other task was granted that privilege, and if permitted by the task's options, control privileges are granted to the requesting task giving the task the right to:

- Initialize an APU if it is waiting for power up link check
- Preempt current task
- Start normal APU execution (if stopped)
- Otherwise control the APU
- Disable APU
- Perform a power up link check on the APU
- Stop APU execution

7.5.1 APUCNTL (APU Control) Macro

The APUCNTL macro is used by a task to gain control privileges to and perform control functions upon the specified APU, and to release or not release control privileges to the specified APU.

Format:

```
  NAME    | OPERATION   |          OPERAND
--------------------------------------------------------
symbol    |  APUCNTL    | [appcb] [,APN=n] [,CNTRFN=n]
          |             | [,DATA=n] [,RELEASE=Y/N]
```

Parameters:

appcb	specifies the address or pointer to the parameter block or a register.
APN=	n specifies the number of the APU for which the control request is made, or it specifies a register containing the APU parameter.
CNTRFN=	n specifies the code command to be sent to the APU. A user can choose any one of the following seven control codes:

1. from idle state, STRTAPU means start normal APU execution.
2. from idle state, SSTEP means single step through a user instruction at the current task.
3. from idle state, TRHOST means transfer the current task to the host CPU.
4. from idle state, LNKCH means send data to APU; receive one's compliment (diagnostic).
5. from idle state, RSCH means reschedule current task at the end of APU queue.
6. STOPAPU means stop APU execution.
7. FETAPU means fetch APU status.

DATA=	n specifies the data sent to the APU during the link check or specifies the register to receive the returned value. This parameter must be on a halfword boundary in the task's writable segment. Bits 8-15 contain the data that is sent to the APU. Bits 0-7 contain the data returned by the APU. This parameter is used only if LNKCH is entered with the CNTFRN parameter.
RELEASE=	Y/N specifies yes or no. Yes means control rights are to be released after completion of the macro function. Other tasks in the system can then gain control rights to the specified APU when this macro completes. This is the default. No means the user does not want to release control rights after completion of this macro function. No allows the user to keep control rights to the specified APU after the macro completes. All other tasks in the system are then prohibited from sending commands to that APU, including the command processor until the controlling task releases those privileges by reentering the APUCNTL macro and specifying Yes with the RELEASE parameter. Yes is the default.

Upon completion of the APUCNTL macro, the following parameter block fields are defined:

- SV13.ERR - error status

byte 1	contains the bit position of the option being executed when the error occurred.
byte 2	contains the error code indicating the type of error.

- SV13.APS receives the APU response status returned after execution of this macro call. The APU status consists of a response byte followed by an error code byte. The response and error code bytes have the format shown in Figure 7-4.

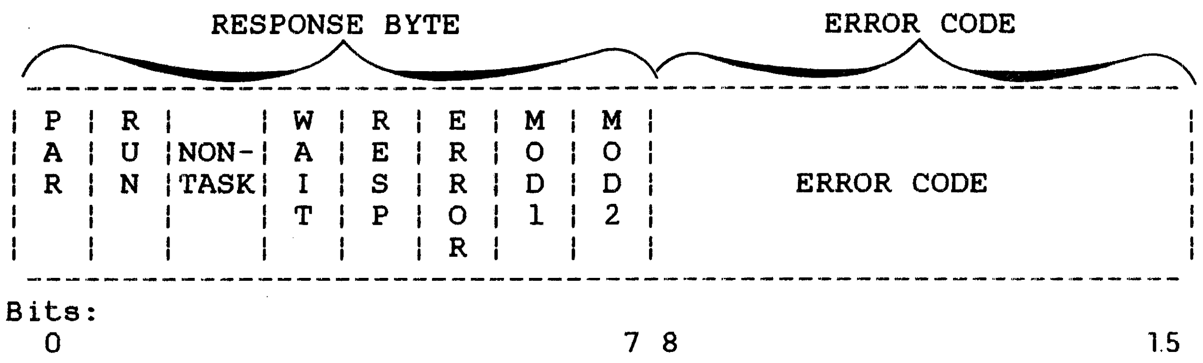


Figure 7-4 APU Hardware Response Bytes

PAR	is the parity bit that ensures the status byte has odd parity.
RUN	is set if the APU is running, it is reset if the APU is idle.
NON-TASK	is set if the current program status word (PSW) bit 15 is set and indicates that no context save area is available. If the bit is reset, the current task must be defined and its task context be ready to accept the current task state active in the APU.
WAIT	is set if the current PSW bit 16 is set or if the APU is working in an internal service state; i.e., scheduling a task. If the bit is reset, the APU is executing instructions.
RESP	is set if the APU is responding to a command from the CPU. If the bit is reset, the APU is generating its own signal indicating a change in APU state.
ERROR	is set if the APU detects an error condition which causes the APU to stop. An error code must be read from the APU to identify the error and to release the APU from the IDLE-ERROR state.
MOD1, MOD2	are set depending on the state of RESP and ERROR. They are encoded to identify one of the following specific conditions:

	RESP	ERROR	MOD1	MOD2	Meaning
Signal, No_error	0	0	0	0	Undefined
	0	0	0	1	APU entering Queue Wait State
	0	0	0	0	Task rescheduled to APU Queue
	0	0	1	1	Task rescheduled to CPU
Signal, Error	0	1	0	0	General error status
	0	1	0	1	Error while in Queue Wait
	0	1	1	0	Error while locking queue
	0	1	1	1	Undefined
Response, No_error	1	0	0	0	General response status
	1	0	0	1	Task is waiting on APU Queue
	1	0	1	0	APU trying to lock a queue
	1	0	1	1	Command Sequence Error
Response, Error	1	1	0	0	Error as result of command
	1	1	0	1	Response, error in Queue Wait
	1	1	1	0	Response, error in Queue Lock
	1	1	1	1	Error as result of command sequence

The error codes defined for the second byte are presented in Table 7-2.

TABLE 7-2 ERROR CODES FOR APU STATUS BYTE

ERROR	CODE	DESCRIPTION
-	X'80'	No error
STA.IDFS EQU	X'01	APUID DEVICE FALSE SYNC
STA.ZID EQU	X'02	ZERO APUID RETURNED BY RTSM
STA.MPCR EQU	X'83	CANNOT FETCH WORDS X'C4'-ECC
STA.MAPU EQU	X'04'	APUID > MAX APU @X'C7'
STA.NDIR EQU	X'85'	BAD A(AFB_DIR)-ECC/ZERO/ALIGN
STA.MAPP EQU	X'86'	BAD A(AFB) - ECC/ZERO/ALIGN
STA.MAPN EQU	X'07'	BAD APB(FLAGS:APB#) WORD-ECC
STA.WAPB EQU	X'08'	WRONG APB NUMBER IN APB
STA.APBK EQU	X'89'	ABP PASSBACK
STA.CMDR EQU	X'8A'	UNRECOGNIZED COMMAND
STA.NTCB EQU	X'0B'	BAD APB A(CTCB)-ECC/ZERO/ALIGN
-	X'8C'	NOT USED
STA.QTIM EQU	X'0D'	QUEUE LOCK TIMEOUT
STA.SUSP EQU	X'0E'	EXECUTION SUSPENDED (TRAP PSW WAIT)
STA.NSST EQU	X'8F'	BAD SSTD - ECC

TABLE 7-2 ERROR CODES FOR APU STATUS BYTE (Continued)

ERROR	CODE	DESCRIPTION
STA.NCTX EQU	X'10'	CANNOT LOAD TASK CONTEXT
STA.NCTS EQU	X'91'	CANNOT STORE TASK CONTEXT
STA.NPFI EQU	X'92'	CANNOT LOAD PWB FAIL IMAGE
STA.SPFI EQU	X'13'	CANNOT STORE POWER FAIL IMAGE
STA.NPST EQU	X'94'	CANNOT LOAD PSTD - ECC
STA.NPFP EQU	X'15'	BAD APB PFAIL PTR-ECC/ZERO
STA.HDST EQU	X'16'	BAD APB MMF NEW PSW-FCC/ZERO LOC
STA.WCTP EQU	X'97'	BAD CTCB CTX PTR-ECC/ZERO/ALIGN
STA.NCNT EQU	X'98'	BAD APB TCB CNT WORD - ECC
STA.QFPT EQU	X'19'	BAD A(APU FRONT TCB)-ECC/ZERO/ALIGN
STA.TCNT EQU	X'1A'	FRONT TCB PRT, TCB AND CNT DISAGREE
STA.QUNP EQU	X'9B'	QUEUE TCB CNT UNDERFLOW
STA.NHQP EQU	X'1C'	BAD APB A(CPU QUEUE) - ECC/ZERO/ ALIGN
STA.NQTP EQU	X'9D'	BAD TCB QHPTR - ECC/ZERO/ALIGN
STA.WAPQ EQU	X'9E'	INCORRECT TCB QUEUE HEAD PTR
STA.NBPT EQU	X'1F'	BAD TCB BPTR - ECC/ZERO/ALIGN
STA.NBFP EQU	X'20'	BAD BACK TCB FPTR - ECC/ZERO/
STA.NBFP EQU	X'20'	BAD BACK TCB FPTR - ECC/ZERO/ALIGN
STA.FLNK EQU	X'A1'	BACK TCB FPTR NOT TO FRONT TCB
STA.NFPT EQU	X'A2'	BAD FRONT TCB FPTR-ECC/ZERO/ALIGN
STA.NFBP EQU	X'23'	BAD FWD TCB BPTR-ECC/ZERO/ALIGN
STA.BLNK EQU	X'A4'	FWD TCB BPTR NOT TO FRONT TCB
STA.FLBL EQU	X'25'	INCONSISTENT FRONT TCB FPTR AND BPTR
STA.FBPT EQU	X'26'	BAD FRONT TCB PTR-ECC/ZERO/ALIGN
STA.BFPT EQU	X'A7'	BAD BACK TCB FPTR-ECC/ZERO/ALIGN
STA.WBFP EQU	X'A8'	BACK TCB FPTR NOT TO FRONT TCB
STA.QOVF EQU	X'29'	TCB QUEUE OVERFLOW (CPU OR APU)
STA.TIM1 EQU	X'2A'	BAD MSH TIME ACCUMULATOR - ECC
STA.TIM2 EQU	X'AB'	BAD LSH TIME ACCUMULATOR - ECC
STA.STIM EQU	X'2C'	BAD TCB START TIME WORD - ECC
STA.STIM EQU	X'2C'	BAD TCB START TIME WORD - ECC
STA.NRTC EQU	X'AD'	CANNOT READ RTSM CLOCK DATA
STA.TMOV EQU	X'AE'	TCB ELAPSED TIME OVERFLOW
STA.PEND EQU	X'2F'	TCB PENDING FLAGS SET ON QUEUE
STA.NPND EQU	X'B0'	BAD PENDING FLAGS WORD - ECC OR CTCB
STA.NPND EQU	X'B0'	BAD PENDING FLAGS WORD - ECC
STA.XINT EQU	X'31'	INTERRUPT FROM RTSM XMTR
STA.PFPS EQU	X'32'	CANNOT LOAD PFAIL PSTD - ECC
STA.PFSS EQU	X'B3'	CANNOT LOAD PFAIL SSTD - ECC

Functional Details:

If appcb is not specified, the parameter block is automatically built and the required parameters are assigned to the appropriate fields. If appcb is specified, all the required parameters not specified in the macro command are assumed to be in the parameter block.

To successfully execute this macro, the task must be established with the APU Control (APC) link option. No other task can have the APU control rights granted to it.

Example:

In this example, the APUCNTL macro enables APU1:

```
    APUCNTL ,APN=1 ,CNTRFN=ENABLE
```

In the following example, the APUCNTL macro performs a link check on the APU specified in R1. It sends the data byte located in bits 24-31 of R3 and receives the returned data in bits 16-23 of R3.

Example:

```
    APUCNTL ,APN=(R1) ,CNTRFN=LNKCH ,DATA=(R3)
```

7.5.2 REQUEUE (Requeue the APU Ready Queue) Macro

The REQUEUE macro gives a task the ability to preempt or reschedule the current task and select the next task on an APU ready queue. The optional parameters allow a task limited ability to reorder the APU ready queue.

Format:

NAME	OPERATION	OPERAND
symbol	REQUEUE	[appcb] [,APN=n] [,TASKID=n]
		[,RESTART=Y/N] [,RESCHED=Y/N]
		[,RELEASE=Y/N]

Parameters:

appcb	is the address or pointer to the appcb parameter block or a register containing the address of the parameter block.
APN=	n specifies the number of the APU to which this function is directed. The task must be assigned to a nonzero LPU, which, in turn must be mapped to a nonzero APU. APU 0 indicates the CPU.
TASKID=	n specifies the address or pointer to a valid task name or the register containing the address of the valid task name.
RESCHED=	Y/N specifies Yes or No. Yes means the currently active task is to be rescheduled back on the APU ready queue before the queue pointer is changed. No means the currently active task is not to be rescheduled on the APU ready queue before the queue pointer is moved. Yes is the default value.
RESTART=	Y/N specifies Yes or No. Yes is the default and restarts the APU after the APU at the top of the ready queue was changed. No specifies that the APU is not to be restarted.

RELEASE=	Y/N specifies Yes or No. Yes is the default and means that the control rights to the specified APU are to be released. No means the control rights are not to be released and no other task in the system (including the command processor) can have control of the APU.

Functional Details:

If appcb is not specified, the paramater block is automatically built by the APPCB macro.

The TASKID parameter must point to a valid task name. This task name must be left-justified, padded with blanks, if necessary, and must start on a fullword boundary. The task name must be on the APU ready queue. If a valid task name is not specified, the task at the head of the queue is assumed to be the task to be preempted.

The RESCHED parameter entered with yes keeps the currently active task in the same position relative to all of the other tasks in the APU ready queue. No causes the currently active task to be positioned just behind a valid task name specified when the current task is finally put back on the queue and gives the user the ability to reorder the ready queue.

The REQUEUE macro could result in any one of the following error conditions:

ITO	The task option prevents granting of control rights. Relink the program using the OPTION APCONTROL command. See the OS/32 Link Reference Manual.
IAN	An invalid APU number was specified. Verify that the APU number has a value from zero through the highest APU number on the system.
IOS	An invalid option byte was specified. Check location SV13.OPT in the parameter block to verify the contents.
COP	The requested control rights are held by another task. There is no recovery from this error except to wait for the other task to release the control rights and try this call again.
DIS	Request is denied because the APU is disabled. The APU must be made active before this call is entered. The optional TASKID parameter secifying the valid task name also must be on the APU ready queue.

EIT	Error occurred in the transmission of a command. This error is significant of hardware failures. Retry and, if the error recurs, call the customer engineer.
NTQ	The task name in the TASKID field was not found on the APU ready queue. Reenter this macro. If the error disappears, the desired task was probably executing an SVC call on the CPU. If the error remains, check the task name and the location of the task in the system.

Examples:

In this example, the REQUEUE macro stops the APU whose number is contained in R2. The pointer to the next task on the ready queue is changed to point to the task whose name is located in BUFFER (TASKNAME). The APU is then restarted:

```
        REQUEUE ,APN=(R2) ,TASKID=BUFFER

          .
          .
          .
    ALIGN 4
    BUFFER DB C'TASKNAME'
```

7.6 TASK DIRECTION MACROS

The Model 3200MPS System task direction macros direct a task to an APU or from an APU back to the CPU. In order to direct a task to a particular APU or from a particular APU back to the CPU, the LPU has to be assigned to the processor where the task is to execute. The LPU must be assigned to an APU with a nonzero number because only the CPU can have a zero designation. The task direction macros utilize the tmpcb parameter block built by the TMPCB macro or built automatically. See Section 5.23. There are two ways to assign the LPU to an APU.

- Self-directed assignment of an LPU where a task sets its own LPU assignment. Self-directed assignment does not require a task name in the task name field of the parameter block built by the TMPCB macro. However, the LPU must be specified in the parameter block.

- Other-directed assignment of an LPU where the LPU is assigned by the task, but the behavior of the task is not affected until the task is transferred from the CPU.

The task direction macros are SETCPU and SETLPU.

7.6.1 SETCPU (Set CPU-Directed Task) Macro

If a task is set to run on an APU and the user wants it set to the CPU, the SETCPU macro will direct task execution to the CPU by resetting the LPU-directed status.

Format:

NAME	OPERATION	OPERAND
symbol	SETCPU	[tmpcb] ,TASKID=n [,DIR=n]

Parameters:

tmpcb	specifies the address or pointer to the parameter block built by the TMPCB macro, or to a register that holds the parameter block address. This is the default if the tmpcb parameter block is built automatically.
TASKID=	n specifies the address or pointer to a valid task name or to a register containing the address of the valid task name. When a valid task name is specified, it is moved into the tmpcb parameter block. If no valid task name is specified, it is assumed to already be in the tmpcb parameter block. This parameter is required only if the tmpcb is not entered.
DIR=	n specifies the type of assignment: S means self-directed assignment and O means other-directed assignment. Other-directed assignment is the default.

Functional Details:

If tmpcb is not specified, the parameter block is automatically built by calling the TMPCB macro and the functions are set according to the DIR parameter. The TMPCB macro is then called to build the tmpcb parameter block.

The valid task name specified with the TASKID parameter must be left-justified in an 8-byte field, padded with blanks, and fullword boundary aligned. When the SETCPU macro is called for other than a self-directed task, the SUSPEND function (see Section 5.23) is called to place the directed task in the task wait state; the LPU-directed status of the task is reset; and the RELEASE function (see Section 5.16) is called to release the directed task from the suspend wait state.

Example:

This example shows how the execution of the SETCPU macro redirects the task whose name is located in BUFFER (TASKNAME) to execute on the CPU. This is an example of other-directed assignment:

```
        SETCPU ,DIR=OT ,TASKID=BUFFER
          .
          .
          .
    BUFFER DB C'TASKNAME'
```

7.6.2 SETLPU (Set LPU-Directed Task) Macro

The SETLPU macro sets the LPU-directed task state. It can also assign a new LPU to the task. If the task is self-directed on its default LPU, it duplicates the function of the CAL/32 reschedule task (RSCH 2) instruction. See the Common Assembly Language/32 (CAL/32) Reference Manual.

Format:

NAME	OPERATION	OPERAND
symbol	SETLPU	[tmpcb] [,DIR=n] [,LPU=n] ,TASKID=n

Parameters:

tmpcb	is the address or pointer to the tmpcb parameter block or to a register containing the address or pointer to the parameter block. This is the default if the tmpcb parameter block is built automatically.
DIR=	n specifies the type of LPU assignment which is either self-directed, where a task sets its own LPU number; or other-directed, where the macro sets the LPU number of the task specified in the TASKID parameter.
LPU=	n specifies the LPU or the register to which the task is reassigned.
TASKID=	n specifies the address or the pointer to a valid task name or to a register containing the address of a valid task name. This parameter is required only if tmpcb is not specified.

Functional Details:

If tmpcb is not specified, the parameter block is automatically built by calling the TMPCB macro and the functions are set according to the DIR parameter. The TMPCB macro is then called to build the tmpcb parameter block.

If a valid task name is specfiied, it is moved into the tmpcb parameter block. If a task name is not specified, it is assumed to be in the tmpcb. The TASKID must be left-justified in an 8-byte field, padded with blanks, and fullword boundary aligned.

When the SETLPU macro is invoked for an other-directed task, the following steps take place:

- The SUSPEND macro is invoked to place the directed task in the task wait state.
- The directed task's LPU is reassigned to the LPU specified with the LPU parameter.
- The directed task is marked for transfer to LPU.
- The RELEASE macro is entered to release the directed task from the task wait state.

If the task is self-directed, or the called task has a higher priority than the calling task, the called task is immediately dispatched to the LPU specified.

Examples:

This example shows how execution of the SETLPU macro sets its own (self-directed) LPU number to 2:

```
SETLPU ,DIR=SD ,LPU=2
```

Execution of the SETLPU macro shown in the following example sets the LPU number of the task specified in location BUFFER (TASKNAME) to 3. This other task must already be in the system:

```
          .
          .
          .
        SETLPU ,DIR=OT ,LUP=3 ,TASKID=BUFFER
          .
          .
          .
    BUFFER DB C'TASKNAME'
```

7.7 TASK TIMER MACROS

The Model 3200MPS System task timer macros measure real-time performance for a system configured with a real-time support module (RTSM). These macros use the CAL/32 RRTC instruction and allow a user to instrument a program and measure its real-time performance. See the Common Assembly Language/32 (CAL/32) Reference Manual for information on the RRTC instruction. These macros measure real-time performance by:

- Creating a parameter block used by the other software interval timer macros.
- Initializing a software interval timer
- Starting a timer
- Returning the total time accumulated
- Returning the total number of times a timer is started or stopped
- Stopping a timer

The Model 3200MPS System timer macros are:

- CRTIMERS
- RESETIME
- STARTIME
- GETIME
- READTCNT
- STOPTIME

7.7.1 CRTIMERS (Create Software Interval Timer) Macro

The CRTIMERS macro creates a parameter block that will be used by the other software interval timer macros in the program to reset the interval timer, initialize it, record the total time accumulated by the timer, read the number of times the timer was activated, and stop the timer. The generated information is stored in the appropriate fields of the parameter block after each of the timer macro functions is completed. This macro does not generate executable code.

Format:

NAME	OPERATION	OPERAND
symbol	CRTIMERS	name1[,name2,...,namen]

Parameters:

name	specifies a 1- to 8-character name representing a software interval timer. Each timer name must be unique within the user program. This name is used by the RESETIME, GETIME, READTCNT and STOPTIME macros.

Functional Details:

The CRTIMERS macro uses 24 bytes for each name specified and must be called in a program segment with read and write access.

The parameter block created by this macro is shown in the examples.

Examples:

In this example, the CRTIMERS macro creates a parameter block with storage space that is 6 words long, on a fullword boundary in the impure segment of code. The format of the parameter block the CRTIMERS macro creates is shown in Figure 7-5.

```
-----------------------------------------------------------------
|0(0)           |1(1)           |2(2)           |3(3)           |
|   Reserved    |  Timer Name   |    Count      |  Start Value  |
|               |               |               |               |
|---------------------------------------------------------------|
|4(4)                                                           |
|                    Accumulator Save Area                      |
|                              .                                |
|                              .                                |
|                              .                                |
-----------------------------------------------------------------
```

Figure 7-5 Format Returned by CRTIMERS Macro

This is an example of how the CRTIMERS macro sets up three software interval timers:

```
                                    1*          MLIBS 8,9,10
                                    2           NLIST
                                    5           LIST
                                    6           CRTIMERS TIMER1,TIMER2,TIMER3
000000:I                            6+          ALIGN 4
             0000 0000:I            6+ TIMER1   EQU   *
000000:I     5449 4D45 5231 2020    6+          DB    C'TIMER1  '          TIMER NAME FIELD
000008:I     0000 0000              6+          DCF   0                    TIMER COUNTER WORD
00000C:I     0000 0000              6+          DCF   0                    TIMER START VALUE
000010:I     0000 0000              6+          DCF   0                    ACCUMULATED TIME
000014:I     0000 0000              6+          DCF   0                    REGISTER SAVE AREA
             0000 0018:I            6+ TIMER2   EQU   *
000018:I     5449 4D45 5232 2020    6+          DB    C'TIMER2  '          TIMER NAME FIELD
000020:I     0000 0000              6+          DCF   0                    TIMER COUNTER WORD
000024:I     0000 0000              6+          DCF   0                    TIMER START VALUE
000028:I     0000 0000              6+          DCF   0                    ACCUMULATED TIME
00002C:I     0000 0000              6+          DCF   0                    REGISTER SAVE AREA
             0000 0030:I            6+ TIMER3   EQU   *
000030:I     5449 4D45 5233 2020    6+          DB    C'TIMER3  '          TIMER NAME FIELD
000038:I     0000 0000              6+          DCF   0                    TIMER COUNTER WORD
00003C:I     0000 0000              6+          DCF   0                    TIMER START VALUE
000040:I     0000 0000              6+          DCF   0                    ACCULATED TIME
000044:I     0000 0000              6+          DCF   0                    REGISTER SAVE AREA
000048:1                            7           END
```

7.7.2 RESETIME (Reset Software Interval Timer) Macro

The RESETIME macro initializes the named software interval timer that will measure the real-time performance of a task running on a CPU or an APU. Each time the interval timer's count is initialized by the RESETIME macro, its count is set to zero.

Format:

NAME	OPERATION	OPERAND
symbol	RESETIME	name1 [,name2,...,namen]

Parameters:

name	specifies a 1- to 8-character name of an interval timer that must have been defined by a CRTIMERS macro. This is a required parameter.

Functional Details:

The named timer's count, accumulated time, and state are reset to zero when initialized by this macro.

Examples:

In this example, the RESET macro resets timers TIMER1, TIMER2, and TIMER3 back to their initial value of zero.

```
    RESETIME TIMER1 ,TIMER2 ,TIMER3
```

7.7.3 STRTIME (Start Software Interval Timer) Macro

The STRTIME macro begins a timing interval for the named software interval timer. The interval timer will measure the real-time performance of a task running on a CPU or an APU.

Format:

NAME	OPERATION	OPERAND
symbol	STRTIME	name [,reg]

Parameters:

name	specifies a 1- to 8-character name for a timer that must have been declared in a CRTIMERS macro.
reg	specifies a register that can be used by the macro for a working or scratch register. If this parameter is not specified, the macro will select, save and restore the contents of its own work register.

Functional Details:

The named timer acts like a stopwatch. This macro starts the theoretical stopwatch, but has no effect if this named timer was already started by a previous STRTIME macro.

This macro reads the current value of the real-time counter and saves it in the named timer parameter block. It also sets a start timer flag.

Examples:

This is an example of how the STRTIME macro starts the software interval timer named TIMER1. Since no work register was specified, this macro saves its own register and restores the original values when it completes:

```
    STRTIME TIMER1
```

In this example, the macro starts a timer and passes it to a specified work register:

```
    STRTIME TIMER2,R2
```

7.7.4 GETIME (Read Software Interval Timer) Macro

The GETIME macro returns the total time accumulated, in microseconds, for the named software interval timer and stores the accumulated time in a user-specified register.

Format:

NAME	OPERATION	OPERAND
symbol	GETIME	name ,reg

Parameters:

name	specifies a 1- to 8-character name of a timer that must have been declared in a CRTIMERS macro. This parameter must be entered.
reg	specifies the user register that will receive the accumulated time. This is a required parameter.

Functional Details:

If the named timer is active, it does not include the time for the current interval, but only the time for all previously completed intervals.

The 32-bit time value returned by this macro is treated as an unsigned integer. Its range is approximately 68 minutes.

There is no check for a counter overflow condition in the specified register. It is the user's responsibility to identify overflow conditions.

Examples:

This example shows how the GETIME macro obtains the accumulated amount of time for a timer named TIMER1 and returns the time in R2. The time returned is a 32-bit unsigned integer and is in microseconds.

```
    GETIME TIMER1 ,R2
```

7.7.5 READTCNT (Read Software Interval Timer's Count) Macro

The READTCNT macro returns a number that shows how many times the named software interval timer was activated (started and stopped) since the last time it was reset and stores the resulting 32-bit unsigned integer in a user-specified register.

Format:

NAME	OPERATION	OPERAND
symbol	READTCNT	name ,reg

Parameter Values:

name	specifies a 1- to 8-character name that must have been declared in a CRTIMERS macro. This parameter is required.
reg	specifies the user register that will receive the interval timer's count. This is a required parameter.

Functional Details:

The 32-bit value returned represents the number of time intervals measured. The accumulated time for this timer divided by the timer's count represents the average time interval for this timer.

The module count of this counter is 2,147,483,647. There is no check in the macro to identify an overflow condition. It is the user's responsibility to identify the overflow condition.

Examples:

In this example, the READTCNT macro obtains the number of times that TIMER1 was started and stopped since the last time it was reset. The macro returns the value in R3 as a 32-bit unsigned integer.

```
    READTCNT TIMER1 ,R3
```

7.7.6 STOPTIME (Stop Software Interval Timer) Macro

The STOPTIME macro ends a timing interval for the named software interval timer. This macro saves and stores the results in a user-specified register. If no register is specified, this macro saves and restores the contents of its own register.

Format:

NAME	OPERATION	OPERAND
symbol	STOPTIME	name [,reg]

Parameters:

name	specifies a 1- to 8-character name of a timer that must have been declared in a CRTIMERS macro. This parameter is required.
reg	specifies a register that can be used by the macro as a working or scratch register. If a register is not specified, the macro saves and restores the contents of its own work register. When the macro completes, the original value is restored in the register.

Functional Details:

The named interval timer acts like a stopwatch. This macro stops the theoretical stopwatch that was started by a STRTIME macro. Otherwise, this macro has no effect on the interval timer.

Examples:

In this example, the STOPTIME macro stops TIMER2 using R2 as its work register:

```
    STOPTIME TIMER2 ,R2
```

This example shows how the STOPTIME macro saves and restores its work register because a register parameter was not specified with the macro:

```
    STOPTIME TIMER1
```

CHAPTER 8
MISCELLANEOUS MACROS

8.1 INTRODUCTION

This chapter presents macros that generate a character constant of a specified length or a message; compare the contents of two fields for a specified length; load a range of registers from an area; move characters; skip to a blank, dump of a storage area; generate the subroutine for the SNAP macro; and save a range of registers in storage.

The formats, parameter values, default values, required parameters, programming considerations, examples, and error messages are supplied for each macro presented in this chapter.

Section 1.4, Parameter Field Value Mnemonics, explains the lowercase abbreviations that appear in the parameter field.

8.2 GENERATE A CHARACTER CONSTANT OF A SPECIFIED LENGTH (CHAR)

The CHAR macro generates a character constant of a specified length. The constant is padded with trailing blanks.

Format:

```
    [symbol]  CHAR  string[,len]
```

Parameter Values:

string	-	'quoted string'
len	-	absolute expression

Default Value:

len	-	length of the string

Programming Considerations:

A character constant is generated for the quoted string. If an optional length is specified as an absolute expression, the string is padded on the right with blanks. There is no limit to the length of the string. No boundary alignment is required or performed. If the length is less than the quoted string, the quoted string is then generated. The length is effectively ignored.

Example:

```
    ALPHA   CHAR   'STRING'
```

Generates 6 bytes

```
    BETA    CHAR   'STRING',9
```

Generates 9 bytes - STRINGbbb

8.3 COMPARE LOGICAL CHARACTER (CLC)

The CLC macro compares the contents of two fields for length. The condition code is set as described in the Programming Considerations section.

Format:

```
    [symbol]  CLC  field1,field2,length
```

Parameter Values:

field1	-	addrx
	-	(reg)
field2	-	addrx
	-	(reg)
length	-	absolute halfword expression
	-	(reg)

Required Parameters:

field1
field2
length

Programming Considerations:

The contents of field1 are compared byte-by-byte to the contents of field2. The comparison stops when the two fields are equal and the length is exhausted, or when the two fields are not equal. The resultant condition code is set as follows:

C	V	G	L	
0	X	0	0	Field1 is equal to field2.
1	X	0	1	Field1 is less than field2.
1	X	1	1	Field1 is less than field2.
0	X	0	1	Field1 is greater than field2.
0	X	1	0	Field1 is greater than field2.

X = undetermined value

If (reg) or an INDEX register is specified for field1 or field2, the register value is repositioned to the unequal byte or to the last equal byte. A zero length always results in equality.

Error Messages:

```
MNOTE CORRECT FORM IS CLC A,B,LEN
Return code = 4
```

8.4 DEFINE SYSTEM MACRO ENVIRONMENT (ENVIRON)

The ENVIRON macro defines the registers to be used in system macro expansion. This macro also allows an error handling subroutine, which is to be done in other modules, to be generated.

Format:

```
    blank ENVIRON options [,PCBREG=][,SCRREG=]
```

Parameter Values:

options	-	ERRSUB (produces the error handling subroutine in this module and sets its name as an ENTRY)
	-	NERRSUB (the error handling subroutine is defined in another module; the names are defined as EXTRN)
	-	SNAPSUB (the subroutine for the SNAP macro is defined in this module; its name is ENTRY)
	-	NSNAPSUB (the subroutine for the SNAP macro is defined in another module; the name is defined as EXTRN)
PCBREG	=	reg (defines the register to be used as the pointer to PCBs by other system macros)
SCRREG	=	reg (defines the register to be used as the scratch register to modify the PCBs by other system macros)

Default Values:

PCBREG = 14 (PCB pointer register)

Subroutines are defined in this routine; no linkage is provided. SCRREG should not be 0 since it is used for indexing and branching.

Programming Considerations:

When writing a program as a series of independent routines to be linked together, it is not desirable to have copies of the error handling subroutines or the SNAP subroutine repeated in each routine. The ENVIRON macro can be coded to define or not define the subroutines in a given routine. If several routines are written, the ENVIRON macro must then be coded in each routine.

When using system macros to modify existing programs, R14 and R15 might be needed for other purposes. The ENVIRON macro can redefine these registers to other values. The subsequently generated code from the system macros will use these registers.

Example:

```
Routine 1
ENVIRON  ERRSUB
    .
    .
    .
END


Routine 2
ENVIRON  NERRSUB
    .
    .
    .
END

Routine 3
ENVIRON  NERRSUB
    .
    .
    .
END
```

The error handling subroutine is generated in Routine 1 and referenced in Routines 2 and 3.

```
ENVIRON PCBREG=2,SCRREG=4
```

causes system macros to use R2 as the PCB pointer and R4 as the scratch register until another ENVIRON macro is encountered.

8.5 FETCH ACCOUNTING INFORMATION (FETACCT)

The FETACCT macro fetches accounting information and stores it in a user-specified, 16-byte area. This area must be in a writable segment.

Format:

```
[symbol]  FETACCT  actinfo [,PCB=][,FORM=]
```

Parameter Values:

actinfo	-	addrx (address or pointer to 16-byte area area where accounting information is to be stored)
	-	(reg) - address or pointer to 16-byte area where accounting information is to be stored
PCB	=	addrx (address or pointer to PCB)
	=	(reg) - address or pointer to PCB
FORM	=	L (list form; only generate PCB)

Programming Considerations:

The accounting information is returned as:

- 4 bytes for user time
- 4 bytes for operating system time
- 4 bytes for wait time
- 4 bytes for roll time

These 16 bytes must be fullword boundary aligned in a writable segment.

8.6 GENERATE A MESSAGE (GENMSG)

The GENMSG macro can generate a number of variable length messages. The first byte of the message is the length of the message including the length byte. Thus, a message of 6 characters actually generates 7 bytes.

Format:

```
    [symbol] GENMSG 'quoted string'
                    [,'quoted string'...]
```

Programming Considerations:

For each string, a byte is generated containing the length of the string plus one (for the byte itself), followed by the string. No boundary alignment is performed. The maximum message length is 255 characters.

GENMSG is useful for generating variable length messages. Find the next message by adding the length byte to a register pointer.

Example:

```
                .
                .
                .
    MESSAGE     GENMSG 'MESSAGE 1'
                GENMSG 'MESG2','MSG 3'
                .
                .
                .
```

3 messages of different lengths

```
          LDAI    6,MESSAGE    Point R6 to beginning
                               of message table
          LIS     4,0          Value of first status
LOOP      EQU     *
          LB      5,0(6)       Get length
          CLB     4,STATUS     STATUS is a byte with a
                               value between 0 through 2
          BE      WRITE        Value found
          AAR     6,5          Add length
          AIS     4,1          Next status
          BS      LOOP         Go back
WRITE     EQU     *
          SIS     5,1          Subtract 1 for actual
                               length of message
          WRITE   OUTPCB,ADDR=1(6),RECL=(5) Write message
```

8.7 LOAD A RANGE OF REGISTERS FROM AN AREA (LDREG)

The LDREG macro loads a range (any number) of registers from a given area. If the END register is less than the START register, a wrap-around feature is built in; that is, 14 through 2 loads registers 14, 15, 0, 1, and 2.

Format:

```
    [symbol]  LDREG  start,end,area
```

Parameter Values:

start	-	decimal register number
end	-	decimal register number
area	-	addrx

Default Values:

none

Required Parameters:

start
end
area

Examples:

Load registers 2 through 8 from address ALPHA:

```
     LDREG 2,8,ALPHA
```

Load registers 14,15,0,1,2 from pointer 7:

```
     LDREG 14,2,0(7)
```

8.8 MOVE CHARACTER (MVC)

The MVC macro moves the contents of a source field into a destination field byte-by-byte.

Format:

```
    [symbol]  MVC  dest,source,length[,EC=]
```

Parameter Values:

dest	-	addrx
	-	(reg)
source	-	addrx
	-	(reg)
length	-	absolute halfword expression
	-	(reg)
EC	=	(immediate byte expression,...)

Default Values:

EC	=	no ending characters

Required Parameters:

dest
source
length

Programming Considerations:

The source field is moved byte-by-byte into the destination field starting at the left-most byte. Overlapping fields can occur.

If (reg) is specified or an INDEX register is used, the register is adjusted to the byte immediately following the last byte moved; that is, two MVCs using the same register result in concatenating the fields without having to adjust the register.

Error Messages:

```
    MNOTE   CORRECT FORM IS   MVC   TO,FROM,LENGTH
```

8.9 SET ACCOUNTING INFORMATION (SETACCT)

The SETACCT macro provides a means of setting 8 bytes of information into the AFT task completion or data overflow account records.

Format:

```
[symbol]  SETACCT  word1,word2[,PCB=][,FORM=]
```

Parameter Values:

word1	-	addrx ('1-4 byte string'; address to first fullword of information)
	-	reg (register containing first fullword of information)
word2	-	addrx ('1-4 byte string'; address to second fullword of information)
	-	reg (register containing second fullword of information)
PCB	=	addrx (address or pointer to PCB)
	=	(reg) - address or pointer to PCB
FORM	=	L (list form; only generate PCB)

Programming Considerations:

Both parameters, word1 and word2, must be specified. If PCB= is coded, word1 and word2 do not have to be specified; the information is assumed to be in the parameter block. If coded as quoted strings, each can be 1- to 4-characters enclosed in quotes. If specified as an addrx, the effective address must be on a fullword boundary.

8.10 SKIP TO BLANK (SKTB)

The SKTB macro repositions a register pointer to the next ASCII blank. It is useful in parsing an input string.

Format:

```
[symbol] SKTB reg
```

Parameter Values:

reg	- register expression

Default Values:

none

Required Parameters:

reg

Programming Considerations:

The register pointer is repositioned to point to the next blank character. If the register is already pointing to a blank character, it is left unchanged.

Error Message:

```
MNOTE NO REGISTER SPECIFIED - NO EXPANSION
Return code = 4
```

8.11 SKIP TO CARRIAGE RETURN (SKTCR)

The SKTCR macro repositions a register pointer to the next ASCII carriage return. It is useful in parsing an input command.

Format:

```
    [symbol]  SKTCR  reg
```

Parameter Values:

reg	-	register expression

Default Values:

none

Required Parameters:

reg

Programming Considerations:

The register pointer is repositioned to point to the next carriage return character. If the register is already pointing to a carriage return, it is left unchanged.

Error Message:

```
    MNOTE NO REGISTER SPECIFIED - NO EXPANSION
    Return code = 4
```

Example:

These macros are useful for parsing an input line and isolating the operands. If R3 points to a buffer containing operands separated by blanks or commas, a carriage return terminates the line, and the length of an operand is to be computed:

```
      .
      .
      .
      READ    INPCB,ADDR=LINE    Read the line

      LDAI    3,LINE             Point to beginning of
                                 the line

      LDAR    4,3
      SKTCR   4                  R4 now points to
                                 carriage return at
                                 end of line

      SKTNB   3                  R3 points to the first
                                 non-blank; it can be
                                 beginning of line

      LDAR    5,3                Hold pointer

      SKTD    5                  Now R3 is at the beginning
                                 and R5 is one byte past
                                 the operand

            SAR     5,3                R5 now has the length of
                                       operand
            .
            .
            .
```

8.12 SKIP TO DELIMITER--BLANK, COMMA, CARRIAGE RETURN (SKTD)

The SKTD macro repositions a register pointer to the next ASCII blank, comma, or carriage return if there are no user-specified delimiters or up to three user-specified delimiters. It is useful in pausing an input command.

Format:

```
    [symbol]  SKTD  reg[,D=]
```

Parameter Values:

reg	-	register expression
D	-	up to three absolute byte expressions enclosed in parentheses

Default Values:

None

Required Parameters:

reg

Programming Considerations:

The register pointer is repositioned to point to the next delimiter defined as one of the following:

- Blank, comma, or carriage return if there are no user-specified delimiters
- Up to three user-specified delimiters

If the register is pointing to a delimiter, it is left unchanged.

Error Message:

```
    MNOTE NO REGISTER SPECIFIED - NO EXPANSION
    Return code = 4
```

8.13 SKIP TO NONBLANK (SKTNB)

The SKTNB macro repositions a register pointer to the next byte which is not an ASCII blank. It is useful in parsing an input command.

Format:

```
    [symbol]  SKTNB  reg
```

Parameter Values:

reg	-	register expression

Default Values:

None

Required Parameters:

reg

Programming Considerations:

The register pointer is repositioned to point to the next nonblank character. If the register is pointing to a nonblank character, it is left unchanged.

Error Message:

```
MNOTE NO REGISTER SPECIFIED - NO EXPANSION
Return code = 4
```

8.14 TAKE A SNAPSHOT DUMP OF AN AREA OF STORAGE (SNAP)

The SNAP macro writes the storage area's contents to the console or an LU in hexadecimal or ASCII format. It is useful in debugging a program.

Format:

```
    [symbol]  SNAP  area,len[,LU=]
```

Parameter Values:

area	-	addrx
	-	(reg)
len	-	absolute halfword expression
	-	(reg)
LU	=	absolute halfword expression

Default Values:

none

Required Parameters:

area
len

Programming Considerations:

A storage area is written to the console or LU in hexadecimal and ASCII format. The area address is rounded down to the nearest fullword. The length is then rounded up to the next 16 bytes. The length is specified in bytes; thus, the minimal area snapped is 16 bytes. If an LU is not specified, SNAP uses the WTO macro. If an LU is specified, an LU must be preassigned. LU=0 cannot be used. Nonprintable characters are printed as a period. The SNAP macro automatically generates the required subroutine by issuing the SNAPSUB macro.

Example:

```
    SNAP  ALPHA,17            would snap 32 bytes
                              starting at the nearest
                              fullword less than or
                              equal to address ALPHA.
```

Error Messages:

```
    MNOTE INVALID NUMBER OF PARAMETERS - CORRECT FORM IS
      SNAP  AREA,LEN
    Return code = 4
```

8.15 GENERATE THE SUBROUTINE FOR THE SNAP MACRO (SNAPSUB)

The SNAPSUB macro generates the subroutine required for the SNAP macro.

Format:

```
    blank    SNAPSUB    blank
```

Programming Considerations:

The SNAP macro issues SNAPSUB; thus, it is not required to issue this macro unless the user wants the subroutine in an impure segment. Only one copy of the subroutine can be generated. The label on the subroutine is SNAPSUB.

8.16 STORE A RANGE OF REGISTERS IN A GIVEN AREA (STREG)

The STREG macro stores a range (any number) of registers in a given area. If the END register is less than the START register, a wrap-around feature is built in; that is, 14 through 2 stores registers 14, 15, 0, 1, and 2.

Format:

```
    [symbol]  STREG  start,end,area
```

Parameter Values:

start	-	decimal register number
end	-	decimal register number
area	-	addrx

Default Values:

    none

Required Parameters:

```
    start
    end
    area
```

Example:

Save registers 2 through 8 in address ALPHA:

```
    STREG 2,8,ALPHA
```

Save registers 14, 15, 0, 1, and 2 in address pointer 7:

```
    STREG 14,2,0(7)
```

INDEX

CUT ALONG LINE

PUBLICATION COMMENT FORM

Please use this postage-paid form to make any comments, suggestions, criticisms, etc. concerning this publication.

From ______	Date ______
Title ______	Publication Title ______
Company ______	Publication Number ______
Address ______	
______	
______	

FOLD FOLD

Check the appropriate item.

☐	Error	Page No. ______	Drawing No. ______
☐	Addition	Page No. ______	Drawing No. ______
☐	Other	Page No. ______	Drawing No. ______

Explanation:

FOLD FOLD

Fold and Staple
No postage necessary if mailed in U.S.A.

6434

STAPLE STAPLE

FOLD FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 22 OCEANPORT, N.J.

POSTAGE WILL BE PAID BY ADDRESSEE

PERKIN-ELMER

Computer Systems Division
2 Crescent Place
Oceanport, NJ 07757

TECH PUBLICATIONS DEPT. MS 322A

FOLD FOLD

STAPLE STAPLE