CRAY RESEARCH, INC.

CRAY-1® AND CRAY X-MP COMPUTER SYSTEMS

MACROS AND OPDEFS REFERENCE MANUAL

SR-0012

CRAY RESEARCH, INC.

CRAY-1® AND CRAY X-MP COMPUTER SYSTEMS

MACROS AND OPDEFS REFERENCE MANUAL

SR-0012

RECORD OF REVISION

CRAY RESEARCH, INC.

PUBLICATION NUMBER SR-0012

Each time this manual is revised and reprinted, all changes issued against the previous version in the form of change packets are incorporated into the new version and the new version is assigned an alphabetic level. Between reprints, changes may be issued against the current version in the form of change packets. Each change packet is assigned a numeric designator, starting with 01 for the first change packet of each revision level.

Every page changed by a reprint or by a change packet has the revision level and change packet number in the lower righthand corner. Changes to part of a page are noted by a change bar along the margin of the page. A change bar in the margin opposite the page number indicates that the entire page is new; a dot in the same place indicates that information has been moved from one page to another, but has not otherwise changed.

Requests for copies of Cray Research, Inc. publications and comments about these publications should be directed to:

CRAY RESEARCH, INC.,
1440 Northland Drive,
Mendota Heights, Minnesota 55120

Revision	Description
	July, 1983 - Original printing.
01	November, 1983 - This revision supports the COS 1.13 release. It updates information about the use and effects of several macros, particularly in relation to new Cray multitasking features. Besides several editorial changes, modifications to this manual include information on ENDRPV's influence on multitasked jobs. Under the positioning macros category of logical I/O macros, POSITION has new features and parameters. Moreover, two new positioning macros are included: SYNCH and TAPEPOS. Two new operands are documented for the LDT permanent dataset macro: CV and FD, used for handling foreign datasets. Under COS-dependent macros, revisions have been made to the description of system task opdefs GETDA and GETNDA. A new dataset locking macro, DSPLOCK, has also been added under COS-dependent macros.

PREFACE

This manual includes macro and opdef instructions for use with the Cray Operating System (COS). Macros are available in Cray Assembly Language (CAL) and are subject to the rules defined in the CAL Assembler Version 1 Reference Manual, publication SR-0000.

The macro and opdef instructions described in this manual are intended for COS users in general. Nonetheless, the macros and opdefs which are included under section 7 are intended primarily for internal system users. Section 8 identifies a particular group of macros which are independent of the operating system.

Other CRI publications that contain macro instructions include:

SR-0000	CAL Assembler Version 1 Reference Manual
SR-0073	COS Simulator (CSIM) Reference Manual
SM-0040	COS EXEC/STP/CSP Internal Reference Manual
SM-0046	IOS Software Internal Reference Manual
SM-0048	IBM MVS Station Internal Reference Manual
SM-0047	CDC NOS Station Internal Reference Manual
SM-0052	CDC NOS/BE Station Internal Reference Manual
SM-0062	DEC VAX/VMS Internal Reference Manual
SM-0070	IBM VM Station Internal Reference Manual

The contents of this manual presuppose reader familiarity with the CRAY-OS Version 1 Reference Manual, publication SR-0011. Likewise, the user should be experienced in coding the Cray Assembly Language (CAL) as described in the CAL Assembler Version 1 Reference Manual, CRI publication SR-0000.

CRI publication SR-0014, the Library Reference Manual, also contains reference information for the macro and opdefs instructions user.

CONTENTS

INTRODUCTION 1

Included with the Cray Operating System is a set of macro and opdef instructions providing the user with a convenient means of table handling or communicating with COS or external routines in a manner that is independent of the exact implementation details. These macro instructions are available only when programming in the Cray Assembly Language (CAL). Some are processed by the assembler using macro definitions defined in the system text, $SYSTXT, and others are defined in COSTXT (section 7).

The following groups of macros and opdefs are described in this manual:

- System action request macros
- Logical I/O macros
- Permanent dataset macros
- CFT linkage macros
- Operational aid macros and opdefs
- COS dependent macros and opdefs
- Structured programming macros

The format for a macro instruction is:

Location	Result	Operand
loc	*name*	$a_1,a_2,\ldots,a_j,f_1=b_1,f_2=b_2,\ldots,f_k=b_k$

loc Location field argument. Certain macros require an entry in this field. For other macros, *loc* is an optional symbolic program address. Macros that generate a table are assigned a word address; macros that generate executable code are assigned a parcel address.

name Name of macro as given in system text

a_i Actual argument string corresponding to positional parameter in prototype. Two consecutive commas indicate a null string.

f$_j$=*b*$_j$ Keyword and actual argument; these entries can be in any order. A space or comma following the equal sign indicates a null string.

Stacked items within braces signify that only one of the listed items must be entered.

A parameter shown in all UPPERCASE letters must be coded literally as shown. A parameter presented in *italics* must be supplied with a value, a symbol, an expression, or a register designator as indicated in the text following the format for each macro.

A macro can be coded through column 72 of a line. It can be continued on the next line by placing a comma in column 1 of the next line and resuming the parameter list in column 2, with no intervening blanks at the end of the first line.

NOTE

Use registers A0 and S0 with care as parameters. When a macro that includes A0 or S0 as a parameter is expanded, special syntax values may be used rather than the value of the contents of A0 or S0.

SYSTEM ACTION REQUEST MACROS 2

The system action request macros are a subset of the system function requests. Each macro generates an instruction sequence that is a call to the Cray Operating System (COS). The octal function value is stored in register S0; S1 and S2 provide additional arguments for some requests. The function is executed when the program exit instruction is executed. Register S0 will have a zero value when the call is completed if there were no errors. If there were errors, an error code is returned. Error codes are defined in the CRAY-OS Version 1 Reference Manual, publication SR-0011.

See the COS EXEC/STP/CSP Internal Reference Manual, publication SM-0040 for more information on system function codes.

The system action request macros are divided into the following main classes:

- Job control macros
- Dataset management macros
- Time or date macros
- Debugging aids macros
- Miscellaneous macros

Most macros that generate executable code have labels.

JOB CONTROL MACROS

Several system action request macros allow the user to set operating characteristics and control job processing. These macros include ABORT, CONTRPV, CSECHO, DELAY, DUMPJOB, ENDP, ENDRPV, IOAREA, JTIME, MEMORY,† MESSAGE, MODE, NORERUN, RECALL, RERUN, RESERVE, ROLL, SETRPV, SWITCH.

† The old version of the MEMORY macro is also supported.

ABORT - ABORT PROGRAM

The ABORT request provides for abnormal termination of the current job step. Processing resumes with the first job control statement following the next EXIT statement unless reprieve processing is enabled. If no such statement exists, the job is terminated.

Format:

Location	Result	Operand
	ABORT	

CONTRPV - CONTINUE FROM REPRIEVE CONDITION

The CONTRPV macro continues normal job processing from within a reprieve subroutine. The program address to continue processing and all A, S, and VL register values are taken from the user-supplied exchange package.

Format:

Location	Result	Operand
	CONTRPV	*xp*

CSECHO - SEND STATEMENT IMAGE TO THE LOGFILE

CSECHO enters the statement at the specified location into the system log and user logfile. This macro does not echo the statement to the user logfile if the statement originated as terminal input from an interactive job. Echoing is also governed by the current ECHO state for JCL statements. (See the CRAY-OS Version 1 Reference Manual, publication SR-0011.)

Format:

Location	Result	Operand
	CSECHO	*address*

address A symbol or an A, S, or T register (except S0, S1, S2) containing the base address of the statement image. This is a required parameter.

DELAY - DELAY JOB PROCESSING

This function removes a job from execution and delays the job from becoming a candidate for processing until the number of milliseconds (specified in the word at the given address) has elapsed.

Format:

Location	Result	Operand
	DELAY	*address*

address A symbol or an A, S, or T register containing the address of the word that contains the number of milliseconds to delay

DUMPJOB - DUMP JOB IMAGE

The DUMPJOB request causes the current job image (including the JTA) to be written to a specified local dataset. If the dataset already exists, it is rewound before writing; otherwise, a new dataset is created for the dump. The dump is formatted as suitable for the dump utility. If the dataset already exists and is a permanent dataset, the job must have unique access and write permission.

Format:

Location	Result	Operand
	DUMPJOB	DN=*dn*

DN=*dn* A symbol or an A, S, or T register (not A0 or S0) c ntaining the address of a dataset name. If *dn* is not specified, $DUMP is assumed. If location *dn* is not defined, the DUMPJOB macro generates the symbolic location.

ENDP - END PROGRAM

The ENDP request causes normal termination of the current program. Processing resumes with the next job control statement if reprieve processing is not enabled for normal job step termination. If reprieve processing is enabled for normal job step termination, the user's reprieve code is entered.

Format:

Location	Result	Operand
	ENDP	

ENDRPV - END REPRIEVE PROCESSING

The ENDRPV request returns to job step termination processing. If the step completed normally, normal termination is completed. If the step aborted, abort processing is resumed. During reprieve processing the system also ensures that all other tasks which belong to the job are excluded from execution when a job step is multitasked. The ENDRPV request ends that exclusion.

Format:

Location	Result	Operand
	ENDRPV	

IOAREA - CONTROL USER ACCESS TO I/O AREA

The IOAREA request instructs the system to allow or deny access to the user's I/O and Dataset Parameter Table (DSP) areas. The DSP is described under this section of this manual.

The IOAREA request also restores the status of the I/O and DSP areas to their initial status. Initially, the user I/O area is assumed to be unlocked.

Format:

Location	Result	Operand
	IOAREA	*key,save*

key Equals any of the following:

LOCK Denies access to the user's I/O buffers and DSP area. The limit address is set to the address specified in JCDSP. (All user logical I/O calls that require access to the DSP area or I/O buffers involve an exchange to the operating system before and after I/O processing.)

UNLOCK Gives full access to the user's I/O buffers and DSP area. The limit address is set to the value specified in JCFL.

RESTORE Reserved for use by the FORTRAN library. If UNLOCK was used previously to unlock the I/O area, RESTORE locks the area.

save Symbolic address where lock status is to be stored; required only if RESTORE is used. The current status of *key* is stored in one word.

JTIME - REQUEST ACCUMULATED CPU TIME FOR JOB

The accumulated CPU time for the job is returned at the location specified in the macro call. The time in seconds is expressed in floating-point form.

Format:

Location	Result	Operand
	JTIME	*address*

address A symbol or an A, S, or T register containing the address where the accumulated CPU time is returned

MEMORY - REQUEST MEMORY

The MEMORY macro determines or changes a job's memory allocation and/or mode of field length reduction.

Format:

Location	Result	Operand
	MEMORY	*code,value*

code *code* can be one of the following:

UC *value* specifies the number of words to be added to (if *value* is positive) or subtracted from (if *value* is negative) the end of the user code/data area.

Memory is added to or deleted from the end of the user code/data area by using the UC code. If the user code/data area is expanded, the new memory is initialized to an installation-defined value.

FL *value* specifies the number of words to which the job's field length is to be changed. If FL is specified and *value* is not, the new field length is set to the maximum allowed the job.

The job's field length is changed by using the FL code. The field length is set to the larger of the requested amount rounded up to the nearest multiple of 512 decimal words or the smallest multiple of 512 decimal words large enough to contain the user code/data, LFT, DSP and buffer areas. The job is placed in user-managed field length reduction for the duration of the job step.

USER The job is put in user-managed field length reduction mode. *value* is ignored. When USER code is specified, the job is placed in user mode until a subsequent request is made to return it to automatic mode.

AUTO The job is put in automatic field length reduction mode. *value* is ignored. When AUTO code is specified, the job is placed in automatic mode and the field length is reduced to the smallest multiple of 512 decimal words that contain the user code/data, LFT, DSP, and buffer areas.

MAXFL The maximum field length allowed the job is determined and returned in *value*.

The job is aborted if honoring the request results in a field length greater than the maximum allowed the job. The maximum is the smaller of the total number of words available to user jobs minus the job's JTA or the amount determined by the MFL parameter on the JOB statement.

CURFL The current field length is determined and returned in *value*.

TOTAL The total amount of unused space in the job is determined and returned in *value*.

value A value or an A, S, or T register containing a value when *code* is UC or FL. An A, S, or T register that contains a returned value when *code* is CURFL, MAXFL, or TOTAL.

Examples:

Location	Result	Operand	Comment
1	10	20	35
	MEMORY	FL	

The job's field length is set to the maximum allowed the job and the job is placed in user mode for the duration of the job step.

Location	Result	Operand	Comment
1	10	20	35
	MEMORY	AUTO	

The job's field length is reduced to a minimum and the job is placed in automatic mode.

Location	Result	Operand	Comment
1	10	20	35
	MEMORY	UC,-5	
	or		
	MEMORY	UC,S5	where (S5) = -5

The job's user code/data area is reduced by five words.

MESSAGE - ENTER MESSAGE IN LOGFILE

The printable ASCII message at the location specified in the macro call is entered in the job and system logfile. The message must be 1 through 80 characters, terminated by a zero byte.

Format:

Location	Result	Operand
	MESSAGE	*address,loc,msgclass,override*

address A symbol or an A, S, or T register (except A0, S0, and S2) containing the starting address of the ASCII message

loc Destination for message. Can be any of the following:

U User logfile only
S System logfile only
US User and system logfiles; default if *loc* is blank.

loc can be a symbol or an A, S, or T register (except A0, S0, S3, or S4).

msgclass Class where the message is to be assigned. Only current class is JCLMSG. *msgclass* can be a symbol or an A, S, or T register (except A0, S0, S2, S3, or S4) containing the message class.

override Message suppression override flag; if present, message is to go to $LOG regardless of ECHO status. All messages destined for system logfile are written to system log regardless of ECHO status.

MODE - SET OPERATING MODE

The MODE macro allows the user to set or clear mode flags in the job's exchange package.

Format:

Location	Result	Operand
	MODE	FI=*option*,BT=*option*

Parameters are in keyword form. At least one parameter must be specified. The parameters are:

FI=*option* Floating interrupt mode. *option* can be:

ENABLE Enables floating-point error interrupts
DISABLE Disables floating-point error interrupts

The default enables floating-point error interrupts. If FI=DISABLE is specified, floating-point errors are ignored.

BT=*option* Bidirectional transfer mode. *option* can be:

ENABLE Enables bidirectional memory transfers
DISABLE Disables bidirectional memory transfers

The default enables bidirectional memory transfers. If BT=DISABLE is specified, block reads and writes are not performed concurrently.

The BT parameter is operational on CRAY X-MP Computer Systems only.

NORERUN - CONTROL DETECTION OF NONRERUNNABLE FUNCTIONS

The NORERUN request instructs the system to begin or cease monitoring user operations for nonrerunnable functions. This request determines whether execution of such functions makes the job not rerunnable without affecting the current rerunnability of the job. The default value is determined by the installation.

Format:

Location	Result	Operand
	NORERUN	*parameter*

parameter A symbol identifying a location or an A, S, or T register containing the address of a location containing either a 0 for ENABLE or a 1 for DISABLE.

ENABLE Causes the system to begin (or continue) monitoring user functions for nonrerunnable operations

DISABLE Causes the system to stop monitoring user operations for nonrerunnable functions

A NORERUN statement does not affect the existing rerunnability of the job. The functions that make a job not rerunnable include:

- Writing to a permanent dataset,
- Saving, deleting, adjusting, or modifying a permanent dataset, and
- Acquiring a dataset from a front-end system.

RECALL - RECALL JOB UPON I/O REQUEST COMPLETION

The RECALL macro removes a job from processing. The job does not become a candidate for processing until the previously issued I/O request for

the specified dataset is completed or partially completed; that is, the job is resumed when another physical request is completed, which may be more then one block of data.

Format:

Location	Result	Operand
	RECALL	*address*

address Symbolic address of the Open Dataset Name Table (ODN) or Dataset Parameter Table (DSP) for this dataset or an A, S, or T register containing the ODN or DSP address. See description of OPEN macro under Dataset Management later in this section.

RERUN - UNCONDITIONALLY SET JOB RERUNNABILITY

The RERUN request instructs the system to mark the job as either rerunnable or not rerunnable regardless of functions previously performed. The future monitoring of nonrerunnable functions is not affected.

Format:

Location	Result	Operand
	RERUN	*parameter*

parameter A symbol identifying a location or an A, S, or T register containing the address of a location that contains either a 0 for ENABLE or a 1 for DISABLE.

ENABLE Causes the system to mark the job rerunnable

DISABLE Causes the system to mark the job not rerunnable

ROLL - ROLL A JOB

A user protects a job against system interruption through the ROLL request. Rolling a job causes it to be written to disk so that the job then can be recovered in the event of a system interruption. Once a job has been rolled, it remains recoverable unless it loses the recoverable status. The following conditions can cause loss of recoverability:

- Random writing to any dataset
- Saving, deleting, adjusting, or modifying a permanent dataset
- Initial writing to a dataset
- Sequential writing to a dataset after any positioning operation, such as REWIND OR BKSP
- Releasing any nonpermanent dataset

Once a job loses its recoverable status, the user can request another ROLL to continue to protect the job against system interruption.

Format:

Location	Result	Operand
	ROLL	

SETRPV - SET JOB STEP REPRIEVE

The SETRPV request enables the user's current job step to maintain control when a job step abort error condition occurs or upon normal termination of the job step. Once enabled by the user, reprieve processing remains in effect until the job step terminates, a selected error condition occurs, or the user disables reprieve processing.

If a selected error condition occurs, the user is reprieved from the normal or abnormal job step termination. The reprieve processing code that is given control attempts a recovery or continues with the normal or abort termination.

I/O errors from $SYSLIB or $IOLIB are not readily recognizable or correctable. At the $IOLIB level, I/O usually involves three steps: initialization, transfer, and termination. I/O errors almost always occur at the transfer stage. Because termination does not occur in this case, any further attempts at initialization fail, thus hampering correction. Any errors reported by the logical I/O routines look like user-requested aborts.

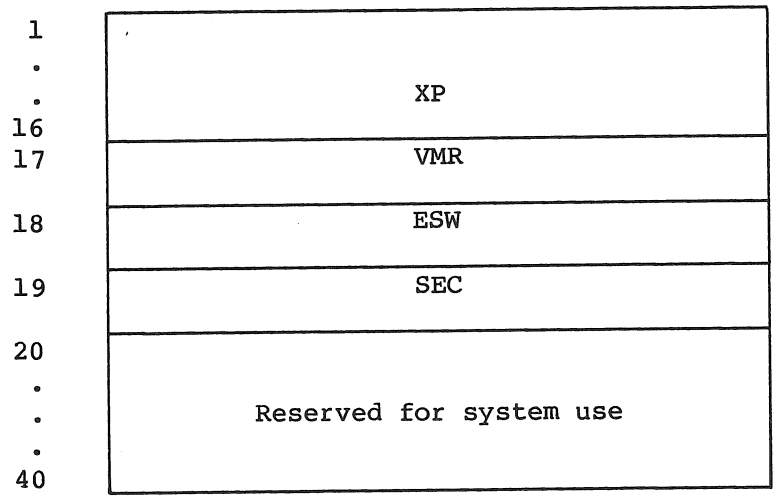
Two types of error conditions are related to a job step: nonfatal and fatal. Nonfatal error conditions can be reprieved any number of times per job step by the user. Each fatal error condition is reprieved only once per job step. The second occurrence of any fatal error condition results in an immediate termination of the job step.

Format:

Location	Result	Operand
	SETRPV	*entry,xpsave,mask*

entry Address where control is passed if an error condition for which reprieve processing is selected occurs

xpsave First word address (FWA) of the area where the system copies the user's exchange package when control is passed to the user's reprieve processing code. This area is formatted as follows, and the contents are those at the time of the error.



XP User exchange package

VMR User Vector Mask register

ESW Error status word. Contains the octal value of the error category reprieved, or 0 if normal termination.

SEC Actual system error code, if any

mask A user-specified octal value indicating the classes of conditions to enable reprieve processing. Any number of classes are specified by combining the appropriate octal *mask* values.

Class (Octal mask value)	Reprievable Condition
0	Disable user reprieve processing
1	Normal job step termination
2	User-requested abort
4	System abort
10	Operator DROP
20	Operator RERUN
40	Memory error
100	Floating-point error
200	Time limit
400	Mass storage limit exceeded
1000	Memory limit exceeded
2000	Link transfer error
4000	Security violation
10000	Interactive console 'attention' interrupt

NOTE

The system disables reprieve processing once the user's reprieve processing code gains control. To be reprieved from future error conditions, the user must issue another SETRPV request. The user cannot issue a SETRPV request for second occurrences of errors defined as fatal.

SWITCH - SET OR CLEAR SENSE SWITCH

The SWITCH macro allows a user to turn on (set) or turn off (clear) pseudo sense switches.

Format:

Location	Result	Operand
	SWITCH	*n*,*x*

n Number of switch (1 through 6) to be set or cleared

x Switch position. *x* can be:

ON Switch *n* is turned on; set to 1.
OFF Switch *n* is turned off; set to 0.

DATASET MANAGEMENT MACROS

The system action request macros involved with dataset management allow the user to open datasets, set up tables, and close, release, or dispose datasets. System action request macros available include CLOSE, DISPOSE, DSP, OPEN, RELEASE, and SUBMIT.

CLOSE - CLOSE DATASET

CLOSE releases the buffer and the Dataset Parameter Table (DSP) for a COS-managed dataset. Disk space is not released and the dataset remains attached to the job.

The buffers are flushed if all of the following conditions are true for the dataset:

- The dataset is currently opened for output.
- No end-of-data is written.
- The dataset is being written sequentially.
- The dataset's DSP is managed by COS.
- It has COS blocked dataset structure.
- It is not memory resident.

Format:

Location	Result	Operand
	CLOSE	*dn*

dn Dataset name. Symbolic address of the Open Dataset Name Table (ODN) for this dataset or an A, S, or T (not A0 or S0) register containing the address of the ODN. See description of OPEN macro.

DISPOSE - DISPOSE DATASET

The DISPOSE macro places a dataset in the appropriate queue as defined by the PDD macro.

Format:

Location	Result	Operand
	DISPOSE	*pddtag*

pddtag Address of PDD macro call

DSP - CREATE DATASET PARAMETER TABLE

The DSP macro creates a table in the user field called the Dataset Parameter Table (DSP). This table holds information concerning the status of the named dataset and the location of the I/O buffer for the dataset. The DSP is illustrated in CRAY-OS Version 1 Reference Manual, publication SR-0011.

The DSP macro is used only when the user needs the DSP and I/O buffer in the user-managed memory portion of the job. Normally, a DSP and buffer for a dataset are created in the upper end of the job's memory (above JCHLM) by execution of an OPEN macro.

When using the DSP macro, the user must also set up a 2-word Open Dataset Name Table (ODN). This ODN must be defined before using an OPEN macro specifying this dataset.

The DSP macro is not executable; it merely sets up a DSP table with the dataset name, first, in, out, and limit fields initialized. An OPEN macro must be executed to make the DSP known to the system.

Format:

Location	Result	Operand
loc	DSP	*dn,first,nb*

loc Symbolic address of DSP. If *loc* is not specified, the address of the dataset name is generated.

dn Dataset name

first Address of the first word of the user-allocated buffer for this dataset

nb Number of 512-word blocks in the dataset buffer

Example:

Location	Result	Operand	Comment
1	10	20	35
x	DSP	XFIL,BUF,1	
ODN	CON	'XFIL'L	ASCII name
	CON	XFIL@	Address of DSP
	.		
	.		
	.		
	OPEN	ODN,T	

OPEN - OPEN DATASET

The OPEN macro prepares a dataset for processing. When an OPEN macro is executed, the dataset is made known to the system if it is not an existing dataset. I/O tables are created in the upper end of the job's memory, included are the Dataset Parameter Table (DSP) and the Logical File Table (LFT). An I/O buffer is created if the dataset is COS blocked format, but not for an unblocked dataset. The address or offset of the DSP table is returned to the user.

An OPEN macro can be executed on a dataset that is already open.

Format:

Location	Result	Operand
	OPEN	*dn,pd,ldt,u*

dn Dataset name. The OPEN macro generates a 2-word Open Dataset Name Table (ODN) the first time an OPEN of the dataset is encountered, unless the user previously generated an ODN for the dataset. The ODN is illustrated in CRAY-OS Version 1 Reference Manual, publication SR-0011. The *dn* becomes the symbolic address of the ODN and is used in all references to the dataset in other I/O requests.

As an alternative, *dn* can be an A, S, or T register (not A0, S0, or S2) containing the ODN address.

pd Processing direction. Can be any of the following:

I	Dataset opened for input
O	Dataset opened for output
IO	Dataset opened for input/output (default)

pd can alternatively be an S or T register (but not an A register) with bit 0 set for input and/or bit 1 set for output.

ldt Label Definition Table (LDT); an optional parameter that is the name of a previously defined LDT for tape processing. The pointer to this field is placed in the ODN built by the macro. The parameter applies to tape datasets only.

u Unblocked. If the *u* parameter is specified, the DSP has DPUDS set and no buffer is allocated.

The *u* parameter is used only as a keyword; no registers are allowed.

If the DSP pointer in the ODN is negative or 0, the OPEN call returns the negative DSP offset in the DSP field of the ODN. The actual DSP address is equal to (JCDSP) - negative DSP offset, where (JCDSP) is the value of the JCDSP field of the Job Communication Block. The negative DSP offset of a dataset does not change when a job's field length changes or as additional datasets are opened or closed.

If the DSP pointer in the ODN is positive and greater than 0, OPEN assumes the DSP field contains the address of the user's own DSP in the user field between the Job Communication Block and (JCHLM) (the value in the JCHLM field of the JCB). The system uses the DSP indicated and does not allocate an additional DSP or buffer in the job's I/O table area. The DSP indicated must already contain the buffer pointers and must indicate a buffer also within the user field. If the dataset is memory resident, this buffer should be large enough to contain the entire dataset plus one block.

Examples:

1. In the following example, OPEN generates an ODN for dataset DSETONE unless one has been previously generated for that dataset. The dataset is opened for input/output processing.

Location	Result	Operand	Comment
1	10	20	35
L	OPEN	DSETONE,IO	

2. In this example, the address of the ODN generated by this OPEN call is passed through register S1; S2 contains processing direction information.

Location	Result	Operand	Comment
1	10	20	35
	OPEN	S1,S2	

3. In this example, the dataset ATAPE is opened for output with LABELX as the Label Definition Table. An ODN for ATAPE has not yet been defined.

Location	Result	Operand	Comment
1	10	20	35
	OPEN	ATAPE,O,LABELX	

RELEASE - RELEASE DATASET TO SYSTEM

The dataset whose Dataset Parameter Table (DSP) address is at the location specified in the macro call is returned to the system. The dataset is closed and the Dataset Name Table (DNT) entry is released. Additional system action depends on the type of dataset. Output datasets are routed to a front end. If a dataset is not a permanent dataset, the disk space associated with that dataset is returned to the system. The dataset is no longer attached to the job.

Format:

Location	Result	Operand
	RELEASE	*address*,HOLD

address Symbolic address of the Open Dataset Name Table (ODN) or Dataset Parameter Table (DSP) for this dataset or an A, S, or T register containing the ODN or DSP address. See description of OPEN and DSP macros in this section.

HOLD Hold generic device; optional parameter. If specified, the generic system resource associated with this dataset is not made available to another job when the dataset is released.

SUBMIT - SUBMIT JOB DATASET

The SUBMIT macro places a job dataset into the Cray system job input queue.

Format:

Location	Result	Operand
	SUBMIT	*pddtag*

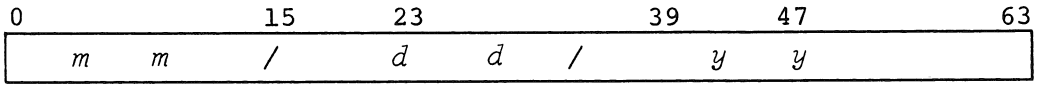
pddtag Address of PDD

TIME AND DATE REQUEST MACROS

Several system action request macros inform the user of the current time or date and the Julian date. These include DATE, DTTS, JDATE, MTTS, TIME, TSDT, and TSMT.

DATE - GET CURRENT DATE

The current date in ASCII is returned at the location specified in the macro call. The format of the date is as follows:



The order can be changed to day, month, and year (the European format) through an installation parameter.

Format:

Location	Result	Operand
	DATE	*address*

address A symbol or an A, S, or T register containing the destination address of the current date

DTTS - DATE AND TIME TO TIMESTAMP CONVERSION

The ASCII date and time are converted into the corresponding system timestamp.

Registers S1 and S2 on entry hold the ASCII date and time as:

	0 63
S1 ASCII date	*m m / d d / y y*
S2 ASCII time	*h h : m m : s s*

Format:

Location	Result	Operand
	DTTS	

On exit from the macro, S1 is the timestamp corresponding to the requested date/time.

JDATE - RETURN JULIAN DATE

The current Julian date in ASCII is returned at the location specified in the macro call. The format of the date is as follows:

0	15	39	63
y y	*d d d*		

Five ASCII characters are left-justified with blank fill in the reply word. The first two characters are the year; the next three are the number of the day in the year.

Format:

Location	Result	Operand
	JDATE	*address*

address A symbol or an A, S, or T register containing the destination address of the current Julian date

MTTS - MACHINE TIME TO TIMESTAMP CONVERSION

A machine time (RT register value) is converted into the corresponding system timestamp (a 1-word encoding of the date and time). Register S1 contains the machine time on entry.

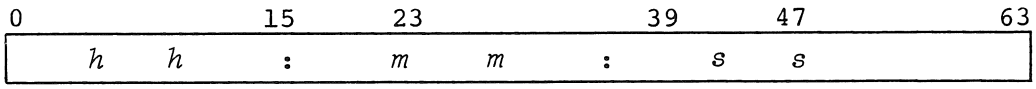
Format:

Location	Result	Operand
	MTTS	

S1 contains the timestamp on exit from the macro.

TIME - GET CURRENT TIME

The current time in ASCII is returned at the location specified in the macro call. The format of the time is as follows:



Format:

Location	Result	Operand
	TIME	*address*

address A symbol or an A, S, or T register containing the destination address of the current time

TSDT - TIMESTAMP TO DATE AND TIME CONVERSION

A timestamp (1-word encoding of a date and time) is converted to the corresponding date and time expressed in ASCII.

Register S1 on entry holds the timestamp. On exit, registers are set as follows:

	0 63
S1 ASCII date	*m m / d d / y y*
S2 ASCII time	*h h : m m : s s*
S3 ASCII fractional seconds	*. s s s s*

Format:

Location	Result	Operand
	TSDT	

TSMT - TIMESTAMP TO MACHINE TIME CONVERSION

A timestamp (1-word date/time encoding) is converted to the corresponding machine time (RT clock value). Register S1 on entry contains the timestamp.

Format:

Location	Result	Operand
	TSMT	

S1 contains the machine time on exit from the macro.

DEBUGGING AID MACROS

The system action request macros in this category permit the user to selectively read or write information during a program run to aid in the debugging process. Included are the DUMP, FREAD, FWRITE, INPUT, LOADREGS, OUTPUT, SAVEREGS, SNAP, UFREAD, and UFWRITE macros. Using the label DEBUG allows conditional execution of the macros DUMP, FREAD, FWRITE, INPUT, OUTPUT, and SNAP.

DUMP - DUMP SELECTED AREAS OF MEMORY

The DUMP macro performs a formatted dump of selected memory areas.

The macro generates exactly four words of inline code; the rest of the logic is in a subroutine created by the macro and loaded into the user area.

The DEBUG option allows conditional execution of the DUMP macro. If the label on the DUMP statement is DEBUG, no label is defined for the generated code. Instead, unless the symbol DEBUG has been set to 1 by a previously assembled SET or equate statement, code generation within the macro is suppressed entirely.

Format:

Location	Result	Operand
	DUMP	(*list*),UNIT=*unit*

list A list of memory ranges separated by commas. The list need not be enclosed in parentheses if it contains only one range. No limit is placed on the number of ranges in the list. Within the list, null elements are ignored, so that each memory range is preceded and followed by blanks. However, a memory range cannot contain embedded blanks. Each non-null range must have one of the following forms:

f..*l*	Dump memory from address *f* to address *l*-1
f	Dump memory word *f*
f(*n*)	Dump *n* words starting at memory address *f*

f, *l*, or *n* can be numbers, labels, register names, or a combination of labels and numbers. Indirect addressing, using the at sign (@) as a prefix, is allowed. For numbers, the default radix is decimal unless a BASE O or BASE M is in effect.

Examples:

(O'200..O'400)	Words 200_8 through 377_8
(0(D'128))	Words 0 through 177_8 (the Job Communication Block)

(R.A1(R.A2))	The starting address is given in A1; the word count is given in A2.
(@R.A1(@R.A2))	The starting address is given in the memory word addressed by A1; the word count is in the memory word addressed by A2.
(R.A1..R.A2)	The address given in A1 through the address immediately before the address given in A2
(@R.A1..@R.A2)	The address given in the memory word addressed by A1 through the address immediately before the address given in the memory word addressed by A2
(TABLE(R.A.BU))	The first *n* words of TABLE, where *n* is held in register A.BU
(TTT-1(@R.B77))	The first *n* words following and including TTT-1, where *n* is held in the memory addressed by register B77
(@PTR(@LTH))	The word addressed by PTR is the start, and the word count is in the word addressed by LTH
(@P..@Q,@A(@L))	Two ranges are dumped. The first range is from the word addressed by P through the word immediately before the word addressed by Q; the second begins at the word addressed by A and includes the number of words given by the value contained in the memory cell addressed by L. Only the low-order 24 bits in P, Q, A, and L are considered in determining the addresses.

UNIT=*unit* A local dataset name, an expression containing only previously defined terms that resolves into a FORTRAN unit number, or the previously defined label of a word containing either a local dataset name or a FORTRAN unit number. The default is $OUT.

Return conditions:

All registers are saved and restored, including the vector registers and VL.

FREAD - READ DATA

The FREAD macro permits a FORTRAN-like read statement that can make use of a previously defined format.

Format:

Location	Result	Operand
	FREAD	*fmt*,(*list*),SV=$\left\{\begin{matrix}\text{YES}\\\text{NO}\end{matrix}\right\}$,UNIT=*unit*, END=*addr*,ERR=*addr*

fmt Format; takes one of the following forms:

faddr Address of a format, possibly defined with the DATA pseudo instruction, as in:
```
DATA     '(F10.0)'
```

((*string*)) A character string enclosed in a double set of parentheses

The default is (5O25).

(*list*) List of addresses for which values are to be read. Even with only one item, the list must be enclosed in parentheses. Each item in the list specifies either the address of a single word or the address of an array.

An array is handled by enclosing the array base address, the word count, and an optional increment in an additional set of parentheses. Examples: ((A,10)) or ((B,LTH,3))

The CAL statement

```
        FREAD    ,((A,10),(B,LTH,3))
```

is equivalent to the FORTRAN statements

```
        READ 20, (A(I), I=1,10), (B(I),I=1,3*LTH,3)
   20   FORMAT (5O25)
```

An array or a single word can be addressed indirectly by using the at sign (@) and the name of a variable containing the indirect address instead of an array name. For example:

((@C,10)) Reads values for the first 10 words of an array beginning at an address held in variable C

((@E,1)) Reads a value for the single word specified by the address held in variable E

To pass a numeric address, use a W prefix (for example, W.177).

SV=$\begin{Bmatrix}YES\\NO\end{Bmatrix}$ Save flag. If SV=NO, the SAVEREGS and LOADREGS macros are not invoked. If SV=YES, all registers are saved and restored. The default is SV=NO.

UNIT=*unit* A local dataset name, an expression containing only previously defined terms that resolves into a FORTRAN unit number, or the previously defined label of a word containing either a local dataset name or a FORTRAN unit number. The default is $IN.

END=*addr* Optional address where a branch occurs if an EOF is encountered

ERR=*addr* Optional address where a branch occurs if an error is encountered during the read

FWRITE - WRITE DATA

The FWRITE macro permits a FORTRAN-like write statement that can make use of a previously defined format.

Format:

Location	Result	Operand
	FWRITE	*fmt*,(*list*),SV=$\begin{Bmatrix}YES\\NO\end{Bmatrix}$,UNIT=*unit*

fmt Format; takes one of the following forms:

faddr Address of a format, possibly defined with the DATA pseudo instruction, as in:
```
DATA     '(F10.0,''TEXT'')'
```

((*string*))
A character string enclosed in a double set of parentheses (for example, ((F10.0,''TEXT'')))

The default is (5025).

(*list*) List of addresses whose contents are to be written. Even with only one item, the list must be enclosed in parentheses. Each item in the list specifies either the address of a single word or the address of an array.

An array is handled by enclosing the array base address, the word count, and an optional increment in an additional set of parentheses. Examples: ((A,10)) or ((B,LTH,3))

The CAL statement

```
      FWRITE    ,((A,10),(B,LTH,3))
```

is equivalent to the FORTRAN statements

```
      PRINT 20, (A(I), I=1,10), (B(3*(I-1)+1), I=1, LTH)
   20 FORMAT (5O25)
```

An array or a single word can be addressed indirectly by using the at sign (@) and the name of a variable containing the indirect address instead of an array name. For example:

((@C,10)) Reads values for the first 10 words of an array beginning at an address held in variable C

((@E,1)) Reads a value for the single word specified by the address held in variable E

To pass a numeric address, use a W prefix (for example, W.177).

SV=$\begin{Bmatrix}\text{YES}\\ \text{NO}\end{Bmatrix}$ Save flag. If SV=NO, the SAVEREGS and LOADREGS macros are not invoked. If SV=YES, all registers are saved and restored. The default is SV=NO.

UNIT=*unit* A local dataset name, an expression containing only previously defined terms that resolves into a FORTRAN unit number, or the previously defined label of a word containing either a local dataset name or a FORTRAN unit number. The default is $OUT.

INPUT - READ DATA

The INPUT macro reads data resident on a dataset or characters already located in memory and assigns values to variables, words of an array, or registers. Its syntax is as close as possible to the syntax of the INPUT statement in SKOL.

The macro generates its code either inline or in a unique subroutine created by the macro. In the latter case, exactly three words of code are generated inline.

The DEBUG option allows conditional execution of the INPUT macro. If the label on the INPUT statement is DEBUG, no label is defined for the generated code. Instead, unless the symbol DEBUG has been set to 1 by a previously assembled SET or equate statement, code generation within the macro is suppressed entirely.

Format:

Location	Result	Operand
	INPUT	(*list*),SV=$\left\{\begin{matrix}\text{YES}\\\text{NO}\end{matrix}\right\}$,IN=$\left\{\begin{matrix}\text{YES}\\\text{NO}\end{matrix}\right\}$,UNIT=*unit*, STRING=*addr*,LTH=*length*,END=*addr*,ERR=*addr*

list A list of input elements, each of which can include a variable name, an array specifier, and a format item. The list need not be enclosed in parentheses if it contains only one element. If it consists of more than one element, the elements are separated by commas. Null elements are ignored, so that each list element is preceded and followed by blanks. However, an element cannot contain embedded blanks. Each non-null element must have one of the following forms:

:fmt A format item not associated with a variable, such as

:fmt	:2x
fmt	2x

:fmt	:/
fmt	/

var:fmt A variable name and the format used to read a value into it

The format can contain any of the edit descriptors available to the Cray FORTRAN (CFT) user. The format cannot contain commas unless the entire list item is enclosed in parentheses.

The variable can refer to a single word, to an array, to a single register, or to an array of registers, and can take any of the following forms:

addr Change the contents of a single word (for example, LABEL-2 or W.177).

addr(*count*) Read values for *count* words beginning at *addr*.

addr(*count*!*incr*)
Read values for *count* words beginning at *addr* and applying an increment of *incr* after each word. The default value for *incr* is 1.

R.*rn* Change the contents of register *rn* (where *r* is A, B, S, or T and *n* is an octal register number or a register designator of the form .*name*.).

R.VL or R.VM
Change the current vector length or vector mask.

R.*rn*(*count*) Change *count* registers starting with *rn*, as in R.A1(5).

R.V*n*(*count*) Change the first *count* elements of V*n*.

R.V*n*+*e* Change the *e*th element in V*n*.

R.V*n*+*e*(*count*)
Change *count* elements, beginning at the *e*th element in V*n*.

In all of the above, *n* must be either an octal number or a previously defined register designator. *count* and *e* are represented by any absolute expression, where the default radix is determined by the calling program. The variable can also refer indirectly to a word or to an array, using a saved register or a word in memory as a pointer. The forms begin with the at sign (@) and include:

@*addr* Modify the word addressed by *addr*.

@*addr*(*count*)
Modify *count* words beginning with the word addressed by *addr*.

@*addr*(*count*!*incr*)
Modify *count* words beginning with the word addressed by *addr*, applying an increment of *incr* after each word.

@R.*rn* Modify the word addressed by register *rn*.

@R.*rn*(*count*)
Modify *count* words beginning with the word addressed by register *rn*.

SV={YES / NO} Save flag. If SV=NO, the SAVEREGS and LOADREGS macros are not invoked, and registers cannot be used for input values; IN=YES must also be specified when SV=NO. The default is SV=YES, which saves and restores all registers.

IN={YES / NO} Inline Code flag. If IN=YES, all the code necessary to perform the INPUT (except the standard subroutines called by the SAVEREGS and LOADREGS macros) is generated inline. The default is IN=NO, which causes three words of code to be generated inline; the rest is contained in a subroutine created by the macro.

UNIT=*unit* A local dataset name, an expression containing only previously defined terms that resolves into a FORTRAN unit number, or the previously defined label of a word containing either a local dataset name or a FORTRAN unit number. The default is $IN.

STRING=*string*
Address of a packed character string that resides in memory. When used in conjunction with the LTH parameter, the STRING parameter allows input (decoding) from the string. The END and ERR parameters cannot be used with STRING and LTH.

LTH=*length*
Number of characters to be decoded from *string*

END=*addr* Optional address where a branch occurs if an end-of-file (EOF) is encountered

ERR=*addr* Optional address where a branch occurs if an error is encountered during the read

Return conditions:

All registers, including the vector registers and the Vector Length register, are saved and restored when SV=YES (the default).

LOADREGS - RESTORE ALL REGISTERS

The LOADREGS macro restores the A, B, S, T, V, VL, and VM registers that were saved by a previously executed SAVEREGS macro.

Format:

Location	Result	Operand
	LOADREGS	[*region*],INLINE=$\left\{\begin{matrix}\text{YES}\\\text{NO}\end{matrix}\right\}$

region The *region* used previously in a corresponding SAVEREGS. If no value is specified, the default is QZH44HZQ. If the region is defined, it must be an area containing O'1230 words; if it is not defined, the LOADREGS macro defines it. If LOADREGS requests are nested, each request must specify a different *region*.

INLINE=$\left\{\begin{matrix}\text{YES}\\\text{NO}\end{matrix}\right\}$

Inline Code flag. If INLINE is omitted (as when LOADREGS is called), A0 and B0 are restored from words O'1200 and O'1000 of the region. If INLINE=YES, both A0 and B0 are lost. If INLINE=NO, B0 is restored from O'1223, but A0 is lost.

OUTPUT - WRITE DATA

The OUTPUT macro transfers variable values and character strings from a user's data area to a dataset or to an area in memory. Its syntax is as close as possible to the syntax of the OUTPUT statement in SKOL.

The macro generates its code either inline or in a unique subroutine created by the macro. In the latter case, exactly three words of code are generated inline.

The DEBUG option allows conditional execution of the OUTPUT macro. If the label on the OUTPUT statement is DEBUG, no label is defined for the generated code. Instead, unless the symbol DEBUG has been set to 1 by a previously assembled SET or equate statement, code generation within the macro is suppressed entirely.

Format:

Location	Result	Operand
	OUTPUT	(*list*),SV={YES/NO},IN={YES/NO},UNIT=*unit*, BUFFER=*addr*,LTH=*length*

list A list of variable names, array names, format items, and string constants separated by commas. The list need not be enclosed in parentheses if it contains only one element. If it consists of more than one element, the elements are separated by commas. Null elements are ignored, so that each element is preceded and followed by blanks. However, an element cannot contain embedded blanks unless it is enclosed in a second level of parentheses. Each non-null element must have one of the following forms:

'*string*' or **string**
: Any character string. The list item must be enclosed in parentheses if the string contains any blanks or commas. If the string is delimited by apostrophes, any inner apostrophes must be doubled. If it is delimited by asterisks, no inner asterisks are allowed.

:*fmt*
: Format item that is not associated with any variable; for example,

:*fmt*	:2x
fmt	2x

:*fmt*	:/
fmt	/

The list item must be enclosed in parentheses if *fmt* contains any commas or blanks.

$PAGE, $SKIP, and $LINE
: These special format items do not require a colon prefix. They generate FORTRAN-style carriage control characters at the beginning of a line. When $SKIP or $PAGE is the first list element, the appropriate literal character (0 or 1) becomes the first element of the OUTPUT format. $LINE is assumed to be present by default unless the first list element is a format item (:*fmt*). If $LINE, $SKIP, or $PAGE occurs later in the list, a comma and a slash are inserted before the carriage control literal to force a new line.

var:*fmt* Variable name and the format to be used for its output

var::*fmt* The same as *var*:*fmt*, except that the variable's name and value are output together

var The same as *var*::022

var(...) The same as *var*(...)::(4025)

The variable can refer to a single word, to an array, to a single register, or to an array of registers and can take any of the following forms:

addr Write the contents of a single word (for example, LABEL-2 or W.177).

addr(*count*) Write *count* words beginning at *addr*.

addr(*count*!*incr*)
Write *count* words beginning at *addr* and applying an increment of *incr* after each word. The default value for *incr* is 1.

R.*rn* Write the contents of register *rn* (where *r* is A, B, S, or T and *n* is an octal register number or a register designator of the form .*name*).

R.VL or R.VM
Write the current vector length or vector mask.

R.*rn*(*count*) Write *count* registers starting with *rn*, as in R.A1(5).

R.V*n*(*count*) Write the first *count* elements of V*n*.

R.V*n*+*e* Write the *e*th element in V*n*.

R.V*n*+*e*(*count*)
Write *count* elements, beginning at the *e*th element in V*n*.

In all of the above, *n* must be either an octal number or a previously defined register designator. *count* and *e* can be represented by any absolute expression.

The variable can also refer indirectly to a word or to an array, using a saved register or a word in memory as a pointer. The forms begin with the at sign (@) and include:

@*addr* Write the word addressed by *addr*.

@*addr*(*count*)
Write *count* words beginning with the word addressed by *addr*.

@*addr*(*count!incr*)
Write *count* words beginning with the word addressed by *addr*, applying an increment of *incr* after each word.

@R.*rn* Write the word addressed by register *rn*.

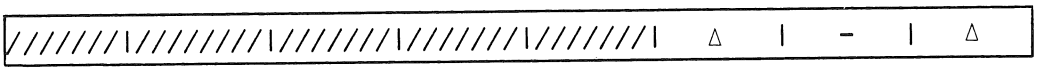
@R.*rn*(*count*)
Write *count* words beginning with the word addressed by register *rn*.

SV={YES / NO} Save flag. If SV=NO, the SAVEREGS and LOADREGS macros are not invoked, and registers cannot be used for output; IN=YES must also be specified if SV=NO. The default is SV=YES, which saves and restores all registers.

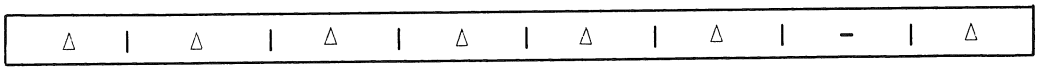
IN={YES / NO} Inline Code flag. If IN=YES, all the code necessary to perform the OUTPUT (except the standard subroutines called by the SAVEREGS and LOADREGS macros) is generated inline. The default is IN=NO, which means that exactly three words of code are generated inline; the rest is contained in a subroutine created by the macro.

UNIT=*unit* A local dataset name, an expression containing only previously defined terms that resolves into a FORTRAN unit number, or the previously defined label of a word containing either a local dataset name or a FORTRAN unit number. The default is $OUT.

UNIT=$LOG is treated as a special parameter value rather than as a dataset name. If UNIT=$LOG, the OUTPUT macro automatically encodes the data (using its own buffer) rather than writing it directly, and uses the MESSAGE macro to write it to both the user log and the system log. OUTPUT looks at the first eight characters of the formatted line. The content of the first eight characters of a message ID is:



where Δ is a space. If the first eight characters do not match the above, OUTPUT inserts the string:



BUFFER=*addr*
: Address of a packed character buffer used instead of an external dataset to accept the output

LTH=*length*
: Number of characters to be encoded (output) into the buffer

Return conditions:
: All registers, including the vector registers and the Vector Length register, are saved and restored when SV=YES (the default).

SAVEREGS - SAVE ALL REGISTERS

The SAVEREGS macro saves all of the A, B, S, T, V, VL, and VM registers. Additionally, it sets up words containing VL+1, P/4, parcel(P), B0/4, and parcel(B0) so that SNAP can handle the VL=VL+1 option and so that SNAP, DUMP, and OUTPUT can output P and B0 in parcel-address format. (Here, parcel(x) means the 2 low-order bits of x.)

Format:

Location	Result	Operand
	SAVEREGS	[*region*],INLINE={YES / NO}

region Label of the first word of a region where registers are to be saved. The default is QZH44HZQ. If the *region* is defined, it must be an area containing O'1230 words; if it is not defined, the SAVEREGS macro defines it. If SAVEREGS requests are nested, each request must specify a different *region*.

INLINE=$\left\{\begin{matrix}\text{YES}\\ \text{NO}\end{matrix}\right\}$

Inline Code flag. If INLINE is omitted (as when SAVEREGS is invoked directly), A0 is saved in word O'1200 and B0 is saved in word O'1000 of the region. If INLINE=YES, both A0 and B0 are lost. If INLINE=NO, B0 is saved in word O'1223 of the region and A0 is lost.

SNAP - TAKE SNAPSHOT OF SELECTED REGISTERS

The SNAP macro writes the contents of selected registers under the control of FORTRAN-style formats selected by the user.

The macro generates exactly three words of inline code; the rest of the logic is in a subroutine called by the macro.

The DEBUG option allows conditional execution of the SNAP macro. If the label on the SNAP statement is DEBUG, no label is defined for the generated code. Instead, unless the symbol DEBUG has been set to 1 by a previously assembled SET or equate statement, code generation within the macro is suppressed entirely.

Format:

Location	Result	Operand
	SNAP	(*list*),UNIT=*unit*,AF=*fmt*,BF=*fmt*, SF=*fmt*,TF=*fmt*,VF=*fmt*,VL=*n*

list A list of registers and register groups separated by commas. The list need not be enclosed in parentheses if it contains only one element. Within the list, null elements are ignored so that each element can be preceded and followed by blanks. However, an element cannot contain embedded blanks. Each element of the list that is not null must have one of the following forms:

R Writes the contents of all *R* registers (where *R* is A, B, S, T, or V)

R_i Writes the contents of register R_i (for example, A7)

R_{i-j} or R_i-R_j
Writes the contents of registers R_i through R_j (for example, A1-A4 or A1-4).

Each i or j must be either an octal number or a previously defined register designator (for example, B.SEP).

No limit is placed on the number of elements in the list or to the number of occurrences of a particular register. If the list is empty, no output is produced except for the usual header. The header, which is always produced, shows the contents of P and B0 as parcel addresses.

UNIT=*unit* A local dataset name, an expression containing only previously defined terms that resolves into a FORTRAN unit number, or the previously defined label of a word containing either a local dataset name or a FORTRAN unit number. The default is $OUT.

AF=*fmt* A register format in decimal; the default is (8(3X,O8)).

BF=*fmt* B register format in decimal; the default is (8(3X,O8)).

SF=*fmt* S register format in decimal; the default is (4O25).

TF=*fmt* T register format in decimal; the default is (4O25).

VF=*fmt* V register format in decimal; the default is (4O25)

VL=*n* Number of V register elements to be snapped. The default is VL=VL. The caller can also specify VL=VL+1 or an absolute expression. If VL is 0 or 64, then VL=VL+1 means 64 rather than 65. The default radix of n is decimal unless a BASE O or BASE M is in effect.

Return conditions:

All registers are saved, including the vector registers and VL.

UFREAD - UNFORMATTED READ

The UFREAD macro performs a FORTRAN-like unformatted read.

Format:

Location	Result	Operand
	UFREAD	*unit*,(*list*),SV=$\left\{\begin{matrix}\text{YES}\\\text{NO}\end{matrix}\right\}$,END=*addr*

unit A local dataset name, an expression containing only previously defined terms that resolves into a FORTRAN unit number, or the previously defined label of a word containing either a local dataset name or a FORTRAN unit number. There is no default.

(*list*) List of addresses for which values are read. Even with only one item, the list must be enclosed in parentheses. Each item in the list specifies either the address of a single word or the address of an array.

An array is handled by enclosing the array base address, the word count, and an optional increment in an additional set of parentheses. Examples: ((A,10)) or ((B,LTH,3))

The CAL statement

```
UFREAD   ,((A,10),(B,LTH,3))
```

is equivalent to the FORTRAN statement

```
READ (A(I), I=1,10), (B(3*(I-1)+1), I=1, LTH)
```

An array or a single word can be addressed indirectly by using the at sign (@) and the name of a variable containing the indirect address instead of an array name. For example:

((@C,10)) Reads values for the first 10 words of an array beginning at an address held in variable C

((@E,1)) Reads a value for the single word specified by the address held in variable E

To pass a numeric address, use a W prefix (for example, W.177).

SV=$\begin{Bmatrix}YES\\NO\end{Bmatrix}$ Save flag. If SV=NO, the SAVEREGS and LOADREGS macros are not invoked. If SV=YES, all registers are saved and restored. The default is SV=NO.

END=*addr* Optional address where a branch occurs if an error is encountered during the read

ERR=*addr* Optional address where a branch occurs if an error is encountered during the read

UFWRITE - UNFORMATTED WRITE

The UFWRITE macro performs a FORTRAN-like unformatted write of output items separated by commas.

Format:

Location	Result	Operand
	UFWRITE	*unit*,(*list*),SV=$\begin{Bmatrix}YES\\NO\end{Bmatrix}$

unit A local dataset name, an expression containing only previously defined terms that resolves into a FORTRAN unit number, or the previously defined label of a word containing either a local dataset name or a FORTRAN unit number. There is no default value.

(*list*) List of addresses whose contents are to be written. Even with only one item, the list must be enclosed in parentheses. Each item in the list specifies either the address of a single word or the address of an array.

An array is handled by enclosing the array base address, the word count, and an optional increment in an additional set of parentheses. Examples: ((A,10)) or ((B,LTH,3))

The CAL statement

```
UFWRITE     $OUT,((A,10),(B,LTH,3))
```

is equivalent to the FORTRAN statement

```
PRINT (A(I), I=1,10), (B(3*(I-1)+1), I=1, LTH)
```

An array or a single word can be addressed indirectly by using the at sign (@) and the name of a variable containing the indirect address instead of an array name. For example:

((@C,10)) Writes the first 10 words of an array beginning at an address held in variable C

((@E,1)) Writes the single word specified by the address held in variable E

To pass a numeric address, use a W prefix (for example, W.177).

SV={YES / NO} Save flag. If SV=NO, the SAVEREGS and LOADREGS macros are not invoked. If SV=YES, all registers are saved and restored. The default is SV=NO.

MISCELLANEOUS MACROS

Macros that do not fit in the other categories are the GETMODE, GETSWS, INSFUN, and SYSID macros.

GETMODE - GET MODE SETTING

The GETMODE macro obtains the mode setting from the user's exchange package and returns it in the S1 register.

Format:

Location	Result	Operand
	GETMODE	

GETSWS - GET SWITCH SETTING

The GETSWS macro allows the user to determine whether a specified sense switch number is set. GETSWS returns the setting of the switch number specified in the S1 register. (S1)=1 if set; 0 if not set.

Format:

Location	Result	Operand
	GETSWS	*n*

n Number of the switch (1 through 6) to be tested

INSFUN - CALL INSTALLATION-DEFINED SUBFUNCTION

The INSFUN macro allows the user to call any one of the installation-defined subfunctions defined in a subfunction table. Control is transferred to the indicated subfunction.

Format:

Location	Result	Operand
	INSFUN	*n,p*

n A symbol or an A, S, or T register (not A0 or S0) containing the subfunction code

p An optional symbol, A, S or T register (not S2), containing the address of a parameter list to be passed to the installation-dependent subfunction.

SYSID - REQUEST SYSTEM IDENTIFICATION

The identification of the current system is returned at the location specified in the macro call. The identification is returned as two words; the first contains the COS revision level in ASCII and the second contains the COS assembly date in ASCII. For example:

Format:

Location	Result	Operand
	SYSID	*address*

address A symbol or an A, S, or T register (not A0 or S0) containing the address where the system ID is returned

LOGICAL I/O MACROS 3

The logical I/O macros generate calls to I/O subroutines to be loaded from the subroutine library and executed as part of the user program. Datasets referenced by these macros must have been opened previously by an OPEN macro.

The four main categories of logical I/O macros are: synchronous read/write, asynchronous (buffered) read/write, unblocked read/write, and positioning.

SYNCHRONOUS READ/WRITE MACROS

The synchronous read/write logical I/O macros allow the user to read and write words or characters and to write an end-of-file (EOF) or an end-of-data (EOD). Control does not return to the user program until all requested data has been moved to or from the dataset buffer.

Upon termination of the Read/write function, register contents are modified as detailed under the description of each macro. A or S registers not specifically mentioned should not be assumed to have any meaningful contents, and will not contain the same values as before the function request. Registers B0, B70 through B77, and T70 through T77 can be changed, as well as VL, VM, V0, and V1. Other B, T, and V registers are not changed.

READ/READP - READ WORDS

The READ and READP macros transfer words of data that are resident on a dataset into the user's data area. Blank compression characters are not recognized, and any compressed blanks are not expanded (see CRAY-OS Version 1 Reference Manual, publication SR-0011).

The READ macro generates a return jump to the $RWDR subroutine, thus causing one record at a time to be processed. Each macro call causes the dataset to be positioned after the end-of-record (EOR) that terminated the read.

The READP macro generates a return jump to the $RWDP subroutine. Words are transmitted to the user's data area as requested by the user. Each call is terminated by reaching an EOR or by satisfying the word count, whichever comes first.

No blank decompression is performed.

When EOR is reached as a result of reading in word mode, the unused bit count from the end-of-record RCW is placed in the field DPBUBC of the Dataset Parameter Table (DSP). Also, the unused bits are zeroed in the user's record area.

Unrecovered data errors do not abort the job; instead, control is returned to the caller. The caller can use the good data read, (A2) through (A4)-1, and then abort. The caller can also skip or accept the bad data. If the caller does nothing, the job aborts when the next read request occurs. See the Library Reference Manual, CRI publication SR-0014, for detailed descriptions of SKIPBAD and ACPTBAD.

When a READ or READP macro refers to a memory-resident dataset, the first such macro causes the dataset to be loaded into the buffer from mass storage, if it exists there. If it does not exist on mass storage, the system I/O routines set the DSP so that it appears that the buffer is filled with data and no attempt is made to read data. Note that the I/O routines cannot distinguish between the cases (1) an existing dataset is declared memory resident, read in, modified in the buffer, rewound, and read again, and (2) no modification of data in the buffer occurs. In either case, the first read following a REWIND reads the unmodified data from disk. If an existing dataset is declared memory resident and is to be modified and reread, use backspace positioning macros rather than REWIND to reposition to beginning-of-data to preserve the modifications. This is necessary only when a memory-resident dataset already exists on mass storage.

Formats:

Location	Result	Operand
	READ	*dn,uda,ct*

Location	Result	Operand
	READP	*dn,uda,ct*

dn Dataset name (symbolic address of the Open Dataset Name Table (ODN) for this dataset) or an A, B, or S register (not A0 or S0) containing the Dataset Parameter Table (DSP) address or negative DSP offset relative to JCDSP

uda User data area first word address (FWA) or an A, B, or S register (not A1) containing the *uda* address

ct Word count or an A, B, or S register (not A1 or A2) containing the number of words to be read

Return conditions:

(A1) DSP address

(A2) FWA of user data area (*uda*)

(A3) Requested word count (*ct*)

(A4) Actual LWA+1 of data transferred to *uda*. (A4)=(A2) if a null record was read.

(S0) Condition of termination:

- < 0 EOR encountered
- = 0 Null record, EOF, EOD, or unrecovered data error encountered
- > 0 User-specified count (A3) exhausted before end-of-record RCW is encountered

(S1) Error status:

- = 0 No errors encountered
- = 1 Unrecovered data error encountered

(S6) Contents of the RCW if (S0)≤0 and (S1)=0; otherwise, meaningless. Note that for READ/READP, the unused bit count can also be obtained from S6 if (S0)<0.

READC/READCP - READ CHARACTERS

The READC and READCP macros transfer character data from a dataset into the user data area.

The READC macro generates a return jump to the $RCHR subroutine, thus causing one record at a time to be processed. Each macro call causes the dataset to be positioned after the EOR that terminated the read.

The READCP macro generates a return jump to the $RCHP subroutine. Characters are transferred to the user data area as requested by the user. Each call is terminated by reaching an EOR or by satisfying the character count, whichever occurs first.

One character from the record is placed, right-adjusted, zero-filled, in each word of the data area. Blank-compressed fields are recognized and expanded, one blank per word.

Unrecovered data errors do not abort the job. Instead, control is returned to the caller. The caller can use the good data read, (A2) through (A4)-1, and then abort. The user can also skip or accept the bad data. If the caller does nothing, the job aborts when the next read request occurs. See Library Reference Manual, CRI publication SR-0014, for detailed descriptions of SKIPBAD and ACPTBAD.

Memory-resident datasets are treated as described for READ/READP macro.

Formats:

Location	Result	Operand
	READC	*dn,uda,ct*

Location	Result	Operand
	READCP	*dn,uda,ct*

dn Dataset name (symbolic address of the Open Dataset Name Table (ODN) for this dataset) or an A, B, or S register (not A0 or S0) containing the Dataset Parameter Table (DSP) address or negative DSP offset

uda User data area first word address (FWA) or an A, B, or S register (not A1) containing the *uda* address

ct Character count or an A, B, or S register (not A1 or A2) containing the character count

Return conditions:

Same as for READ/READP, except that the requested count (A3) and data-transfer length (A4-A2) is in characters rather than words. Unused bits are not meaningful for READC/READCP since the unused characters are reflected in the number of characters transferred ((A4)-(A2)).

WRITE/WRITEP - WRITE WORDS

The WRITE macro generates a return jump to either the $WWDR or $WWDS subroutine, depending on whether an unused bit count is specified. Words are written from the user's data area. An end-of-record RCW is written following each WRITE. The end-of-record RCW indicates how many bits in the last words are unused, if any. No blank compression is performed.

The WRITEP macro generates a return jump to the $WWDP subroutine. Words are written from the user's data area as requested by the user. No EOR is written. No blank compression is performed.

If the dataset is memory resident and the WRITE or WRITEP causes the buffer to become full, the memory-resident flags are cleared and the buffers are flushed to mass storage.

To write only an end-of-record RCW, the WRITE macro with word count of 0 is used.

Formats:

Location	Result	Operand
	WRITE	*dn,uda,ct,ubc*
	WRITE	*dn,uda,ct*

Location	Result	Operand
	WRITEP	*dn,uda,ct*

dn Dataset name (symbolic address of the Open Dataset Name Table for this dataset) or an A, B, or S register (not A0 or S0) containing the Dataset Parameter Table (DSP) address or negative DSP offset

uda User data area first word address (FWA) or an A, B, or S register (not A1) containing the *uda* address

ct Word count or an A, B, or S register (not A1 or A2) containing the word count

ubc Unused bit count or an A, B, or S register (not A1, A2, or A3) containing the unused bit count or null. If null, record contains no unused bits. Not applicable in WRITEP.

Return conditions:

(A1) DSP address

(A2) FWA of user data area (*uda*)

(A3) Requested word count (*ct*)

WRITEC/WRITECP - WRITE CHARACTERS

The WRITEC and WRITECP macros transfer characters from the user's data area to the dataset.

The WRITEC macro generates a return jump to the $WCHR subroutine, thus causing one record at a time to be processed. An end-of-record RCW is written following each WRITEC.

The WRITECP macro generates a return jump to the $WCHP subroutine. Characters are written from the user's data area as requested by the user. No EOR is written.

One character is taken from bits 56 through 63 of each word of the data area and packed into the record, eight characters per word. Blank compression occurs.

Memory-resident datasets are handled as described for WRITE/WRITEP.

To write only an end-of-record RCW, the WRITEC macro with character count of 0 is used.

Formats:

Location	Result	Operand
	WRITEC	*dn,uda,ct*

Location	Result	Operand
	WRITECP	*dn,uda,ct*

dn Dataset name (symbolic address of the Open Dataset Name Table for this dataset) or an A, B, or S register (not A0 or S0) containing the Dataset Parameter Table (DSP) address or negative DSP offset

uda User data area first word address (FWA) or an A, B, or S register (not A1) containing the *uda* addresss

ct Character count or an A, B, or S register (not A1 or A2) containing the character count

Return conditions:

Same as for WRITE/WRITEP except that the requested count (A3) is in characters rather than words

WRITED - WRITE END OF DATA

The WRITED macro generates a return jump to the $WEOD subroutine, causing an end-of-record RCW (if not previously written), an end-of-file RCW (if not previously written), and an end-of-data RCW to be written.

The WRITED macro causes buffers to be flushed. If the dataset is memory resident, buffers are flushed to mass storage only if the EOD occurs within the last block of the buffer; in this case, the memory-resident flags are also cleared.

Format:

Location	Result	Operand
	WRITED	*dn*

dn Dataset name (symbolic address of the Open Dataset Name Table for this dataset) or an A, B, or S register (not A0 or S0) containing the Dataset Parameter Table (DSP) address or negative DSP offset

Return conditions:

(A1) DSP address

WRITEF - WRITE END OF FILE

The WRITEF macro generates a return jump to the $WEOF subroutine, causing an end-of-record RCW (if not previously written) and an end-of-file RCW to be written.

If the WRITEF macro causes the buffer for a memory-resident dataset to be full, the memory-resident flags are cleared and the buffers are flushed to mass storage.

Format:

Location	Result	Operand
	WRITEF	*dn*

dn Dataset name (symbolic address of the Open Dataset Name Table (ODN) for this dataset) or an A, B, or S register (not A0 or S0) containing the Dataset Parameter Table (DSP) address or negative DSP offset

Return conditions:
(A1) DSP address

ASYNCHRONOUS READ/WRITE MACROS

The asynchronous read/write logical I/O macros allow the user to read and write words and to write an EOF or an EOD. These macros provide the Cray Assembly Language (CAL) programmer with the same capabilities as the FORTRAN BUFFER IN/BUFFER OUT statements.

Control returns to the user immediately. It is the user's responsibility to ensure that requested data transfers are complete and error free by examining the DSP before attempting to process input data or requesting additional writes. The macro BUFCHECK is provided to make the necessary checks.

All of the asynchronous blocked I/O macros use registers A0, A1, A2, S0, S1, and S2. Other A and S registers, and all B, T, and V registers remain unchanged (except B0). Unblocked I/O processing also uses registers A6, S3, and S4. In all cases, after the I/O function completes, A1 contains the DSP address. The other registers used are not meaningful. All status responses must be obtained from the DSP.

Asynchronous requests for unblocked datasets require that the *uda* parameter specify the address of an area in the user's program. Also, the *ct* parameter must specify a value that is a multiple of 512.

Memory-resident datasets are handled the same as for the synchronous read/write macros. See the description of the READ, WRITE, WRITEF, and WRITED macros for the handling of BUFINP, BUFOUTP, BUFEOF, and BUFEOD, respectively.

BUFCHECK - CHECK BUFFERED I/O COMPLETION

The BUFCHECK macro requests the system to wait until the buffered I/O on a dataset has completed and, optionally, to go to an error address if the DSP status contains any error flags when the I/O completes.

Format:

Location	Result	Operand
	BUFCHECK	*dn,err*

dn Dataset name (symbolic address of the Open Dataset Name Table (ODN) for this dataset). It is the Dataset Parameter Table (DSP) address only if *dn*=(A1). The ODN address is given in any A register other than A0; any S register except S0, S1, or S2; or any B register.

err Optional error address. If any error bits are set in the DSP on completion of the I/O, control is transferred to *err*, if specified. If *err* is not specified, it is the user's responsibility to detect any errors. Note that for this purpose, DPEOI does not constitute an error bit.

Return conditions:
If *err* is specified, S1 contains a copy of W@DPERR.

BUFEOD - WRITE END OF DATA ON DATASET

The BUFEOD macro writes an EOD to a dataset. Control returns immediately to the user and it is the user's responsibility to monitor the DPBIO field. An EOR and an EOF are also written, if necessary.

Issuing a BUFEOD macro for an unblocked dataset produces an error.

Format:

Location	Result	Operand
	BUFEOD	*dn,rcl*

dn Dataset name (symbolic address of the Open Dataset Name Table (ODN) for this dataset). It is the Dataset Parameter Table (DSP) address only if *dn*=(A1). The ODN address is given in any A register other than A0; any S register except S0, S1, or S2; or any B register.

rcl Optional Recall flag. If not null, the macro expansion includes a RECALL loop until the I/O is completed.

BUFEOF - WRITE END OF FILE ON DATASET

The BUFEOF macro writes an EOF on a dataset. Control returns immediately to the user program, giving the user the responsibility of monitoring the DPBIO field. An EOR is written if the dataset is mid record.

Issuing a BUFEOF macro for an unblocked dataset produces an error.

Format:

Location	Result	Operand
	BUFEOF	*dn,rcl*

dn Dataset name (symbolic address of the Open Dataset Name Table (ODN) for this dataset). It is the Dataset Parameter Table (DSP) address only if *dn*=(A1). The ODN address is given in any A register other than A0; any S register except S0, S1, or S2; or any B register.

rcl Optional Recall flag. If not null, the macro expansion includes a RECALL loop until the I/O is completed.

BUFIN/BUFINP - TRANSFER DATA FROM DATASET TO USER RECORD AREA

The BUFIN and BUFINP macros transfer words of data from a dataset to a user record area. Both macros generate a system call to F$BIO.

The BUFIN macro transfers data from the current position to EOR or until the specified word count is exhausted. The dataset is positioned after the end of the current record. Field DPBUBC indicates the count of unused bits in the last word of the record. If the word count is exhausted before end of record, the unused bit count is set to 0.

The BUFINP macro transfers data from the current position to EOR or until the specified word count is exhausted. The dataset remains positioned mid record if the word count is exhausted before EOR is reached. The unused bit count is set in the same way as for BUFIN.

In both cases, control returns to the user program immediately, giving the user the responsibility of monitoring the proper DSP fields to determine when the transfer is complete and whether any errors occurred.

If the dataset is unblocked, the specified word count is transferred.

Formats:

Location	Result	Operand
	BUFIN	*dn,uda,ct,rcl*

Location	Result	Operand
	BUFINP	*dn,uda,ct,rcl*

dn Dataset name (symbolic address of the Open Dataset Name Table (ODN) for this dataset). It is the Dataset Parameter Table (DSP) address only if *dn*=(A1). The ODN address is given in any A register other than A0; any S register except S0, S1 or S2; or any B register.

uda User record area or A, S, or T register (not A0, S0, S1, or S2) containing the *uda* address

ct Word count or A, S, or T register containing word count (not S1 or S2)

rcl Optional Recall flag. If not null, the macro expansion contains a RECALL loop until the I/O is completed.

Registers S1 and S2 construct the parameter word (W@DPBIO) and cannot contain parameter address or values.

When I/O is completed, for both BUFIN and BUFINP, the actual number of words transferred can be obtained from the DPBWC field of the DSP for a blocked dataset. For an unblocked dataset, this field is valid only upon completion of the BUFCHECK macro or upon completion of the BUFIN (BUFINP) macro if recall was specified.

BUFOUT/BUFOUTP - TRANSFER DATA FROM USER RECORD AREA TO DATASET

The BUFOUT and BUFOUTP macros transfer data from a user's record area to a dataset using the system F$BIO function.

The BUFOUT macro transfers the specified number of words and writes an end-of-record RCW on the dataset. Optionally, an unused bit count can be specified, giving the number of bits in the last word of data that is not to be considered as part of the data. The end-of-record RCW contains this unused bit count.

The BUFOUTP macro transfers the specified number of words but does not write an end-of-record RCW. Subsequent BUFOUTP macro calls continue to construct the record. A subsequent BUFOUT macro terminates the record with an EOR. Unused bits are meaningless for BUFOUTP.

In both cases, control returns to the user program immediately, giving the user the responsibility of monitoring the proper DSP fields to determine when the transfer is complete and whether any errors occurred.

If the dataset is unblocked, the specified word count is transferred. Unused bits are meaningless. The specified count must be a multiple of 512.

Formats:

Location	Result	Operand
	BUFOUT	*dn,uda,ct,ubc,rcl*

Location	Result	Operand
	BUFOUTP	*dn,uda,ct,ubc,rcl*

dn Dataset name (symbolic address of the Open Dataset Name Table (ODN) for this dataset). It is the Dataset Parameter Table (DSP) address only if *dn*=(A1). The ODN address is given in any A register other than A0; any S register except S0, S1, or S2; or any B register.

uda User record area or A, S, or T register containing record area address (not A0, S0, S1, or S2)

ct Word count or A, S, or T register containing word count (not A0, S0, S1 or S2)

ubc Optional unused bit count or A, S, or T register containing unused bit count (not A0, S0, or S2) or null. If null, record contains no unused bits. This field is ignored for BUFOUTP.

rcl Optional recall flag. If not null, the macro expansion contains a RECALL loop until the I/O is completed.

Registers S1 and S2 construct the parameter word (W@DPBIO) and must not contain parameter addresses or values, except that S1 can contain the unused bit count.

UNBLOCKED READ/WRITE MACROS

The unblocked dataset read and write macros allow the user to read and write data directly into or from a buffer supplied by a program rather than by the system. The job waits for I/O to complete.

The system does no blocking or deblocking of unblocked datasets.

Upon termination of the READ/WRITE function, register contents are modified as detailed under the description of each macro. A or S registers not specifically mentioned should not be assumed to have any meaningful contents, and do not contain the same values as before the function request. Registers B0, B70 through B77, and T70 through T77 can be changed, as well as VL, VM, V0, and V1. Other B, T, and V registers are not changed.

READU - TRANSFER DATA FROM DATASET TO USER'S AREA

The READU macro transfers words of data from an unblocked dataset into an area specified by the caller. The READU macro generates a return jump to the $RLB subroutine.

Format:

Location	Result	Operand
	READU	*dn,uda,ct*

dn Dataset name (symbolic address of the Open Dataset Name Table for this dataset) or an A, B, or S register (except A0 or S0) containing the Dataset Parameter Table (DSP) address or negative DSP offset

uda User data area first word address (FWA) or an A, B, or S register (except A0, A1, or S0) containing the *uda* address

ct Word count or an A, B, or S register (except A0, A1, A2, or S0) containing the number of words to be transferred. *ct* must be a multiple of 512.

Return conditions:

(A1) DSP address

(A2) FWA of user data area (*uda*)

(A3) Requested word count (*ct*)

(A4) Actual LWA+1 of data transferred

(S0) Completion status. One of the following:

-1.0	Operation complete, no errors
0.0	Attempt to read past allocated data
+1.0	Parity error
+2.0	Unrecovered hardware error†

WRITEU - TRANSFER DATA FROM USER'S AREA TO DATASET

The WRITEU macro transfers data from the user's area to an unblocked dataset. The WRITEU macro generates a return jump to the $WLB subroutine.

Format:

Location	Result	Operand
	WRITEU	*dn,uda,ct*

dn Dataset name (symbolic address of the Open Dataset Name Table for this dataset) or an A, B, or S register (not A0 or S0) containing the Dataset Parameter Table (DSP) address or negative DSP offset

† Deferred implementation

uda User data area first word address (FWA) or an A, B, or S register (not A0, A1, or S0) containing the *uda* address

ct Word count or an A, B, or S register (not A1 or A2) containing the number of words to be transferred. *ct* must specify a multiple of 512.

Return conditions:

(A1) DSP address

(A2) FWA of user data area (*uda*)

(A3) Requested word count (*ct*)

(S0) Completion status. One of the following:

-1.0	Operation complete, no errors
0.0	Attempt to write past allocated data
+1.0	Parity error
+2.0	Unrecovered hardware error†

POSITIONING MACROS

The user can rewind datasets, backspace records or files, get the current dataset position, and position datasets using the positioning logical I/O macros. See each macro description for register contents on return. Other registers mentioned as used by READ/WRITE will be meaningless on return.

When a dataset is positioned backward and the last operation on the dataset was a write operation, an EOD is written (and an EOR and EOF, if necessary). See the WRITE, WRITEF, and WRITED macro descriptions for handling of memory-resident datasets during the EOD processing. If the last operation was not a write operation, backward positioning has no special effect on a dataset.

ASETPOS - ASYNCHRONOUSLY POSITION DATASET

The ASETPOS macro generates a return jump to the $ASPOS subroutine. With asynchronous positioning, the job continues executing while positioning occurs. The dataset is positioned at the word indicated by the word offset specified, which must be at a record boundary (at BOD, or following EOR or EOF, or before EOD).

† Deferred implementation

For a blocked dataset, the macro initiates a read to the I/O buffer before positioning occurs. For an unblocked dataset, the DSP is updated to reflect the specified position within the dataset. No I/O request is actually issued. ASETPOS applies to mass storage datasets only; it is illegal for tape datasets.

Format:

Location	Result	Operand
	ASETPOS	*dn,pos*

dn Dataset name (symbolic address of the Open Dataset Name Table (ODN) for this dataset) or an A, B, or S register containing the Dataset Parameter Table (DSP) address or negative DSP offset

pos Dataset position. Can be any of the following:

EOD Position the dataset preceding EOD.

BOD Position the dataset at BOD.

S_n or T_n
Position the dataset to the word address contained in the specified S or T register. If *pos* is not S1, S1 is destroyed.

Return conditions:

(A1) Address of Dataset Parameter Table (DSP)

(S1) Dataset position; see GETPOS for meaning of flags.

(S6) Record control word after which dataset is positioned, or 0 at BOD

BKSP - BACKSPACE RECORD

Th BKSP macro generates a return jump to the $BKSP subroutine. The dataset is backspaced one record. If the initial position is at BOD, no action occurs. If the initial position is mid record, the dataset is backspaced to the beginning of that record.

Because the backspace operation occurs within the buffer for memory-resident datasets, such datasets receive special handling only if an EOD must be written. Changes made in the buffer contents are preserved.

Issuing a BKSP macro for an unblocked dataset produces an error.

BKSP applies to mass storage datasets only, and is illegal on tape datasets.

Format:

Location	Result	Operand
	BKSP	*dn*

dn Dataset name (symbolic address of the Open Dataset Name Table for this dataset) or an A, B, or S register containing the Dataset Parameter Table (DSP) address or negative DSP offset

Return conditions:

(A1) DSP address

(S6) The RCW after which the dataset was positioned; equals 0 if BOD is encountered.

BKSPF - BACKSPACE FILE

The BKSPF macro generates a return jump to the $BKSPF subroutine. The dataset is backspaced one file. If the initial position is at BOD, no action occurs. If the initial position is mid file, the dataset is backspaced to the beginning of that file.

Because the backspace operation occurs within the buffer for memory-resident datasets, such datasets receive special handling only if an EOD must be written. Changes made in the buffer contents are preserved.

Issuing a BKSPF macro for an unblocked dataset produces an error. BKSPF applies to mass storage datasets only, and is illegal on tape datasets.

Format:

Location	Result	Operand
	BKSPF	*dn*

dn Dataset name (symbolic address of the Open Dataset Name Table for this dataset) or an A, B, or S register containing the Dataset Parameter Table (DSP) address or negative DSP offset

Return conditions:
Same as for BKSP

GETPOS - GET CURRENT DATASET POSITION

The GETPOS macro generates a return jump to the $GPOS subroutine. This subroutine returns the current dataset position in S1. The dataset position is the number of words between the BOD and the present position, excluding BCWs but including RCWs.

Format:

Location	Result	Operand
	GETPOS	*dn*

dn Dataset name (symbolic address of the Open Dataset Name Table for this dataset) or an A, B, or S register (except A0 or S0) containing the Dataset Parameter Table (DSP) address or negative DSP offset

Return conditions:

(A1) DSP address

(S1) For a blocked dataset, S1 contains dataset position flags. Bits 0-2 indicate position within records or files; bits 31-63 indicate physical word address within the file, including RCWs. At BOD, (S1)=0. Bit 0=1 if dataset is positioned immediately following an RCW. Bit 2=1 if the RCW is an end-of-file RCW. If bit 0=0, bit 2 also equals 0, and the dataset is mid record or at BOD. Bit 1 is unused.

For an unblocked dataset, S1 returns the relative position of the current block within the dataset.

(S2) For an unblocked dataset, S2 contains the same address contained in bits 31-63 of S1 for blocked datasets.

For a blocked dataset, S2 contains the physical word address relative to the beginning of the dataset, including RCWs.

POSITION - POSITION TAPE

The POSITION macro allows the user to rewind or position an opened tape dataset at a particular tape block of the dataset. Data blocks on tapes are numbered so that block number 1 is the first data block on a tape. Before a tape dataset is positioned using the TP option, the dataset must be synchronized with the SYNCH macro.

The POSITION macro uses registers S0, S1, S2, S3, A1 and A2.

Format:

Location	Result	Operand
	POSITION	*dn*,REWIND
	or	
	POSITION	*dn*,TP,B=$\left\{\begin{matrix}+\\-\end{matrix}\right\}$*nb*,V=$\left\{\begin{matrix}+\\-\end{matrix}\right\}$*nv*,L=*pla*
	or	
	POSITION	*dn*,TP,B=*nb*,VOL=*vi*,L=*pla*

dn Dataset name. *dn* is a symbolic address of the Open Dataset Name Table (ODN) for this dataset, or an A, S, or T register which contains the ODN address. The ODN is described in the COS Table Descriptions Internal Reference Manual, CRI publication SM-0045.

REWIND Rewinds a tape dataset. *dn* is the only other parameter that can be used with REWIND.

TP Tape positioning request. TP and REWIND are mutually exclusive. When TP is coded, B, V, and VSN are valid parameters.

B=$\left\{\begin{matrix}+\\-\end{matrix}\right\}$*nb* Block. It can be an expression or an S, A, or T register which contains a number. However, B cannot be A0, S0, S1, S2, or S3. The value of *nb* cannot be greater than 15728639. Possible specifications for *nb* are:

+nb Specifies the amount of blocks to forward-space from the current position. The + sign is invalid if either V or VOL is coded for POSITION.

-nb Specifies the amount of blocks to back-space from the current position. The - sign is invalid if either V or VOL is coded for POSITION.

nb Specifies the absolute block number to be positioned.

V=$\left\{\begin{matrix}+\\-\end{matrix}\right\}$*nv* Volume. Volume can be an expression or an S, A, or T register which contains a number. However, V cannot be A0, S0, S1, S2, or S3. *nv* cannot be greater than 15728639.

V and VOL are mutually exclusive. Possible specifications for *nv* are:

+nv Indicates the number of volumes to forward-space from the current volume

-nv Specifies the number of volumes to back-space from the current volume

nv Indicates the absolute volume number to be positioned

If V is specified, the B parameter must specify B=*nb*, without + or - signs.

VOL=*vi* Volume identifier to be mounted. *vi* is a character string 1 to 6 characters long. *vi* can also be specified as an S or T register which contains the volume identifier; however, *vi* cannot be S0, S1, S2, or S3. When a register is specified, VOL must be left-justified and zero-filled. V and VOL are mutually exclusive.

If VOL=*vi* is coded, then the B parameter must specify B=*nb*, without + or - signs.

L=*pla* Parameter list address. *pla* is the address of a storage area whose length is defined by the symbol LE@ppl. *pla* holds the position request parameters.

The parameter list address may be a storage address, or an A, S or T register containing the address of the parameter list address. If *pla* is not coded, the POSITION macro generates a parameter list.

Return conditions:

Register S1 contains a return code when control returns to the user. The return codes are:

TPOK=0	Tape positioned successfully
TPNT=2	Dataset is not a tape dataset
TPNR=3	Tape is not at end-of-record
TPNS=4	Positioning request not fully satisfied. S2 contains the number of blocks not forward- or back-spaced.

REWIND - REWIND DATASET

The REWIND macro generates a return jump to the $REWD subroutine, causing the dataset to be positioned at beginning of data (BOD).

The REWIND macro causes all buffer pointers in the DSP to be reset to indicate an empty buffer. For memory-resident datasets, the next read causes the pointers to be reset. If the memory-resident dataset previously existed on mass storage, any changes made to the contents of the buffer before the rewind are lost. This is because the disk copy of the dataset is reread without the changes being flushed. If the dataset did not previously exist on disk, any changes in the buffer contents are preserved across the rewind and read sequence. To preserve changed buffer contents for a memory-resident dataset that previously existed on disk, use BKSPF to reposition the dataset.

Format:

Location	Result	Operand
	REWIND	*dn*

dn Dataset name. *dn* is the symbolic address of the Open Dataset Name Table for this dataset. The ODN is described in the COS Table Descriptions Internal Reference Manual, CRI publication SM-0045.

dn can also be an A, B, or S register (not A0 or S0) containing the Dataset Parameter Table (DSP) address or negative DSP offset. The DSP is described in section 2 of this manual.

Return conditions:

(A1) DSP address

SETPOS- SYNCHRONOUSLY POSITION DATASET

The SETPOS macro generates a return jump to the $SPOS subroutine. With synchronous positioning, the job waits for positioning to complete before continuing. The dataset is positioned at the word indicated by the word offset specified, which must be at a record boundary (at BOD, or following EOR or EOF, or before EOD).

For a blocked dataset, the macro initiates a read to fill the I/O buffer before positioning occurs. For an unblocked dataset, the DSP is updated to reflect the specified position within the dataset, but no I/O request is actually issued. SETPOS applies to mass storage datasets only, and is illegal for tape datasets.

Format:

Location	Result	Operand
	SETPOS	*dn,pos*

dn Dataset name. *dn* is the symbolic address of the Open Dataset Name Table (ODN) for this dataset. The ODN is described in the COS Table Descriptions Internal Reference Manual, CRI publication SM-0045.

dn can also be an A, B, or S register containing the Dataset Parameter Table (DSP) address or negative DSP offset. The DSP is described in section 2 of this manual.

pos Dataset position. Can be any of the following:

EOD Position the dataset preceding EOD.
BOD Position the dataset at BOD.

S_n or T_n
Position the dataset to the word address contained in the specified S or T register. If *pos* is not (S1), (S1) is destroyed.

Return conditions:

(A1) DSP address

(S1) Dataset position (see GETPOS for meaning of flags).

(S6) Record control word after which dataset is positioned, or 0 at beginning-of-data.

SYNCH - SYNCHRONIZE

The SYNCH macro synchronizes the program and the tape. Before issuing SYNCH, the dataset must be opened. All previous I/O operations must also be tested for completion before SYNCH is issued.

If the dataset is synchronized for input, it must be positioned at an EOR control word. However, an EOR is added to the end of the data before synchronization if: a) the dataset is an output dataset, and b) the data in the circular buffer does not end with an EOR control word. A tape dataset is not synchronized after any I/O macro is issued. For an output tape, control is not returned to the user until all of the data in the circular buffer is written to the tape.

The SYNCH macro uses registers S0, S1, S2, S3, S5, S6, S7, A1 and A2.

Format:

Location	Result	Operand
	SYNCH	*dn,pd*

dn Dataset name. *dn* is the symbolic address of the ODN table for the tape dataset. The ODN is described in the COS Table Descriptions Internal Reference Manual, CRI publication SM-0045.

dn can also be an A, S, or T register that contains the address of the ODN table. However, *dn* cannot be the following registers: A0, S0, or S1.

pd Processing direction. *pd* can be:

I Input dataset
O Output dataset

Return conditions:

When control returns to the caller, register S1 contains a return code. A return of anything other than 0 indicates that no synchronization occurred. The return codes for SYNCH are:

TPOK=0 Execution without error.
TPER=1 Execution error; the error code is in the DPERR field of the DSP.
TPNT=2 Dataset is not a tape dataset.

TAPEPOS - TAPE POSITION INFORMATION

The TAPEPOS macro produces information about the position of a tape dataset that has been successfully opened.

The information returned by TAPEPOS refers to the block that the user is going to read or write. For output datasets, the information returned by TAPEPOS can be meaningless unless the tape dataset has been synchronized by the SYNCH macro before the TAPEPOS macro is issued. A storage area of 11 words is necessary to hold the information produced.

The TAPEPOS macro uses registers S0, S1, S2, S6, S7, A1 and A2.

Format:

Location	Result	Operand
	TAPEPOS	*dn,sa*

dn Dataset name. *dn* is the symbolic address of the ODN table for the tape dataset. *dn* can also be an A, S, or T register that contains the address of the ODN table. The ODN table is described in the COS Table Descriptions Internal Reference Manual, CRI publication SM-0045.

dn cannot be the following registers: A0, S0, S1, or S2.

sa Storage address. *sa* can also be an A, S, or T register that contains the address of eleven words of storage in the user's storage area to hold the tape information.

Table 3-1 illustrates the format for information returned by the TAPEPOS macro.

Table 3-1. Information returned by the TAPEPOS macro

Field	Word	Bit	Length	Description
TPVSN	0	16	48	VSN of last block processed
TPPDN1	1	0	64	Characters 1 through 8 of PDN
TPPDN2	2	0	64	Characters 9 through 16 of PDN
TPPDN3	3	0	64	Characters 17 through 24 of PDN
TPPDN4	4	0	64	Characters 25 through 32 of PDN
TPPDN5	5	0	64	Characters 33 through 40 of PDN
TPPDN6	6	0	32	Characters 41 through 44 of PDN
TPRSV1	6	32	32	Reserved for future use
TPSEC	7	0	16	File section number
TPSEQ	7	16	16	File sequence number
TPRSV2	7	32	16	Reserved for future use
TPRSV3	7	48	16	Reserved for future use
TPVBC	8	0	32	Volume block count of last block processed by program
TPCBC	8	32	32	Number of blocks in circular buffer. For output, blocks not sent to IOP; for input, always zero.

Return conditions:

Register A1 contains a return code when control returns to the user A return code other than zero indicates an error. The return codes for TAPEPOS are:

TPOK=0 Tape information returned to the user without errors.
TPNT=2 Dataset is not a tape dataset.

PERMANENT DATASET MACROS 4

The permanent dataset macro instructions are a subset of the system function requests. Each macro generates a call to the Cray Operating System (COS) or creates a table to be later used in such a call. The function code octal value is stored in register S0; S1 and S2 provide optional arguments. The function code is processed when the program exit instruction is executed. (Note that the contents of the registers used are not restored after the call is completed.)

The permanent dataset macro instructions are divided into two categories: those that define and those that manage permanent datasets.

PERMANENT DATASET DEFINITION MACROS

The PDD macro generates a parameter table containing information about the dataset. The ACCESS, ADJUST, DELETE, DISPOSE, PERMIT, SAVE, and SUBMIT macros involved in permanent dataset management use the PDD Table. Thus, the PDD macro must accompany the use of the permanent dataset management macros. For a description of the DISPOSE and SUBMIT macros, see Dataset Management Macros, section 2. The others are described later in this section.

The LDT macro generates a table containing information required to process labels for tape datasets. The LDT macro must accompany the PDD and ACCESS macros in a program accessing a labeled tape dataset if label processing is to occur.

LDT - CREATE LABEL DEFINITION TABLE

The LDT macro creates a table called the Label Definition Table (LDT). This macro is nonexecutable, cannot appear in in-line code, and accompanies the PDD and ACCESS macros in a program accessing a labeled tape dataset.

Format:

Location	Result	Operand
ldttag	LDT	CV=*cv*,FD=*fd*,VOL=(*vsn*$_1$,*vsn*$_2$,...*vsn*$_n$), FSEC=*fsec*,FSEQ=*fseq*,GEN=*gen*,GVN=*gvn*,CDT=*yyddd*, XDT=*yyddd*,RF=*rf*,MBS=*mbs*,RS=*rs*

ldttag Symbolic address of the LDT; identical to *ldt* on PDD macro.

Parameters are in keyword form.

CV=*cv* Foreign dataset conversion mode. CV indicates if implicit data conversion is to be done by the run time library. CV values are:

ON Data conversion turned on. ON causes the library to convert the foreign internal representation to or from Cray internal representation, according to the I/O list.

OFF Data conversion turned off. The data type is not considered when OFF is specified. Full Cray words are moved to or from the foreign dataset.

FD=*fd* Foreign tape dataset translation identifier. *fd* is a 3-character code which indicates that foreign dataset translation is to be performed on the dataset. This parameter is required for run time translation. Valid values for FD are:

IBM IBM compatible sequential file
CDC Control Data compatible sequential file

VOL=(*vol*$_i$)
Volume identifier list. A list of 6-character alphanumeric volume identifiers, separated by commas, that comprise the tape dataset. The maximum number of volume identifiers per dataset is specified by an installation parameter.

FSEC=*fsec*
File section number. A number from 1 through 9999 specifying the volume in the dataset. The first section (or volume) of a dataset is numbered 0001. The default is 1.

FSEQ=*fseq*† File sequence number. A number from 1 to 9999 identifying this file among the files of this set. The first file is numbered 0001. The default is 1.

GEN=*gen*† Generation number. A number from 1 to 9999 distinguishing successive generations of the file. The default is 1.

GVN=*gvn*† Generation version number. A number from 1 to 9999 distinguishing among successive iterations of the same generation. The default is 0.

CDT=*yyddd* Creation date. *yy* specifies the year and is a number from 0 to 99. *ddd* specifies the day within the year and is a number from 001 to 366 indicating the creation date for this file.

XDT=*yyddd* Expiration date. The expiration date is in the same format as the creation date, indicating the date when this file can be overwritten.

RF=*rf* Tape dataset record format. *rf* is a 1- to 8-character code describing the record type. *rf* values for IBM tape datasets are:

U	Undefined format
F	Fixed format
FB	Fixed blocked format
V	Variable format
VB	Variable blocked format
VBS	Variable blocked spanned format

For CDC tape datasets, *rf* values are:

IIW	Internal tape format, internal block type, control word record type
SIIW	System or SCOPE internal tape format, internal block type, control word type
ICW	Internal tape format, character count block type, control word record type
SICW	System or SCOPE internal tape format, character count block type, control word record type
ICZ	Internal tape format, character count block type, zero byte record type
SICZ	System or SCOPE internal tape format, character count block type, zero byte record type

† Deferred implementation

ICS Internal tape format, character count block type, system-logical record type

SICS System or SCOPE internal tape format, character count block type, system-logical record type

RS=*rs* Record size. *rs* is expressed in units of 8-bit bytes.

MBS=*mbs* Maximum tape block size; that is, the number of 8-bit bytes in the largest tape block to be read or written. The maximum size allowed at the installation and the default are specified as installation parameters.

PDD - CREATE PERMANENT DATASET DEFINITION TABLE

The PDD macro creates a parameter table called the Permanent Dataset Definition Table (PDD). This macro is nonexecutable and must accompany the ACCESS, SAVE, DELETE, ADJUST, PERMIT, DISPOSE, or SUBMIT macros in a program. It cannot appear in inline code.

Format:

Location	Result	Operand
pddtag	PDD	DN=*dn*,PDN=*pdn*,SDN=*sdn*,ID=*uid*,MF=*mf*,TID=*tid*, DF=*df*,DC=*dc*,SF=*sf*,RT=*rt*,ED=*ed*,RD=*rd*,WT=*wt*, MN=*mn*,DT=*dt*,CS=*cs*,LB=*lb*,LDT=*ldt*, NEW=[ON / OFF],MSG=[ON / OFF],UQ=[ON / OFF],WAIT=[ON / OFF], DEFER=[ON / OFF],NRLS=[ON / OFF],EXO=[ON / OFF],SID=*mf*,DID=*mf*, OWN=*ov*,PARTIAL=[NO / YES],PAM=*m*,ADN=*adn*,ADNM=*m*, TA=[YES / NO],RP=[YES / NO],USR=*ov*.

pddtag Symbolic address of the PDD Table

Parameters are in keyword form; the only required parameter is DN.

DN=*dn* Dataset name. DN is a required parameter.

PDN=*pdn* Permanent dataset name. The default value is *dn*.

SDN=*sdn* Staged dataset name; 1-15 alphanumeric characters by which the dataset is known at the destination mainframe. The default is the local dataset name (DN).

ID=*uid* User identification; 1-8 alphanumeric characters assigned by the dataset creator.

MF=*mf* Mainframe identifier; 2 alphanumeric character identification. This parameter identifies the front-end station where the dataset is to be staged. If omitted, the mainframe where the issuing job originated is used. If MF is given a value of CRAY ID and DC=IN, the dataset is disposed to the Cray system input queue, after first issuing a warning message.

NOTE

If using the DISPOSE macro, see the description of the DISPOSE control statement in CRAY-OS Version 1 Reference Manual, publication SR-0011.

TID=*tid* Terminal identifier; 1-8 alphanumeric character identifier for the destination terminal. The default is the terminal of job origin.

DF=*df* Dataset format. This parameter defines whether the destination computer is to perform character conversion. The default is CB.

df is a 2-character alpha code defined for use on the front-end computer system. CRI suggests support of the following codes:

CD Character/deblocked. The front-end system performs character conversion from 8-bit ASCII, if necessary.

CB Character/blocked. No deblocking is performed at the Cray mainframe before staging. The front end performs deblocking and character conversion from 8-bit ASCII, if necessary.

BD Binary/deblocked. The front-end system performs no character conversion.

BB Binary/blocked. The front-end computer performs no character conversion but does perform deblocking. No deblocking is performed at the Cray computer before staging.

TR Transparent. No blocking/deblocking or character conversion is performed.

IC Interchange tape datasets only. In interchange format, each tape block of data corresponds to a single logical record in COS blocked format.

Other codes can be added by the local site. Undefined pairs of characters can be passed but are treated as transparent mode by the Cray system.

DC=*dc* Disposition code; disposition to be made of the dataset. The default is PR (print).

dc is a 2-character alphabetic code describing the destination of the dataset as follows:

IN Input (job) dataset. The dataset is to be queued as a job on the mainframe specified by the MF parameter.

ST Stage to mainframe. Dataset is made permanent at the mainframe designated by the MF parameter.

SC Scratch dataset. Dataset is deleted.

PR Print dataset. Dataset is printed on any printer available at the mainframe designated by the MF parameter. PR is the default value.

PU Punch dataset. Dataset is punched on any card punch available at the mainframe designated by the MF parameter.

PT Plot dataset. Dataset is plotted on any available plotter at the mainframe designated by the MF parameter.

MT Write dataset on magnetic tape at the mainframe designated by the MF parameter.

SF=*sf* Special form information to be passed to the front-end system; 1-8 alphanumeric characters. SF is defined by the needs of the front-end system. Consult a Cray Research analyst for options.

RT=*rt* Retention period; a value between 0 and 4095 specifying the number of days a permanent dataset is to be retained by the system. The default is an installation-defined value.

ED=*ed* Edition number; a value between 1 and 4095 assigned by the dataset creator. The default is the highest edition number known to the system.

RD=*rd* Read control word; 1-8 alphanumeric characters assigned by the dataset creator. The default is no read control word.

WT=*wt* Write control word; 1-8 alphanumeric characters assigned by the dataset creator. The default is no write control word.

MN=*mn* Maintenance control word; 1-8 alphanumeric characters assigned by the dataset creator. The default is no maintenance control word.

DT=*dt* Tape dataset generic device name. (See definition of generic device names in the CRAY-OS Version 1 Reference Manual, publication SR-0011). This parameter is required for tape datasets; it is ignored when used for mass storage datasets. The generic name *TAPE specifies a device capable of 1600 or 6250 bpi.

CS=*cs* Character set of tape dataset, for data only. This parameter applies only to tape datasets, and is ignored when used for mass storage datasets.

AS ASCII; default.
EB EBCDIC

LB=*lb* Tape dataset label processing option. This parameter applies only to tape datasets; it is ignored when used for mass storage datasets.

BLP By-pass label processing†
SL IBM standard labeled tapes
NL Unlabeled tapes; default.
AL ANSI standard labeled tapes

† Deferred implementation

LDT=*ldt* Label Definition Table (LDT). The name of the LDT for tape processing. This parameter applies only to tape datasets, and is ignored when used for mass storage datasets. *ltd* is identical to *ldttag* on the LDT macro.

NEW={ON / OFF} Tape dataset is to be created; the dataset must be written starting at the beginning of information.

ON Tape dataset to be created
OFF Tape dataset not to be created; default.

MSG={ON / OFF} Normal completion message suppression indicator. The default is OFF.

ON Indicator is set, message is suppressed
OFF Indicator is cleared, message is not suppressed

UQ={ON / OFF} Unique access. If UQ is specified, write, maintenance, and/or read permission is granted if the appropriate write or maintenance control words are specified. The default (OFF) is multiread access if the read control word is specified (if one exists).

WAIT={ON / OFF}

Job wait/nowait. If WAIT=ON is specified, the job waits for the dataset to be transferred to the front-end system. If the transfer is canceled, the job is aborted. If WAIT=OFF is specified, the job resumes immediately and does not wait for the dataset to be transferred. If the transfer is canceled, the job is not aborted. If the parameter is omitted, an installation default parameter is used.

DEFER={ON / OFF}

Deferred disposition. When DEFER is specified, disposing of the dataset is delayed until the dataset is released either by a RELEASE request or by termination.

The default is OFF; the dataset is disposed immediately.

NRLS={ON / OFF}

No release. When NRLS=ON is specified, the dataset remains local to the job after a DISPOSE request has been processed. The default is NRLS=OFF.

NOTE

The dataset is available only for reading when NRLS=ON is specified.

EXO=$\begin{Bmatrix} ON \\ OFF \end{Bmatrix}$ Execute-only dataset. EXO=ON sets the execute-only status of a dataset. EXO=OFF clears the execute-only status. If omitted, the status is ignored.

SID=*mf* Default source mainframe identifier. Two alphanumeric characters. This parameter defines the source front-end station where all staging to the Cray system mainframe defaults.

DID=*mf* Default destination mainframe identifier. Two alphanumeric characters. This parameter defines the destination front-end station where all staging from the Cray system mainframe defaults.

NOTE

Use of the MF parameter with either SID or DID is not allowed.

OWN=*ov* Ownership value; default is jobs ownership value.

PARTIAL=*opt*
Partial delete option; default is NO.

PAM=*m* Public access mode; default is installation defined.

ADN=*adn* Attributes dataset name; default is no *adn*.

ADNM=*m* Attributes to be propagated from *adn*; default is all.

TA=*opt* Access tracking option; default is installation defined.

RP=*opt* Remove permit option; default is NO.

USR=*ov* Dataset user number

PERMANENT DATASET MANAGEMENT MACROS

The user can access, save, adjust, delete, and permit permanent datasets by use of the permanent dataset management macros. All of these macros must be accompanied by the PDD macro in the job.

ACCESS - ACCESS PERMANENT DATASET

The ACCESS macro associates an existing permanent dataset with a job and assures that the user is authorized to use this dataset. ACCESS must precede any other references for the permanent dataset.

Format:

Location	Result	Operand
	ACCESS	*pddtag*

pddtag Address of PDD

ADJUST - ADJUST PERMANENT DATASET

The ADJUST macro changes the size of a permanent dataset, that is, redefines EOD for the dataset. A dataset must be accessed with the write permission control word and unique access within a job before an ADJUST is issued.

If all of the following conditions are true for the dataset, ADJUST makes a call to close the dataset and consequently to flush the buffer. This assures that all the data is written to the dataset.

- Dataset is currently opened for output.
- Dataset has not had an EOD written.
- Dataset is being written sequentially.
- Dataset has COS blocked dataset structure.
- Dataset's DSP is managed by COS.

ADJUST does not close the dataset unless all of these conditions are true.

ADJUST applies to mass storage datsets only, and is ignored when used with tape datasets.

Format:

Location	Result	Operand
	ADJUST	*pddtag*

pddtag Address of PDD

DELETE - DELETE PERMANENT DATASET

The DELETE macro removes a permanent dataset from the Dataset Catalog (DSC). If the dataset is mass storage, it must be accessed within a job with the maintenance permission control word and unique access before a DELETE is issued. If the dataset is a tape dataset, a request is made to the servicing front end to remove the dataset from the catalog.

Format:

Location	Result	Operand
	DELETE	*pddtag*

pddtag Address of PDD

PERMIT - EXPLICITLY PERMIT DATASET

The PERMIT macro allows a user to explicitly designate user permanent dataset availability. The macro works exactly the same as the PERMIT control statement described in CRAY-OS Version 1 Reference Manual, publication SR-0011.

The dataset need not be local for a PERMIT to be issued. However, if the ADN parameter is used in defining the PDD, that dataset must be local and permanent.

Format:

Location	Result	Operand
	PERMIT	*pddtag*

pddtag Address of PDD

SAVE - SAVE PERMANENT DATASET

The SAVE macro enters a local mass storage dataset in the Dataset Catalog, making it permanent. A permanent dataset is uniquely identified by permanent dataset name, user identification, and edition number. If the dataset is a tape dataset, a request is sent to the servicing frontend to catalog the dataset.

SAVE has a twofold function:

- Creation of an initial edition of a permanent dataset
- Creation of an additional edition of a permanent dataset

If all of the following conditions are true for the dataset, SAVE makes a call to close the dataset and consequently to flush the buffer. This assures that all the data is written to the dataset.

- The dataset is currently opened for output.
- The dataset has not had an EOD written.
- The dataset is being written sequentially.
- The dataset has COS blocked dataset structure.
- The dataset's DSP is managed by COS.

SAVE does not close the dataset unless all of these conditions are true.

Format:

Location	Result	Operand
	SAVE	*pddtag*

pddtag Address of PDD

CFT LINKAGE MACROS 5

The CFT linkage macros handle subroutine linkage between CFT-compiled routines and CAL-assembled routines.

These macros perform the following functions:

- Generate code for standard entry and exit sequences
- Build the proper linkages for subroutine calls
- Define symbolic names for passed-in arguments
- Assign symbolic names to B and T registers
- Allocate space for local temporary variable storage
- Fetch argument addresses
- Retrieve the number of arguments passed to a subroutine
- Load and store local temporary variables
- Return the actual address of local temporary variable storage

These macros maintain compatibility across versions of CFT.

DESIGN OF THE ENTRY BLOCK MACROS

With the use of the CFT linkage macros, the entry section of a CAL routine can be made to resemble the entry area of a CFT subroutine. The DEFARG, DEFB, DEFT, ALLOC, MXCALLEN, PROGRAM and ENTER macros define the calling list, B and T register usage and temporary storage space, and routine type. These macros are generally used at the beginning of a module and can be considered nonexecutable code or data declarations.

These macros must be used in the following order:

1. DEFARG macro
2. DEFB macro
3. DEFT macro
4. ALLOC macro
5. MXCALLEN macro
6. ENTER or PROGRAM macro

DEFARG - DEFINE CALLING PARAMETERS

The DEFARG macro defines a symbolic name for a passed-in parameter. The symbols are assigned to the passed-in arguments in the order in which they are defined. For example, the symbol attached to the first DEFARG is assigned to the first argument, the symbol attached to the second DEFARG is assigned to the second argument, etc. These symbols can be used in the ARGADD macro, which is defined later. The number of arguments defined is also used by the ENTER macro to control the generation of the entry sequence.

Format:

Location	Result	Operand
name	DEFARG	

name Symbol to be assigned to the passed-in argument

Example:

Location	Result	Operand	Comment
1	10	20	35
COUNT	DEFARG		Defines argument named COUNT

DEFB - ASSIGN NAMES TO B REGISTERS

The DEFB macro reserves a B register and assigns a symbolic name to it. If B registers are not assigned explicitly, the DEFB macro uses the next available B register. The B registers can be assigned from either of two classes, temporary or nontemporary.

Eight temporary B registers are available (B70 through B77). Temporary B registers are not saved on entry to a subroutine nor preserved across

calls. Use of these registers does not cause overhead on entry or exit. Temporary B registers should not be used if lower level routines are called, since the registers can be destroyed during the call.

Format:

Location	Result	Operand
name	DEFB	*explicit register designator*

name Symbol to be assigned to a B register

explicit register designator
Specific B register to be assigned the symbolic name. Values can be blank, TEMP, or an *expression*.

If this field is blank, the next available B register is assigned starting with the first register after those used by the calling sequence. If the word TEMP is used in this field, then the next available temporary B register is assigned starting with B70. An explicit register can also be designated by coding an *expression* in this field. Coding a specific number in this field should be avoided since the number of B registers used by the CFT calling sequence may change, possibly invalidating the specific register usage. This macro also checks if a register has been previously assigned a name and prohibits its reuse.

Example:

Location	Result	Operand	Comment
1	10	20	35
VADDR	DEFB		Assigns next available nontemporary B register
SEGS	DEFB	VADDR+7	Skips 7 B registers before assigning next nontemporary
LEN	DEFB	TEMP	Assigns next available temporary B register

DEFT - ASSIGN NAMES TO T REGISTERS

The DEFT macro reserves a T register and assigns a symbolic name to it. If T registers are not assigned explicitly, the DEFT macro uses the next available T register. The T registers can be assigned from either of two

classes, temporary or nontemporary. Eight temporary T registers are available (T70 through T77). Temporary T registers are not saved on entry to a subroutine and use of these registers causes no overhead on entry or exit. Temporary registers are neither saved nor preserved across calls. Temporary T registers should not be used if lower level routines are called, since the registers might be destroyed during the call.

Format:

Location	Result	Operand
name	DEFT	*explicit register designator*

name Symbol to be assigned to a T register

explicit register designator
Specific T register to be assigned the symbolic name. Values can be blank, TEMP, or an *expression*.

If this field is left blank, the next available T register is assigned starting with the first register after those used by the calling sequence. If the word TEMP is used in this field, then the next available temporary T register is assigned starting with T70. An explicit register can also be designated by coding an *expression* in this field. Coding a specific number in this field should be avoided since the number of T registers used by the CFT calling sequence can change, possibly invalidating specific register usage. This macro also checks if a register has been previously assigned a name and prohibits its reuse.

Example:

Location	Result	Operand	Comment
1	10	20	35
COEFF	DEFT		Assigns next available nontemporary T register
XMULT	DEFT	COEFF+3	Skips 3 T registers before assigning next nontemporary
SIZE	DEFT	TEMP	Assigns next available temporary T register

ALLOC - ALLOCATE SPACE FOR LOCAL TEMPORARY VARIABLES

The ALLOC macro establishes memory storage space and assigns a symbolic name to it. This space is used for local storage within the routine and is not assumed to be zeroed on entry or preserved on exit. This macro uses the CAL pseudo instruction BSS (see CAL Assembler Version 1 Reference Manual, CRI publication SR-0000); but when reentrant code is generated, this macro creates definitions to dynamically allocate storage space from a stack at run time. The symbolic names defined by this macro are used in conjunction with the LOAD, STORE, and VARADD macros explained later.

Format:

Location	Result	Operand
name	ALLOC	*size*

name Symbolic name associated with the first word of the storage area

size Size in words of the area to be allocated. This field can be any valid CAL expression. The default for the size is one word.

Example:

Location	Result	Operand	Comment
1	10	20	35
WIDTH	ALLOC		Allocates one word of memory storage space
VTEMP	ALLOC	D'64	Allocates 64 words of memory storage space

MXCALLEN - DECLARE MAXIMUM CALLING LIST LENGTH

This macro allocates storage space for an argument list to be passed to call-by-address routines. The length is calculated by checking all calls to call-by-address routines and determining the maximum argument list length used by any call. This macro is only needed when dynamic stack management is used.

Format:

Location	Result	Operand
	MXCALLEN	*length*

length Maximum length of any argument list passed to a call-by-address routine. The length does not include the argument list header word.

Example:

Location	Result	Operand	Comment
1	10	20	35
	MXCALLEN	7	Defines seven words to be used to store argument lists for call-by-address routines.

PROGRAM - GENERATE MAINLINE CAL ROUTINE START POINT

The PROGRAM macro generates a starting point for a CAL mainline routine. This macro uses a START pseudo instruction to declare a main entry point for a program and establishes symbols required for the LOAD, STORE, VARADD, CALL, and CALLV macros. When used with stacks, this macro generates stack storage space for B and T register save areas and local temporary variables.

Format:

Location	Result	Operand
name	PROGRAM	STKPTR=*stkptr*,SCR=*scr*

name Symbolic name associated with the start point

STKPTR=*stkptr*
STKPTR specifies an A register to contain the base-of-stack frame pointer on exit from the PROGRAM macro. The LOAD, STORE, VARADD, CALL and CALLV macros default to use this register to load the stack frame pointer. Only registers A1 through A5 and A7 are valid options for the STKPTR parameter (register A6 is used as the argument list pointer). The default value for the stack pointer register is A7.

NOTE

The STKPTR parameter is used only when dynamic stack management at run time is in effect.

SCR=*scr* Scratch register designates a default register to be used by the LOAD, STORE and VARADD macros. The valid options are registers A1 through A5 and A7. The scratch register must not be the same as the register specified by the STKPTR parameter. The default value of the SCR parameter is A5.

ENTER - GENERATE CFT-CALLABLE ENTRY POINT

The ENTER macro generates code for a standard call from CFT. The macro generates call-by-value or call-by-address entry points and conditionally generates code for both types of CFT calling sequences and for reentrant entry sequences. The ENTER macro also force loads argument values, if desired, and aligns the first executable statement on an instruction buffer boundary.

Format:

Location	Result	Operand
name	ENTER	NP=*np*,NB=*nb*,NT=*nt*,MODE=*mode*,TYPE=*type*, SHARED=*share*,PRELOAD=*nr*,ARGSIZE=*size*, ALIGN=*align*,STKPTR=*stkptr*,SCR=*scr*, COPYIN=*copy*,INSRTMAC=(*macname*,(*maclist*))

name Symbolic name associated with the entry point

NP=*np* Number of parameters expected to be passed to the routine. The default value is the number of parameters defined using the DEFARG macro.

NB=*nb* Number of nontemporary B registers used by the routine. These registers are saved on entry and restored on exit. *nb* does not include the B registers used by the calling sequence. The default is the number of nontemporary B registers defined using the DEFB macro.

NT=*nt* Number of nontemporary T registers used by the routine. These registers are saved on entry and restored on exit. Currently no T registers are used in the calling sequence. The default is the number of nontemporary T registers defined using the DEFT macro.

MODE=*mode* Entry sequence to be generated. Available options are USER, LIBRARY, and BASELVL.

USER mode entry is default.

LIBRARY mode entry is intended for special use in the libraries and is a somewhat faster but more restrictive and unsafe entry method.

BASELVL mode is a simplified and restrictive entry sequence intended for use in the primitive level library routines that make no external calls.

On completion of the entry sequence for MODE=LIBRARY and MODE=BASELVL, register A1 contains the traceback information for the routine. This must be preserved during execution for traceback to function properly.

****

CAUTION

MODE=LIBRARY and MODE=BASELVL are intended for use by Cray systems programmers and are not for general use.

****

TYPE=*type* Method to be used for passing arguments. The available options are VALUE, ADDRESS and BOTH. The default entry type is ADDRESS.

TYPE=ADDRESS entry produces a standard CFT callable entry point. This assumes that the arguments are passed to the routine by an argument address list that contains the addresses of the actual arguments.

TYPE=VALUE assumes that the arguments are passed to the routine in the S or V registers. TYPE=VALUE entries are incompatible with the method of calling subroutines used by CFT (except for functions declared by CFT's VFUNCTION directive).

TYPE=BOTH generates both entry types, an ADDRESS entry followed by a VALUE entry. To distinguish between these two entry points, a % is appended to the name for the call-by-value entry point.

SHARED=*share*

Name associated with a previously defined ENTER macro. If this parameter is specified, the previous entry and the current entry share storage space for the B & T save area. SHARED is intended for routines such as SIN and COS, which also share code sequences. When SHARED is used, the NB and NT parameters cannot be used. The number of B and T registers saved, the amount of memory allocated, and the maximum calling list length must be the same for both entries being shared. The name specified by the SHARED parameter must precede the current entry. No unshared entries can be used between the current entry and the entry being shared.

PRELOAD=*nr*

Number of arguments to be loaded into the S or V registers in a call-by-address entry. If the first character of the entry point name is %, then the arguments are loaded into V registers; otherwise, arguments are loaded into S registers. V register arguments have two parts: the first part is the base address of the argument, the second part is the address of the skip distance between argument values. The default number of registers to preload is 0 for MODE=USER and is the value of *np* for MODE=LIBRARY and BASELVL entries.

ARGSIZE=*size*

Size of the arguments to preload. The options are ONEWORD, TWOWORD, and THREEWORD. This option is only necessary when MODE=ADDRESS or BOTH and PRELOAD are being used. The default for the ARGSIZE option is ONEWORD.

When ARGSIZE=ONEWORD is specified, arguments are loaded in order into the corresponding registers with the first argument loaded in S1 or V1, the second argument loaded in S2 or V2, etc. No more than seven 1-word arguments can be preloaded.

When ARGSIZE=TWOWORD is specified, arguments are assumed to consist of two words stored in memory with the most significant word first. The 2-word arguments are loaded into the registers with the first argument's most significant word loaded into S1 or V1 and the first argument's least significant word loaded into S2 or V2. The second argument is loaded into S3 and S4 or V3 and V4. No more than three 2-word arguments can be preloaded.

When ARGSIZE=THREEWORD is specified, preloading is similar to 2-word arguments except that registers S1, S2 and S3 or V1, V2 and V3 are loaded. Only two 3-word arguments can be preloaded.

ALIGN=*align*

Causes the entry macro to align the first executable instruction of the entry sequence on an instruction buffer boundary. The values for the ALIGN parameter are ON or OFF. The ALIGN=ON option forces instruction buffer alignment. The default value for the ALIGN parameter is OFF.

STKPTR=*stkptr*

Specifies an A register to contain the base of stack frame pointer on exit from the ENTER macro. The LOAD, STORE, VARADD, CALL and CALLV macros default to use this register to load the stack frame pointer. Only A1 through A5 and A7 are valid options for the STKPTR parameter (register A6 is the argument list pointer). The default value for the scratch register is A7.

NOTE

The STKPTR parameter is used only when dynamic stack management at run time is in effect.

SCR=*scr* Specifies an A register to be used as a scratch register during the entry sequence. The valid options are A1 through A5 and A7. The scratch register must not be the same as the register specified by the STKPTR parameter. The default value of the SCR parameter is A5. The SCR parameter also serves to designate a default value for the LOAD, STORE and VARADD macros. If no SCR parameter is coded on the ENTER macro, then the LOAD, STORE, and VARADD macros default to use A5; otherwise, these macros default to use the SCR register defined on the ENTER macro.

COPYIN=*copy*

Causes the ENTER macro to build code to convert the new calling sequence to the old calling sequence. This parameter is intended to aid in the conversion of routines written for the old calling sequence. The values for this parameter are ON and OFF. When COPYIN=ON is specified, code is produced to translate the new calling sequence to the old version. This parameter should not be used on new programs and should be avoided on older programs because of the overhead added to the entry sequence. When COPYIN=ON is specified and the new calling sequence is in effect, error traceback processing does not function properly because of the incompatibilities between the two calling sequences.

INSTRMAC=(*macname*,(*maclist*))

Only needed when the user performs special processing before the standard entry sequence is executed. Cases where special processing might be necessary include preserving registers before entry and setting a hardware semaphore. The code to be inserted is placed immediately following any constants generated by the ENTER macro and immediately preceding the code needed to perform the entry. The user should define special processing as a macro and reference this macro name on the INSRTMAC parameter. The INSRTMAC parameter causes the ENTER macro to invoke the user-defined macro.

maclist consists of a list of parameters to be passed to the user-defined macro specified by the INSRTMAC parameter. This parameter is optional but, if included, should follow *macname* and be enclosed in parentheses.

Examples:

Location	Result	Operand	Comment
1	10	20	35
GETDATA	ENTER	MODE=USER,TYPE=ADDRESS,ALIGN=ON	

Defines an entry point called GETDATA for a call-by-address routine. Forces the first executable instruction to be aligned on an instruction buffer boundary. The number of arguments to be passed to this routine is defined by the number of DEFARG macros used. The B and T registers to be saved are defined by the DEFB and DEFT macros.

Location	Result	Operand	Comment
1	10	20	35
FOLR	ENTER	NP=3,NB=7,NT=4,MODE=USER,TYPE=BOTH	

Defines two entry points to a routine. The first entry point is FOLR, which is call by address. The second entry point is FOLR%, which is call by value and not accessible from CFT. The number of parameters passed to these entry points is explicitly defined to be 3 and must agree with the number of DEFARG macros if they are used. Seven B registers are to be saved in addition to those used by the calling sequence. These registers must agree with the definitions provided by the DEFB macros if they are used. Four T registers are to be saved in addition to those used by the calling sequence. These registers must agree with the definitions provided by the DEFT macros if they are used.

Location	Result	Operand	Comment
1	10	20	35
SECDED	ENTER	NP=2,MODE=LIBRARY,TYPE=BOTH,ARGSIZE=TWOWORD, STKPTR=A4,SCR=A5	

Defines two entry points to a routine. The first entry point is SECDED, which is call by address. The second entry point is SECDED%, which is call by value. The number of parameters is explicitly declared to be 2 and must agree with the number of DEFARG macros if they are used. The arguments are declared to be two words long and are preloaded for the call-by-address entry into registers S1 and S2 for argument 1 and registers S3 and S4 for argument 2. Register A5 is used for a scratch register during the entry sequence and is the default scratch register for all LOAD, STORE, and VARADD macros. Register A4 contains the base-of-stack pointer value if dynamic stacks are used at run time. The shortened form of the entry sequence used for library routines is generated. Register A1 contains a pointer to error traceback information on completion of the entry sequence.

RETRIEVE PASSED-IN ARGUMENT LIST INFORMATION MACROS

Two macros get information about passed-in dummy arguments for call-by-address routines. The ARGADD macro returns the addresses of passed in arguments from the calling list. The NUMARG macro returns the number of arguments passed to the routine.

ARGADD - FETCH ARGUMENT ADDRESS

The ARGADD macro fetches an argument address (not a value) and returns it in a register. This macro is needed only for call-by-address routines. The ARGADD macro can be used only when an ENTER macro has been used first. Arguments should be referred to symbolically by first declaring them with a DEFARG macro.

Format:

Location	Result	Operand
	ARGADD	*result*,*argument*,USE=*use*,ARGPTR=*argptr*

result Result register (either A or S) to be loaded with the address of the argument. If the result register is an S register, the entire word (64 bits) from the argument list is returned in the register. This parameter is important for character arguments where the first 40 bits of the argument list value contain information about the string size and starting bit.

argument Name or number of the argument address to be returned. *argument* can be an A register containing the number of the argument address to be returned. The first argument is number 1, the second is number 2, etc. Argument names should be used and the names should be declared by the DEFARG macro.

USE=*use* An A register to reload the argument address list pointer if the current value of this pointer has been destroyed.

ARGPTR=*argptr*
An A register currently containing the address of the argument list. This parameter prevents the argument list pointer from being reloaded before the argument address is fetched. The ARGPTR and USE parameters cannot be used at the same time since USE causes the argument list pointer to be reloaded and ARGPTR specifies that the argument list pointer is already available and need not be reloaded. If neither the ARGPTR nor the USE parameter is specified, the macro attempts to use the result register to recover the argument list pointer. If the result register is A0 or an S register, it issues a warning message and uses A6.

Example:

Location	Result	Operand	Comment
1	10	20	35
	ARGADD	A1,CX,USE=A6	Returns the address of the CX argument in register A1. Reload the argument address list pointer into register A6 (register A6 is assumed to have been destroyed before this point and must be reloaded).
	ARGADD	A2,Y,ARGPTR=A6	Returns the argument address in register A2 for a dummy argument named Y. Register A6 is assumed to currently contain the pointer to the argument address list.
	ARGADD	A3,A2,USE=A6	Returns the address of an argument in A3. The number of the argument to return is contained in A2.

NUMARG - GET THE NUMBER OF ARGUMENTS PASSED IN

The NUMARG macro gets the number of arguments actually passed to a routine. This macro works for both call-by-address and call-by-value routines provided that the calling routine has properly used the CALL or CALLV macro (see discussion later in this section). The NUMARG macro can be used only when the ENTER macro is used to define the entry sequence.

Format:

Location	Result	Operand
	NUMARG	*result*,USE=*use*,ARGPTR=*argptr*

result Result register (either A or S) to be loaded with the number of arguments

USE=*use* An A register to reload the argument address list pointer if the current value of this pointer has been destroyed.

ARGPTR=*argptr*
An A register currently containing the address of the argument list. This parameter prevents the argument list pointer from being reloaded before the argument address is fetched. The ARGPTR and USE parameters cannot be used at the same time since USE causes the argument list pointer to be reloaded and ARGPTR specifies that the argument list pointer is already available and need not be reloaded. If neither the ARGPTR nor the USE parameter is specified, the macro attempts to use the result register to recover the argument list pointer. If the result register is A0 or an S register, it issues a warning message and uses A6.

Example:

Location	Result	Operand	Comment
1	10	20	35
	NUMARG	A1,USE=A6	Returns the number of arguments actually passed to the routine in A1. Reloads the argument address list pointer into register A6 (register A6 is assumed to have been destroyed before this point and must be reloaded).
	NUMARG	A2,ARGPTR=A6	Returns, in A2, the number of arguments actually passed to the routine. Register A6 is assumed to currently contain the pointer to the argument list.

REFERENCE LOCAL TEMPORARY VARIABLE STORAGE MACROS

Three macros reference temporary memory storage space defined by the ALLOC macro, discussed earlier in this section. This storage space is assumed to be defined only for the length of time that the routine is executing and is not preserved once a program exits. This space is defined using the CAL pseudo instruction BSS (see CAL Assembler Version 1 Reference Manual, CRI publication SR-0000) when stacks are not being used and is acquired at run time from a general pool when stacks are being used. The three macros, LOAD, STORE, and VARADD, can be used to store values into memory, retrieve values from memory, and return the actual address of a memory storage location for a particular execution of a routine.

LOAD - GET VALUE FROM MEMORY INTO A REGISTER

The LOAD macro transfers data from a local temporary storage area into a register. The macro generates code to load from memory areas defined by the CAL pseudo instruction BSS (see CAL Assembler Version 1 Reference Manual, CRI publication SR-0000) and by the ALLOC macro. If dynamic stack management is in effect, the LOAD macro retrieves data from areas defined at run time from the general memory pool.

Format:

Location	Result	Operand
	LOAD	*result*,*dataname*,INDEX=*index*,STKPTR=*stkptr*, USE=*use*, SCR=*scr*,SKIP=*skip*

result Result register (either A, S, or V) to be loaded from memory

dataname Name of the memory location to be loaded. This can be the name of an area defined by an ALLOC macro or a CAL pseudo instruction BSS.

INDEX=*index*

An A register for loading an offset. This value is added to the address of the data item and can be used as an offset for elements in tables or to increment to the next segment for a vector load.

STKPTR=*stkptr*

An A register that currently contains the value of the stack pointer. The macro uses this register as an index to recover values from memory. *stkptr* prevents the stack pointer from being reloaded. This parameter is needed only when the data item to load is on the stack.

USE=*use* An A or B register for recovering the dynamic stack pointer. If the current pointer to the stack has been destroyed, a USE parameter must be coded to force a reload of the stack pointer value. If an A register is specified for the USE parameter, that register contains the current stack base address after the macro has been executed. If a B register is specified, the B register is used as temporary storage to recover the stack pointer. Its contents are indeterminate after the LOAD macro. The USE and STKPTR parameters cannot be used at the same time since STKPTR specifies that the stack pointer is already available and USE specifies that the stack pointer must be reloaded. If neither STKPTR nor USE is specified, a warning message is issued and the stack pointer register specified on the ENTER macro is reloaded with the current stack pointer value; default is A7. The USE parameter is needed only when the data item to load is on the stack.

SCR=*scr* Specifies a scratch A register to be used during execution of the LOAD macro. This register is needed only when vector registers are to be loaded or when an INDEX parameter is used. If no register is specified in these cases, a warning message is issued and the scratch register defined by the ENTER macro is used; default is A5.

SKIP=*skip* Specifies an A register containing the skip distance for a vector load. The default value for the skip parameter is register A0 (a skip distance of 1).

Examples:

Location	Result	Operand	Comment
1	10	20	35
	LOAD	S1,X,STKPTR=A7	Transfers to register S1 the contents of the memory location labeled X. It is assumed that register A7 contains the stack base address and that it need not be reloaded.
	LOAD	V1,VSAVE	Transfers to register V1 beginning at the memory location labeled VSAVE. The number of words to transfer is determined by the current value of the VL register. The skip distance between elements is assumed to be 1. A warning is issued and the stack pointer register specified on the ENTER macro is reloaded. Another warning is issued because no scratch register was specified and one was needed. The scratch register specified on the ENTER macro is used.
	LOAD	A2,TADDR,USE=A7	Transfers to register A2 the contents of the memory location labeled TADDR. It is assumed that the STKPTR register defined by the ENTER macro has been destroyed and that A7 recovers the current stack base address.
	LOAD	S3,TABLE, INDEX=A4, SCR=A5, STKPTR=A7	Transfers to register S3 from the memory address computed by finding the current address of TABLE and adding the contents of register A4. The stack pointer is contained in register A7 and need not be reloaded. A5 is used as a scratch register to perform intermediate address calculations.

STORE - STORE THE VALUE FROM A REGISTER INTO MEMORY

The STORE macro transfers data from a register to a local temporary storage area. The macro generates code stored to memory areas defined by the CAL pseudo instruction BSS and by the ALLOC macro. If dynamic stack management is in effect, the STORE macro stores data to areas defined at run time from the general memory pool.

Format:

Location	Result	Operand
	STORE	*source*,*dataname*,INDEX=*index*,STKPTR=*stkptr*, USE=*use*,SCR=*scr*,SKIP=*skip*

source Source register (either A, S, or V) to be stored to memory

dataname Name of the destination memory location. This parameter can be the name of an area defined by an ALLOC macro or CAL pseudo instruction BSS.

INDEX=*index*
An A register for storing an offset. This value is added to the address of the data item. This value can be used to offset to elements in tables or to increment to the next segment for a vector store.

STKPTR=*stkptr*
An A register that currently contains the value of the stack pointer. The macro uses this register as an index to recover values from memory. *stkptr* prevents the stack pointer from being reloaded. This parameter is needed only when the item to which data is stored is on the stack.

USE=*use* An A or B register for recovering the dynamic stack pointer. If the current pointer to the stack has been destroyed, a USE parameter must be coded to force a reload of the stack pointer value. If an A register is specified for the USE parameter, that register contains the current stack base address after the macro has been executed. If a B register is specified, the B register is used as temporary storage to recover the stack pointer. Its contents are indeterminate after the STORE macro. The USE and STKPTR parameters cannot be used at the same time since STKPTR specifies that the stack pointer is already available and USE specifies that the stack pointer must be reloaded. If neither STKPTR nor USE is specified, a warning message is issued and the stack pointer register

specified on the ENTER macro (default is A7) is reloaded with the current stack pointer value. The USE parameter is needed only when the item to which data is stored is on the stack.

SCR=*scr* Scratch register specifies a scratch A register for use during execution of the STORE macro. This register is needed only when vector registers are to be stored or when an INDEX parameter is used. If no register is specified in these cases, a warning message is issued and the scratch register defined by the ENTER macro is used; default is A5.

SKIP=*skip* Skip distance specifies an A register containing the skip distance for a vector store. The default value for the skip parameter is register A0 (a skip distance of 1).

Examples:

Location	Result	Operand	Comment
1	10	20	35
	STORE	S1,TEMP1, STKPTR=A7	Transfers the contents of register S1 to the memory location labeled TEMP1. It is assumed that register A7 contains the current stack base address.
	STORE	V1,VSAVE, SCR=A5, STKPTR=A7	Transfers the contents of register V1 to the area beginning at the memory location labeled VSAVE. The number of words to transfer is determined by the current value of the VL register. The skip distance between elements is assumed to be 1 and the stack pointer is contained in A7. Register A5 is used for intermediate address calculations.
	STORE	A2,TADDR,USE=A7	Transfers the contents of register A2 to the memory location labeled TADDR. It is assumed that the STKPTR register defined by the ENTER macro has been destroyed and that A7 recovers the current stack base address.

Examples (continued):

Location	Result	Operand	Comment
1	10	20	35
	STORE	S3,TABLE, INDEX=A4	Transfers the contents of S3 to the memory address computed by finding the current address of TABLE and adding the contents of register A4. The STKPTR register defined on the ENTER macro recovers the stack base address. The register defined by the SCR parameter on the ENTER macro performs intermediate address calculations; default is A5. Warnings are issued for both STKPTR and SCR register usage.

VARADD - RETURN THE ADDRESS OF A MEMORY LOCATION

The VARADD macro retrieves the address of a memory storage area. The macro generates code to return addresses of memory locations defined by the ALLOC macro for both stack and nonstack modes.

Format:

Location	Result	Operand
	VARADD	*result*,*dataname*,STKPTR=*stkptr*,USE=*use*,SCR=*scr*

result Result register (either A or S) to receive the address of the memory location

dataname Name of the memory location address to be found. This name can be defined by an ALLOC macro or a CAL pseudo instruction BSS.

STKPTR=*stkptr*
An A register currently containing the value of the stack pointer. The macro uses this register as an index to compute the memory address. This parameter prevents the stack pointer from being reloaded. This parameter is needed only when the data item is on the stack.

USE=*use* An A or B register for recovering the dynamic stack pointer. If the current pointer to the stack has been destroyed, a USE parameter must be coded to force a reload of the stack pointer value. If an A register is specified for the USE parameter, that register contains the current stack base address after the macro has been executed. If a B register is specified, the B register is used as temporary storage to recover the stack pointer. Its contents are indeterminate after the VARADD macro. The USE and STKPTR parameters cannot be used at the same time since STKPTR specifies that the stack pointer is already available and USE specifies that the stack pointer must be reloaded. If neither STKPTR nor USE is specified, a warning message is issued and the stack pointer register specified on the ENTER macro is reloaded with the current stack pointer value; default is A7. The USE parameter is needed only when the data item is on the stack.

SCR=*scr* Specifies a scratch A register for use during execution of the VARADD macro. This register is needed only when S registers or register A0 are to receive the memory address. If no register is specified in these cases, a warning message is issued and the scratch register defined by the ENTER macro is used; default is A5.

Examples:

Location	Result	Operand	Comment
1	10	20	35
	VARADD	A1,TEMP1, STKPTR=A7	Gets the address of the memory location labeled TEMP1. It is assumed that register A7 contains the stack pointer and that it need not be reloaded.
	VARADD	A2,VADDR,USE=A7	Gets the address of the memory location labeled VADDR. It is assumed that the STKPTR register defined by the ENTER macro has been destroyed and that A7 recovers the current stack base address.

Examples (continued):

Location	Result	Operand	Comment
1	10	20	35
	VARADD	A0,TABLE,SCR=A2	Returns in A0 the address of the memory location labeled VADDR. The STKPTR register defined on the ENTER macro reloads the stack pointer and a warning is issued. Register A2 performs intermediate calculations to find the address.

CALL EXTERNAL ROUTINES MACROS

Two macros generate the call-by-address and call-by-value calling sequences. These macros provide the correct calling linkages for Cray products and assist the traceback function for error processing.

CALL - CALL A ROUTINE USING CALL-BY-ADDRESS SEQUENCE

The CALL macro builds an argument address block for a call-by-address routine and invokes the routine. The address block is built separately and pointed to by register A6.

Format:

Location	Result	Operand
	CALL	*name*,*arglist*,STKPTR=*stkptr*,USE=*use*

name Name of the call-by-address routine being called or an A register containing the address of the routine to call

arglist List of arguments to be passed to the call-by-address routine. The list can consist of literal values, registers, or memory locations. If several items are in the argument list, they must be separated by commas and enclosed in parentheses.

NOTE

If registers are specified as arguments, the contents of these registers are stored in the argument list passed to the called routine. For example, if the user specifies S1 in the *arglist*, the contents of S1 are stored in the argument list passed to the called routine and the called routine regards the contents of S1 as the address where the actual argument is stored.

STKPTR=*stkptr*
An A register currently containing the value of the stack pointer. The macro uses this register as an index to compute memory addresses. This parameter prevents the stack pointer from being reloaded. This parameter is needed only when stacks are in use.

USE=*use* An A or B register to recover the dynamic stack pointer. If the current pointer to the stack has been destroyed, a USE parameter must be coded to force a reload of the stack pointer value. If an A register is specified for the USE parameter, that register should not be used in the argument list. The USE and STKPTR parameters should not be used at the same time since STKPTR specifies that the stack pointer is already available and USE specifies that the stack pointer must be reloaded. If neither STKPTR nor USE is specified, warning messages are issued and the stack pointer register specified on the ENTER macro is reloaded with the current stack pointer value; default is A7. The stack pointer provides an area to store the argument address list when dynamic memory management is being used. The stack pointer value also computes the addresses of local temporary variables defined by the ALLOC macro. The USE parameter is needed only when stacks are in use.

Example:

Location	Result	Operand	Comment
1	10	20	35
	CALL	SASUM,(N,A3,1) STKPTR=A7	Calls routine SASUM using call-by-address method. The first argument is stored in memory location N. The second argument is stored in memory at a location pointed to by A3.

Example (continued):

Location	Result	Operand	Comment
1	10	20	35
			The third argument is a literal value of 1 and is defined as a memory constant. The stack pointer is contained in A7 and need not be reloaded.

CALLV - CALL A ROUTINE USING CALL-BY-VALUE SEQUENCE

The CALLV macro generates a call by value to an external routine. The arguments for the routine are passed in registers S1 through S7. Register S1 contains the first argument, S2, the second, etc. A word is built containing the number of arguments and pointed to by register A6.

Format:

Location	Result	Operand
	CALLV	*name*,*arglist*,STKPTR=*stkptr*,USE=*use*

name Name of the call-by-value routine being called or an A register containing the address of the routine to call

arglist List of arguments to be passed to the call-by-value routine. The list can consist of literal values, registers, or memory locations. If several items are in the argument list, they must be separated by commas and enclosed in parentheses.

STKPTR=*stkptr*
An A register currently containing the value of the stack pointer. The macro uses this register as an index to compute memory addresses. This parameter prevents the stack pointer from being reloaded. This parameter is needed only when data items passed as arguments are on the stack.

USE=*use* An A or B register to recover the dynamic stack pointer. If the current pointer to the stack has been destroyed, a USE parameter must be coded to force a reload of the stack pointer value. If an A register is specified for the USE parameter, that register should not be used in the argument

list. The USE and STKPTR parameters should not be used at the same time since STKPTR specifies that the stack pointer is already available and USE specifies that the stack pointer must be reloaded. If neither STKPTR nor USE is specified, warning messages are issued and the stack pointer register specified on the ENTER macro is reloaded with the current stack pointer value; default is A7. The stack pointer computes the addresses of local variables defined by the ALLOC macro. The USE parameter is needed only when stacks are in use.

Example:

Location	Result	Operand	Comment
1	10	20	35
	CALLV	RTOI%,(S5,5)	Call routine RTOI% using call-by-value method. The first argument is moved to S1 from register S5. The second argument is generated by loading S2 with a literal value of 5. Since neither of these arguemtns is on the stack, no STKPTR or USE parameter is needed.

****

CAUTION

The CALLV macro loads the S registers in order without checking if the S register being loaded is used later as an argument. CALLV ROUTINE, (S2,S1) causes S1 to be loaded from S2 and S2 to be loaded with the modified contents of S1. This example results in both S1 and S2 being loaded with the original contents of S2.

****

EXIT - TERMINATE SUBROUTINE AND RETURN TO CALLER

The EXIT macro constructs a subroutine exit sequence. The exit sequence includes restoring any B and T registers saved by the entry sequence and deallocating any dynamic memory storage used by a routine. The EXIT macro also provides an alternate exit address for routines that require a different exit, such as for error conditions.

Format:

Location	Result	Operand
	EXIT	NB=*nb*,NT=*nt*,MODE=*mode*,NAME=*name*,KEEP=*keep*, ALTEXIT=*altexit*,INSRTMAC=(*macname*,(*maclist*))

NB=*nb* Number of nontemporary B registers to restore. This does not include the B registers used by the calling sequence. The default is the number of B registers saved by the last ENTER macro.

NT=*nt* Number of nontemporary T registers to restore. This does not include the T registers used by the calling sequence. The default is the number of T registers saved by the last ENTER macro.

MODE=*mode* Exit sequence to be generated. *mode* can be USER, LIBRARY, and BASELVL.

USER mode exit is default.

LIBRARY mode exit is for special use in the libraries and is a faster but more restrictive and unsafe exit method.

BASELVL mode is a very simplified and restrictive exit sequence. BASELVL mode exit is intended for use in the primitive level library routines that make no external calls and should be used with extreme caution. Register A1 is assumed to contain the traceback information for exit MODE=LIBRARY and MODE=BASELVL. This means that register A1 must be preserved during the execution of a routine if these exit types are to be used. MODE=BASELVL exit must be used on exit if MODE=BASELVL was used on entry.

****

CAUTION

MODE=LIBRARY and MODE=BASELVL are intended for use by Cray systems programmers and are not for general use.

****

NAME=*name* Name associated with the ENTER macro that corresponds to this EXIT. The default is the name of the last ENTER macro without a SHARED clause.

KEEP=*keep* Causes the exit macro to preserve any A or S registers used during the exit sequence. The values for the KEEP parameter are ON or OFF. If KEEP=ON is specified, then registers B77 and B76 are used to save the A registers needed by the exit code. KEEP=OFF destroys registers A0 and A7 during exit. The default value for the KEEP parameter is OFF.

ALTEXIT=*altexit*
Specifies an A register containing an alternate return address. This register should not be A0, A1, or A7. The default is to return to the calling routine.

INSRTMAC=(*macname*,(*maclist*))
Needed only when the user performs special processing before the standard exit sequence is executed. Cases where this might be necessary include restoring registers before exit or clearing a hardware semaphore, etc. The code to be inserted is placed immediately following the standard exit sequence and immediately preceding the jump to B00. The user should define this special processing as a macro and reference this macro name on the INSRTMAC parameter. The INSRTMAC parameter causes the EXIT macro to invoke the user-defined macro.

maclist consists of a list of parameters to be passed to the user-defined macro specified by the INSRTMAC parameter. This parameter is optional but, if included, it should follow the *macname* and be enclosed in parentheses.

Examples:

Location	Result	Operand	Comment
1	10	20	35
	EXIT	MODE=USER	Restores default number of B and T registers defined by the entry sequence and perform standard exit sequence for a user routine.
	EXIT	NB=3,NT=5, ALTEXIT=A3	Restores 3 B registers and 5 T registers not including those used by the calling sequence. Jumps to the address stored in register A3 for the exit.
	EXIT	INSRTMAC=(RESTREGS,(V1,V2,V3))	Performs standard exit processing but before jumping back to calling routine, invokes the RESTREGS macro. The RESTREGS macro is passed a parameter list of V1,V2,V3.

TABLE AND SEMAPHORE MANIPULATION 6

The three groups of macros and opdefs included in this section assist the user in the manipulation of tables and semaphores. The macros in the first group define and construct both normal and complex tables. These tables are maintained during program execution by the partial-word manipulation opdefs in the second group. Finally, the macros in the third category assist the user in defining and manipulating semaphores.

TABLE DEFINITION AND CONSTRUCTION MACROS

Table definition and construction macros aid the programmer in defining and building tables. By using these macros, the programmer can define two types of tables: tables oriented to 64-bit words defined by *normal* macros, and those *not* oriented to 64-bit words defined by *complex* macros.

These tables are maintained during program execution by the partial-word manipulation opdefs. Each table type has its own set of opdefs for getting and setting defined fields.

NOTE

Mixing normal and complex table macros for the definition and maintenance of a table structure is not valid and results in assembly errors.

NORMAL TABLE MACROS

Tables that use 64-bit words are constructed using the following macros:

- BUILD Constructs a table structure at assembly time
- ENDTABLE Designates the end of a table definition
- FIELD Defines a field within the current table structure

- NEXTWORD Advances a specified number of words in the current table structure
- REDEFINE Redefines a specified word or group of words in the current table structure
- SUBFIELD Identifies fields contained within a larger field
- TABLE Defines the overall table attributes

BUILD - Construct a table structure

The BUILD macro is used with the TABLE and FIELD macros to construct a table at assembly time. The table is defined in terms of the symbols defined by TABLE and FIELD macros. The BUILD macro eliminates the need for coding variable word definition (VWD) statements (see the CAL Assembler Version 1 Reference Manual, CRI publication SR-0000) to set initial values. Fields specified in the BUILD macro must not overlap or be repeated, although these conditions are not diagnosed. Fields not explicitly named in the BUILD macro are initialized to 0.

Format:

Location	Result	Operand
loc	BUILD	*pfx,length,(list)*

loc Location symbol. If present, *loc* is defined as the location of word 0 of the table being built and has an attribute of a word. BUILD always forces the location counter to a word boundary before constructing the table.

pfx A 2- or 3-character prefix identifying the field definition for the table being built. This prefix is not necessarily the table name as specified on the table macro, but is the prefix used in the FIELD macro (such as, DP for DSP or EQ for EQT).

NOTE

Use of the BUILD macro requires that all FIELD definitions use field names beginning with the characters given by *pfx*.

length Designates length of table. Normally, *length* should be L, LH, LE, or SZ. BUILD concatenates the length designator with the @ character and *pfx* to produce a symbol that defines the length of the table designated by the TABLE macro location field.

If the length of the table cannot be given by concatenating the L, LH, LE, or SZ characters with @*pfx*, it must be given in the format (*length*@*pfx*) where the parentheses and blank are mandatory.

list Field names and values to be entered into the fields. The list must be enclosed in parentheses unless only one field name and value is specified or no field is specified. If no field is specified, the table is initialized with all zeros.

A field name can be any characters for which a FIELD macro has been assembled, without the 2- or 3-character prefix. Fields can be in any order. The list must not contain the character. Entries in the list have the form:

fieldname=value

where *value* is the data to be assembled into the field and can contain any characters (except \) legal in the value field of a VWD instruction (see the CAL Assembler Version 1 Reference Manual, CRI publication SR-0000).

Examples:

1. The following examples show the table definition and its fields and the construct of the table.

   (a) This entry defines a table named TAB; all field definitions are prefixed by TAB.

Location	Result	Operand	Comment
1	10	20	35
TAB	TABLE	LE=20,NE=3,LH=2	

   (b) These entries define some fields for the table header.

Location	Result	Operand	Comment
1	10	20	35
TABHF1	FIELD	0,0,16	
TABHF2	FIELD	1,0,64	

Example 1 (continued):

(c) These entries define some fields for the table entry.

Location	Result	Operand	Comment
1	10	20	35
	REDEFINE	0,1	
TABF1	FIELD	0,0,64	
TABF2	FIELD	2,1,4	
TABF3	FIELD	3,6,12	

(d) The following set of instructions constructs the table during assembly.

Location	Result	Operand	Comment
1	10	20	35
B@TAB	BUILD	TAB,LH,(HF1='TB'L,HF2='HDR 2'L)	Construct header
	BUILD	TAB,LE,(F1='TABF1'L,F2=O'37)	Build one entry
	DUP	NE@TAB-1,1	
	BUILD	TAB,LE	Build remaining entries for all fields initialized to 0

2. The following example builds an Equipment Table (EQT) entry. Refer to the COS Table Descriptions Internal Reference Manual, publication SM-0045,† field names.

(a) This entry builds the EQT header.

Location	Result	Operand	Comment
1	10	20	35
B@EQT	BUILD	EQ,(LH@EQT),(NE=I@NDD819,NUA=0,TN='EQT'L)	

† This manual is available only on tape. See your CRI analyst for information.

Example 2 (continued):

(b) This code builds the entry for the master device.

Location	Result	Operand	Comment
1	10	20	35
EQT00	BUILD	EQ,(LE@EQT),(LDV='DD-19-20'L,MSD=1,CH1=2,UN=0,____	
,_____		DRT=DRT00,CPD=NCY819,TPC=NTK819,AU=NBL819,LNK=EQT04)	

(c) The following builds an entry for a non-master device.

Location	Result	Operand	Comment
1	10	20	35
EQT01	BUILD	EQ,(LE@EQT),(LDV='DD-19-30'L,CH1=3,UN=0,DRT=DRT01,__	
,_____		CPD=NCY819,TPC=NTK819,AU=NBL819,LNK=EQT05)	

NOTE

The length parameter is enclosed in parentheses and contains a blank character because the length of an Equipment Table (EQT) entry or header is not defined in terms of the same prefix as the field names. The expansion of the macro yields the following line:

Location	Result	Operand	Comment
1	10	20	35
%TL	SET	LE@EQT @EQ	

The blank before @EQ terminates the assembler scan of the line. If the blank is omitted, the expansion shown below produces an assembly error. Omitting the parentheses produces the same expansion and also produces an assembly error.

Location	Result	Operand	Comment
1	10	20	35
%TL	SET	LE@EQT@EQ	

ENDTABLE - Designate the end of a table definition

The ENDTABLE macro specifies the end of a table description. It verifies that LE@*name* (if defined) is at least as large as the last word defined plus either the last specified block size (the *length* from a NEXTWORD or REDEFINE macro) or 1, if the last field in a table is not a block. If no LE@*name* symbol is defined, one is defined with a value equal to the highest next word counter reached during the assembly of the associated FIELD macros. If the LE@*name* is defined, it is checked to ensure that no FIELD macro for the table referenced or word number is inconsistent with the defined LE@*name* symbol.

Format:

Location	Result	Operand
name	ENDTABLE	

name A 1- to 5-character name of the table being described

FIELD - Define a field with current table structure

The FIELD macro sets up the special labels for fields in tables. These labels can then be used by the GET, SGET, PUT, SPUT, and SET macros. When adding a new table or set of FIELD macros, precede the first call to FIELD with a TABLE macro, even if it contains no parameters. Otherwise, an assembly error might result on the first FIELD macro. All field definitions must be in numerically ascending order by word and bit number. The current word counter is set each time the assembler encounters either (a) a FIELD macro that contains a specific numeric value for *word*, (b) a NEXTWORD macro, (c) a REDEFINE macro, or (d) a FIELD macro containing a + in the word parameter.

Format:

Location	Result	Operand
name	FIELD	*word,sbit,number*

Expansion:

W@*name*	=	*word*
S@*name*	=	*sbit*
N@*name*	=	*number*

name A 1- to 6-character name of the field being described

word Relative 64-bit word index in table or table entry in which the field resides. Special variations of this parameter are:

* * Suppresses the definition of W@*name*

* $ Indicates that the current word (the last explicitly given word, or the last word number generated by a NEXTWORD macro) is to be used. This format is useful when new words are added to the middle of a large table.

* + Causes the macro expansion to generate a call to NEXTWORD, with *word* value of the current next word counter and a *length* value of 1.

sbit Starting (leftmost) bit number of the field in the word. Bit 0 is the sign bit. * suppresses the S@*name* and N@*name* definitions; *number* must be omitted if * is used.

number Field size in number of bits. This parameter should not be left null unless *sbit*=*.

Example:

Location	Result	Operand	Comment
1	10	20	35
DNBSZ	FIELD	0,6,24	
DNDAT	FIELD	1,24,40	
DNBFZ	FIELD	3,12,15	
DNUSR	FIELD	16,*	
DNXYA	FIELD	*,24,40	

NEXTWORD - Advance a specified number of words

The NEXTWORD macro allows the addition of words into large tables without recoding the definition of all fields in subsequent words of the table, unless such subsequent definitions provide specific numeric values for *word*. This macro is used with a special form of the FIELD macro to automatically maintain the current word number.

Format:

Location	Result	Operand
	NEXTWORD	*word,length*

word Optional word number to reset counters. If *word* is omitted, counters are set to the next sequential word resulting from a previous NEXTWORD macro or FIELD macro containing a specific word number. Default value is the current value of the current word counter plus the *length* from the most recently encountered NEXTWORD or REDEFINE macro, or current value of the current word counter + 1 if a FIELD macro specifying a numeric word value, or a +, has been encountered since the NEXTWORD or REDEFINE.

length Optional size of block being defined. If *length* is omitted, one word is assumed. *length* is used when a block larger than one word is needed in a table but FIELD macros do not exist for individual fields within words other than the first word. If *length* is present, the following NEXTWORD macro sets the current word counter to current word counter plus *length*. The default value is 1.

Example:

Location	Result	Operand	Comment
1	10	20	35
TAB	TABLE	LH=3	Begin definition of table TAB. Current word counter is set to 0.
F1	FIELD	$,0,24	W@F1=0 (Fields in header)
	NEXTWORD		
F2	FIELD	$,40,24	W@F2=1
	NEXTWORD		
F3	FIELD	$,40,24	W@F3=2
	REDEFINE	0	Reset current word to 0 to define fields in entry.
E1	FIELD	$,0,1	W@E1=0
E2	FIELD	2,40,24	W@E2=2, redefines current word.
	NEXTWORD	4,6	
E3	FIELD	$,40,24	W@E3=4
	NEXTWORD		
E4	FIELD	$,0,1	W@E4=D'10
TAB	ENDTABLE		Defines LE@TAB=D'11.

REDEFINE - Redefine a specified number of words

The REDEFINE macro allows specified words to be redefined when defining a table that has multiple formats for one or more words. For example, this macro is used for the Permanent Dataset Definition Table (PDD), which has different formats depending on the function being performed. A REDEFINE macro must precede each word or block of words that has one or more alternative definitions and each possible format, to reset necessary counters and prevent FIELD macro errors. All parameters are optional.

Format:

Location	Result	Operand
name	REDEFINE	*word,length*

name A 1- to 5-character table identifier referenced in the TABLE macro for the table. If *name* is present, W@*name* is defined as *word*. If *length* is 1 or omitted and *name* is present, S@*name* and N@*name* are defined as 0 and 64, respectively. If *name* is not present, no symbols are defined.

word The beginning word number of the range to be defined. If *word* is ommitted, the redefinition begins at the current word value.

length Length of a block in words being reserved when a field is to be a multiple of words in length (that is, the Dataset Name Table portion of a System Dataset Table entry). The default value is 1.

SUBFIELD - Identify fields within a larger field

The SUBFIELD macro allows fields to overlap when two or more fields use the same bit range. This macro is used only when overlapping fields are logical subfields, such as the DPERR field that contains 12 separately addressable bits, each having its own field definition and name. Use of this macro is not appropriate when two separate formats are used for either an entire table or one or more words in a table. The SUBFIELD macro must be preceded by a FIELD macro. If a subfield is defined using the FIELD macro instead of the SUBFIELD macro, an assembly error is produced.

A subfield must be completely contained within the bit range described by the immediately preceding FIELD macro. Multiple SUBFIELD macros may follow one FIELD macro, and all must be completely within that field.

Format:

Location	Result	Operand
name	SUBFIELD	*sbit,length*

Expansion:

W@*name*	=	%%LOC$$
S@*name*	=	*sbit*
N@*name*	=	*length*

name A 1- to 6-character name of the field being described

sbit Number identifying the leftmost bit of the field

length Number identifying the field length in bits

W@*name*, S@*name*, and N@*name* are defined. W@*name* uses the word number from the most recent FIELD macro.

Example:

A typical example is the Dataset Parameter Table (DSP) error bits, where each bit is defined and the group of bits is referenced as field DPERR.

Location	Result	Operand	Comment
1	10	20	35
DPERR	FIELD	1,1,12	DSP error flags
DPEOI	SUBFIELD	1,1	EOI in buffer
DPENX	SUBFIELD	2,1	Dataset does not exist
DPEOP	SUBFIELD	3,2	Dataset not open

Expansion:

```
W@DPEOP  =  %%LOC$$
S@DPEOP  =  3
N@DPEOP  =  2
```

TABLE - Define the overall table attributes

The TABLE macro defines symbols relating to a table but it does not create the table (see BUILD). This macro describes the special labels for the length of the table header, length of each entry in the table, number of entries in the table, and size of the table.

Format:

Location	Result	Operand
name	TABLE	LH=*lh*,LE=*le*,NE=*ne*,L=*lt*,SZ=*st*

Expansion:

```
LH@name  =  lh
LE@name  =  le
NE@name  =  ne
L@name   =  lt
SZ@name  =  st
```

All parameters are optional.

name A 1- to 5-character name of the table being described

LH=*lh* Length of table header in words

LE=*le* Length of entry in words

NE=*ne* Number of entries in the table

L=*lt* Table length in words

SZ=*st* Table length in words

The following instruction defines a symbol equal to the size of a table containing a header and multiple entries. Occurrence of a TABLE macro sets the current word number (used by a special form of the FIELD macro) to 0.

Format:

Location	Result	Operand
SZ@*name*	=	LH@*name*+LE@*name**NE@*name*

Example:

Location	Result	Operand	Comment
1	10	20	35
DRT	TABLE	LH=3,LE=67,NE=2,SZ=LH@DRT+LE@DRT*NE@DRT	

COMPLEX MACROS

The programmer can define fields in tables that are capable of spanning 64-bit word boundaries. Tables of this type are defined using the following macros:

- CENDTAB Designates the end of a complex table structure
- CFIELD Defines a field within the current complex table structure
- CNXTWORD Advances a specified number of 64-bit words in the current complex table structure
- CREDEF Redefines a specified word or group of 64-bit words for the current complex table structure

- CSBFIELD Identifies fields contained within a larger field
- CTABLE Defines the overall table attributes

CENDTAB - End a complex table structure

The CENDTAB macro terminates the definition of the current complex table and automatically assigns a value to table entry length (LE@*name*) if not defined. When the entry length has been previously defined by CTABLE, that length is checked to ensure that no field definition resides outside of that length.

Format:

Location	Result	Operand
name	CENDTAB	

name A 1- to 5-character name of the table being described. *name* is required and must match the name on the CTABLE macro.

Example:

Location	Result	Operand	Comment
1	10	20	35
VOL1	CTABLE		
VLID	CFIELD	0,0,32	Volume ID. Equal to VOL1
VLLVL	CFIELD	9,56,8	Standards Level
VOL1	CENDTAB		

Expansion:

```
LE@VOL1  =          10
```

CFIELD - Define a field in the current complex table

The CFIELD macro identifies a field definition within a complex table structure. This identification is accomplished by defining special labels that in turn are used by the CPUT and CGET opdefs. The CFIELD macro differs from the FIELD macro in that fields can be defined in CFIELD that span 64-bit word boundaries. The definition of a complex field must adhere to the following rules:

- The definition must be contained within a complex table definition, (that is, within a CTABLE CENDTAB sequence).

- All definitions must be in numerically ascending order with respect to word and bit positions.
- The field length cannot exceed 64 bits.

Format:

Location	Result	Operand
name	CFIELD	*word,sbit,length*

Expansion:

X@*name*	=	*word*
T@*name*	=	*sbit*
O@*name*	=	*length*

name A 1- to 6-character name of the field being described. If *name* is omitted, the definition of the special labels is skipped.

word 64-bit word index into the table. If *word* is numeric, it must be greater than or equal to the word index of the previously defined field. Special variations of this parameter are:

- * Suppresses the definition of X@*name*
- $ Equivalent to the current 64-bit word being defined in the table
- \+ Equivalent to the next 64-bit word in the table

NOTE

With complex tables, fields can span 64-bit word boundaries, causing an automatic incrementing of the current word location for a table.

sbit Starting (leftmost) bit number of the field. If *sbit* is numeric, that value must be between 0 and 63 inclusive and must also be greater than the *length* of the previously defined field. The special value for *sbit* is *. This value suppresses the starting bit (T@*name*) and length (O@*name*) definitions.

length Field length in number of bits. If *length* is omitted, a default length of 1 bit is assumed. If *length* is given a numeric value, *length* must be between 1 and 64, inclusive.

Example:

Location	Result	Operand	Comment
1	10	20	35
VLLID	CFIELD	0,0,24	
VLNUM	CFIELD	$,24,8	
VLVSN	CFIELD	$,32,48	
VLACC	CFIELD	$,16,8	

Expansion:

```
X@VLLID  =          0
T@VLLID  =          0
O@VLLID  =          24

X@VLNUM  =          0
T@VLNUM  =          24
O@VLNUM  =          8

X@VLVSN  =          0
T@VLVSN  =          32
O@VLVSN  =          48

X@VLACC  =          1
T@VLACC  =          16
O@VLACC  =          8
```

CNXTWORD - Advance a specific number of 64-bit words

The CNXTWORD macro allocates an area in a table but leaves out definitions. It is used primarily when the table being constructed contains other previously defined tables as elements, such as the Job Table Area (JTA) storing a Permanent Dataset Definition (PDD) in its static part. The use of CNXTWORD complements tables that are constructed with relative word references in the field definitions (that is, *name* CFIELD $,*sbit*,*length*).

Format:

Location	Result	Operand
	CNXTWORD	*word*,*length*

word The 64-bit word indexes the table where the CNXTWORD is to begin. If *word* is omitted, the current 64-bit word being constructed is used.

NOTE

With complex tables, fields can span 64-bit word boundaries, thus causing an automatic incrementing of the current word location for a table.

length Length of the block to allocate in 64-bit words. If *length* is omitted, the default block length is set to one 64-bit word. The value for *length* plus the value for *word* becomes the new current word counter after the CNXTWORD macro.

Example:

Location	Result	Operand	Comment
1	10	20	35
	DLRL	CFIELD	1,16,40
		CNXTWORD	,LE@ROS
	DLBOF	CFIELD	$,8,16

Expansion:

```
X@DLRL   =          1
T@DLRL   =          16
O@DLRL   =          40

X@DLBOF  =          1+LE@ROS+1
T@DLBOF  =          8
O@DLBOF  =          16
```

CREDEF - Redefine specific number of 64-bit words

The CREDEF macro allows a block of 64-bit words to be redefined within a table. The primary use of this macro is to redefine whole tables such as those used for the ANSI tape dataset label group definitions.

Format:

Location	Result	Operand
	CREDEF	*word,length*

word The 64-bit word index into the table where the redefinition is to begin. If *word* is omitted, the redefinition begins at the current 64-bit word being constructed.

NOTE

With complex tables, fields can span 64-bit word boundaries, causing an automatic incrementing of the current word location for a table.

length Length of the block to redefine in 64-bit words. If *length* is omitted, the default block length is set to one 64-bit word.

Example:

Location	Result	Operand	Comment
1	10	20	35
DLG	CTABLE	LE=10	
DLID	CFIELD	0,0,32	
DLFID	CFIELD	0,*	
	CREDEF	0,LE@DLG	
DLID	CFIELD	0,0,32	
DLFMT	CFIELD	0,32,8	

CSBFIELD - Define field entirely within another field

The CSBFIELD macro is used to define a field (defined by CFIELD) as smaller, separate fields. As with CFIELD, fields defined by CSBFIELD can span 64-bit word boundaries. The following rules must be followed for a field to be defined successfully.

- The subfield must lie within the last defined field (CFIELD).
- A subfield must begin after the end of the last subfield defined.

Format:

Location	Result	Operand
name	CSBFIELD	*sbit,length*

Expansion:

X@*name*	=	%%LOC$$
T@*name*	=	*sbit*
O@*name*	=	*length*

name A 1- to 6-character name of the field being described

sbit Number identifying the leftmost bit of the field

length Number identifying the field length in bits

Example:

Location	Result	Operand	Comment
1	10	20	35
HDXPR	CFIELD	5,56,48	
HDXPYR	CSBFIELD	56,24	
HDXPDY	CSBFIELD	16,24	

Expansion:

X@HDXPR	=	5
T@HDXPR	=	56
O@HDXPR	=	48
X@HDXPYR	=	5
T@HDXPYR	=	56
O@HDXPYR	=	24
X@HDXPDY	=	6
T@HDXPDY	=	16
O@HDXPDY	=	24

CTABLE - Define overall table attributes

The CTABLE macro identifies the beginning of a definition for a new complex table structure. It also defines the table in terms of special labels, the length of the table header, the length of a table entry, the number of table entries, and the overall table length. If any attribute is omitted from the parameter list, its corresponding symbol is not defined. This macro must precede each unique table definition.

Format:

Location	Result	Operand
name	CTABLE	LH=*lh*,LE=*le*,NE=*ne*,L=*lt*,SZ=*st*

Expansion:

LH@*name*	=	*lh*
LE@*name*	=	*le*
NE@*name*	=	*ne*
L@*name*	=	*lt*
SZ@*name*	=	*st*

name A required 1- to 5-character name of the table being described

LH=*lh* Length of the table header in 64-bit words

LE=*le* Length of table entry in 64-bit words

NE=*ne* Number of table entries

L=*lt* Table length in 64-bit words

SZ=*st* Table length in 64-bit words

Example:

Location	Result	Operand	Comment
1	10	20	35
VOL1	CTABLE	LH=0,LE=D'10	

Expansion:

LH@VOL1	=	0
LE@VOL1	=	D'10

PARTIAL-WORD MANIPULATION OPDEFS

The partial-word manipulation opdefs maintain tables during program execution. They can either obtain values from or set values into defined fields of a table. These opdefs are in two basic categories: normal fields that use 64-bit words and complex fields that use other than 64-bit words. Complex fields can physically reside in more than one 64-bit word. The fetching and storing of values are always with the value right-justified in its holding register.

NORMAL OPDEFS

The opdefs for normal fields include:

- GET Fetches the contents of a field
- GETF Fetches the contents of a field; cannot be followed by SPUT.
- PUT Stores data from a register into a field
- SET Packs a field value into a register possibly along with other fields
- SGET Fetches the contents of a field (short form)
- SPUT Stores data from a register into a field (short form)

GETF generates less code than GET and should be used instead of GET, except in those rare cases when SPUT is to follow. GET and PUT are sufficient for fetching and storing, respectively. However, SGET, SPUT, and SET are provided for object-time code efficiency. These opdefs assume that the field to be fetched, stored, or packed is in a word that is already in a register as a result of an earlier GET call. Additionally, SPUT assumes that the GET call was for the same field to be stored and that the mask for the field is in a register ready for use by SPUT. When a GET is for a field occupying a full word, CAL does not generate a mask. In this case, SPUT cannot be used after the GET. For this reason, the SPUT, SGET, and SET macros should be used cautiously. ERRIF pseudo instructions should be used whenever more than one field is assumed to be in the same word.

NOTE

Register A0 should be used with caution in these opdefs. It always takes the value of 0. Register S0 takes the value of 0 in some situations and of 2**63 in other situations. S0 should not be used unless it is explicitly stated in the parameter description that S0 is legal.

GET - Fetch contents of a field

The GET opdef provides a means of conveniently fetching a field entry (defined by a FIELD macro) from a table. The field specified as *name* is transferred right-justified into S*i*.

Format:

Location	Result	Operand
loc	GET,S*i*	S*j*&S*k*,*name*,A*l*

loc Optional location field

S*i* S register to receive value being fetched from field

S*j* S register to be used by GET to hold full word from which field is extracted. S*j* can be the same as S*i*; however, in this case an SPUT opdef cannot follow since S*j* no longer contains the full word.

S*k* S register to be used by GET to hold bit mask of field. S*k* cannot be the same as S*j*. If S*k* is the same as S*i*, SPUT cannot be used in the following code because the mask will no longer be in S*k*.

name Name of field being fetched

A*l* A register containing pointer to base of table or table entry; the contents of this register are unchanged by GET. If register A0 is specified, then W@*name* is used as the absolute address of the word to load. This should not be done except when referencing the JLB.

Example:

Location	Result	Operand	Comment
1	10	20	35
	GET,S1	S2&S3,DPIBN,A1	

Expansion:

```
S.2         W@DPIBN,A.1
S.3         >N@DPIBN
S.3         S.3>S@DPIBN
S.1         S.2&S.3
S.1         S.1>D'64-S@DPIBN-N@DPIBN
```

GETF - Fetch contents of a field

The GETF opdef provides a means of conveniently fetching a field entry (defined by a FIELD macro) from a table. The field specified as *name* is transferred right-justified into S*i* or A*i*

Format:

Location	Result	Operand
	GETF,S*i*	S*j*,*name*,A*k*
	GETF,A*i*	S*j*,*name*,A*k*

S*i* S register to receive value being fetched from field

A*i* A register to receive value being fetched from field

S*j* S register to be used for a mask if necessary

name Name of field being fetched

A*k* A register containing pointer to base of table or entry

Example:

Location	Result	Operand	Comment
1	10	20	35
	GETF,S1	S7,DPIBN,A1	
	GETF,A2	S7,DPIBN,A1	

Expansion:

```
S.1     W@DPIBN,A.1
S.7     <N@DPIBN
S.1     S.1>D'64-S@DPIBN-N@DPIBN
S.1     S.1&S.7
```

PUT - Store data from a register into a field

The PUT opdef provides a means of conveniently storing a field entry in a table.

Format:

Location	Result	Operand
loc	PUT,S*i*	S*j*&S*k*,*name*,A*l*

loc Optional location field

S*i* S register containing value to be stored in field, right-justified; unchanged by PUT.

Sj S register to be used by PUT to hold full word. S0 is legal.

Sk S register to be used by PUT to hold bit mask of field. *Si*, *Sj*, and *Sk* must be unique registers.

name Name of field being stored

Al A register containing pointer to base of table or table entry; the contents of this register are unchanged by PUT. If register A0 is specified, then W@*name* is used as the absolute address of the word to load. This should not be done except when referencing the JLB.

Example:

Location	Result	Operand	Comment
1	10	20	35
	PUT,S1	S2&S3,DPIBN,A1	

Expansion:

```
S.2          W@DPIBN,A.1
S.3          >N@DPIBN
S.3          S.3>S@DPIBN
S.1          S.1<D'64-N@DPIBN-S@DPIBN
S.2          S.1!S.2&S.3
S.1          S.1>D'64-S@DPIBN-N@DPIBN
W@DPIBN,A.1 S.2
```

SET - Pack field value into a register

A series of SET opdefs can pack several fields, one field at a time, into one register. The caller must code a store instruction at the end of a series of SET opdefs. If new information is being packed into portions of an existing word, the SET must follow a GET. If the entire word is changing, the register to receive the newly packed information should be zeroed.

Format:

Location	Result	Operand
loc	SET,*Si*	*Sj*&*Sk*,*name*

loc Optional field location

S*i* S register containing value to be packed into a field of S*j*

S*j* S register in which word is being packed

S*k* S register to be used by SET to hold bit mask for field. S*i*, S*j*, and S*k* must be unique registers.

name Name of field to be packed

Example:

Location	Result	Operand	Comment
1	10	20	35
	SET,S3	S6&S7,JXBLO	

Expansion:

```
S.7     >N@JXBLO
S.7     S.7>S@JXBLO
S.3     S.3<D'64-N@JXBLO-S@JXBLO
S.6     S.3!S.6&S.7
S.3     S.3>D'64-S@JXBLO-N@JXBLO
```

SGET - Fetch contents of a field

When more than one field from a word is being used, SGET rather than GET can be used for second and subsequent fields to reduce the amount of object code generated. SGET can follow GET, PUT, or SPUT on the same word. Use the ERRIF pseudo instruction to verify that the fields are in the same word (such as, ERRIF W@DPIBP,NE,W@DPIBN).

Format:

Location	Result	Operand
loc	SGET,S*i*	S*j*&S*k*,*name*

loc Optional location field

S*i* S register to receive value being fetched from field

S*j* S register assumed to hold full word from which field is extracted (register set by GET, PUT, or SPUT). S*j* can be the same as S*i*; however, an SPUT opdef cannot follow the SGET opdef since S*j* no longer contains the full word.

Sk S register to be used by SGET to hold bit mask of field. *Sk* cannot be the same as *Sj*. If *Sk* is the same as *Si*, SPUT cannot be used in the following code. The mask will no longer be in *Sk*.

name Name of field being fetched

Example:

Location	Result	Operand	Comment
1	10	20	35
	SGET,S1	S2&S3,DPIBP	

Expansion:

```
S.3     >N@DPIBP
S.3     S.3>S@DPIBP
S.1     S.2&S.3
S.1     S.1>D'64-S@DPIBP-N@DPIBP
```

SPUT - Store data from a register into a field

SPUT resembles PUT but can be used after a GET or SGET for the same field to generate less object-time code.

Format:

Location	Result	Operand
loc	SPUT,*Si*	*Sj*&*Sk*,*name*,A*l*

loc Optional field location

Si S register containing value to be stored in field; contents of this register are not changed by SPUT.

Sj S register assumed to contain full word in which field is to be stored (register set by GET)

Sk S register assumed to contain bit mask for field (register set by GET). *Si*, *Sj*, and *Sk* must be unique registers.

name Name of field being stored

A*l* A register containing pointer to base of table or table entry; the contents of this register are the same as for the GET and are unchanged by the PUT.

Example:

Location	Result	Operand	Comment
1	10	20	35
	SPUT,S1	S2&S3,DPIBN,A1	

Expansion:

```
        S.1          S.1<D'64-S@DPIBN-N@DPIBN
        S.2          S.1!S.2&S.3
        S.1          S.1>D'64-S@DPIBN-N@DPIBN
        W@DPIBN,A.1  S.2
```

COMPLEX OPDEFS

The opdefs used in complex field manipulation are:

- CGET Fetches the contents of a field into a register
- CPUT Stores the contents of a register into a field

Due to hardware restrictions and an attempt to save on register usage, the short forms of these macros are not provided.

NOTE

These macros cannot be used in reentrant code.

CGET - Fetch contents of a field into a register

The CGET opdef provides a means of conveniently fetching a field value that was defined by the CFIELD macro. This opdef recognizes when it must obtain the value from more than one 64-bit word and appropriately generates the machine instructions to do so.

Format:

Location	Result	Operand
loc	CGET,*field*,S*i*	S*j*,S*k*,A*l*

loc Optional field location

field A 1- to 6-character name of the field to be fetched

S*i* S register to receive the field value; right-justified. S0 is not a valid register.

S*j* S register to be used as a scratch register. The contents of this register, upon exit, have no meaningful value. S*i* and S*j* must be unique registers. S0 is not a valid register.

S*k* A register to be used as a scratch register. S0 is not a valid register.

A*l* A register containing the base address of the table or table entry. This register is left unchanged. A*l* and A*k* must be unique registers.

CPUT - Store contents of a register into a field

The CPUT opdef conveniently stores a field value into a table or table entry. The field must have been defined using the CFIELD macro or CSBFIELD macro. CPUT recognizes and automatically generates the additional machine instructions for storing fields that span 64-bit word boundaries.

Format:

Location	Result	Operand
loc	CPUT,S*i*,*field*	S*j*,S*k*,A*l*

loc Optional field location

S*i* S register containing the right-justified value to store. The contents of the registers remain unchanged. S0 is not a valid register.

field Name of the field in which to store the value

S*j* S register to be used as a scratch register. The contents of this register, upon exit, have no meaningful value. S*j*, S*k*, and S*i* must be unique registers. S0 is not a valid register.

Sk S register to be used as a scratch register. The contents of this register, upon exit, have no meaningful value. Sk, Si, and Sj must be unique registers. S0 is not a valid register.

Al A register containing the base address of the table or table entry. This register is not destroyed by CPUT.

SEMAPHORE MANIPULATION MACROS

The semaphore manipulation macros are designed to allow the user to symbolically define semaphore names and to use the CRAY X-MP hardware semaphores. These macros generate executable code on CRAY X-MP systems. Code generation is conditionally controlled by use of the CPU= parameter on the CAL control statement. Executable code is produced when CPU='CRAY-XMP'.

NOTE

CRI has reserved the first 16 semaphore bits (0-15) for use by Cray software. The user should use only semaphore bits 16-31.

The semaphore manipulation macros include:

- DEFSM Defines semaphore name
- CLRSM Unconditionally clears a semaphore
- GETSM Gets current status of semaphore bit
- SETSM Unconditionally sets a semaphore
- TEST$SET Tests and sets semaphore

DEFSM - DEFINE SEMAPHORE NAME

The DEFSM macro defines a symbol to be used for a specific semaphore bit. This macro accepts a user-specified symbol of up to 5 characters and a number of a semaphore bit from 0 to 31. If no number is specified for the semaphore bit to assign, the semaphore following the last

previous bit defined is assigned. For example, if semaphore bit 17 is assigned and a DEFSM with no number specified is used, semaphore bit 18 is assigned. If a DEFSM with no number specified is the first DEFSM to be used, semaphore bit 16 is assigned. Checking is done to ensure that a semaphore has not already been assigned with a different name. This macro only defines symbols and generates no executable code.

Format:

Location	Result	Operand
name	DEFSM	*number*

name Symbol to be assigned to the semaphore bit. This must be 5 characters or less.

number Number of the semaphore bit to be assigned. This must be a number in the range 0 to 31. If no number is specified, the semaphore bit following the last previous bit defined is assigned. If this is the first use of a DEFSM macro, semaphore bit 16 is assigned.

Example:

Location	Result	Operand	Comment
1	10	20	35
BWAIT	DEFSM		Defines Busy/Wait; semaphore 16
CODLK	DEFSM		Defines semaphore 17 for code lock
IOLCK	DEFSM	30	Defines semaphore 30 for I/O lock
SIGNL	DEFSM		Defines semaphore 31 as hardware signal

CLRSM - UNCONDITIONALLY CLEAR A SEMAPHORE, WITHOUT WAITING

The CLRSM macro causes a semaphore clear instruction to be generated for the CRAY X-MP computer. This instruction unconditionally clears a semaphore bit to 0 without regard to its prior state. The semaphore to clear must have been defined using the DEFSM macro.

Format:

Location	Result	Operand
	CLRSM	*name*

name Name of the semaphore to clear. This name must have been defined using the DEFSM macro.

Example:

Location	Result	Operand	Comment
1	10	20	35
	CLRSM	CODLK	Allows other processor to enter

GETSM - GET CURRENT STATUS OF SEMAPHORE BIT

The GETSM macro gets the current state of a semaphore bit and returns the state in the sign bit of an S register (the remainder of the S register is cleared). The GETSM macro reads the semaphore register on the CRAY X-MP computer and shifts to place the desired semaphore bit in the sign bit of an S register.

NOTE

No protection is used for this operation. The bit being interrogated might be changed by the other processor before the user can check its value. The GETSM macro should not be used in place of a TEST$SET macro. GETSM is intended for signaling purposes.

Format:

Location	Result	Operand
	GETSM	*name*,REG=*reg*

name Name of the semaphore to interrogate. This name must have been defined using the DEFSM macro.

REG=*reg* S register in which to return the semaphore value. The current value of the semaphore is returned in the sign bit with the remainder of the word set to 0. If no REG= clause is specified, the result is returned in S0.

Example:

Location	Result	Operand	Comment
1	10	20	35
	GETSM	SIGNL	Gets signal value in S0

SETSM - UNCONDITIONALLY SET A SEMAPHORE, DO NOT WAIT

The SETSM macro causes a semaphore set instruction to be generated for the CRAY X-MP computer. This instruction unconditionally sets a semaphore bit to 1 without checking or waiting for its prior state. The semaphore to set must have been defined using the DEFSM macro.

Format:

Location	Result	Operand
	SETSM	*name*

name Name of the semaphore to set. This name must have been defined using the DEFSM macro.

Example:

Location	Result	Operand	Comment
1	10	20	35
	SETSM	BWAIT	Forces Busy/Wait to Busy state

TEST$SET - TEST SEMAPHORE AND WAIT IF SET, SET IF CLEAR

The TEST$SET macro generates a CRAY X-MP semaphore test and set instruction. The test and set instruction causes a processor to hold issue if the semaphore is set and to set the semaphore if it is clear. This macro generates test and set instructions for the semaphores defined by the DEFSM macro.

Format:

Location	Result	Operand
	TEST$SET	*name*

name Name of the semaphore to test and set. This name must have been defined using the DEFSM macro.

Example:

Location	Result	Operand	Comment
1	10	20	35
	TEST$SET	CODLK	Forces single path for code segment

CAL EXTENSION MACROS AND OPDEFS

The DIVIDE opdef, PVEC macro, $CYCLES macro, $DECMIC macro, and RECIPCON macro are described in this subsection.

DIVIDE OPDEF - PROVIDE A PRECODED DIVIDE ROUTINE

The DIVIDE opdef enhances the instruction repertoire of the CAL assembler by providing a precoded divide routine accessible through a call in machine language syntax.

****

CAUTION

The contents of registers S*j* and S*k* are destroyed.

****

Format:

Location	Result	Operand
symbol	S*i*	S*j*/FS*k*

Expansion:

symbol	S*i*	/HS*k*
	S*j*	S*j**FS*i*
	S*k*	S*k**IS*i*
	S*i*	S*j**FS*k*

symbol Valid location field name; optional.

S*i* S register that gets the result

S*j* S register containing the dividend; S*j* and S*i* must be unique registers.

FS*k* S register containing the divisor; S*k* and S*i* must be unique registers.

PVEC MACRO - PASS ELEMENTS OF VECTOR REGISTER TO SCALAR ROUTINE

The PVEC macro is used in pseudo-vectorized arithmetic routines. The macro generates a loop to get arguments from vector registers, passes these to a scalar routine and stores the results in vector registers or vector storage locations. This has the effect of taking vectors as input and producing vector results using a scalar routine. This is necessary when no purely vectorized versions of arithmetic algorithms exist. This macro replaces the VECA, VECB, and VECC routines.

Format:

Location	Result	Operand
	PVEC	*routine*,vec_1,vec_2,vec_3,vec_4,LENGTH=*length*

routine Name of the scalar routine to call

vec_1 Vector register or the name of a storage area for a vector (the storage area can be on the stack). The values in the register or storage area are assigned sequentially to register S1 and passed to the scalar routine.

vec_2 Vector register or the name of a storage area for a vector (the storage area can be on the stack). The values in the register or storage area are assigned sequentially to register S2 and passed to the scalar routine.

vec_3 Vector register or the name of a storage area for a vector (the storage area can be on the stack). The values in the register or storage area are assigned sequentially to register S3 and passed to the scalar routine.

vec_4 Vector register or the name of a storage area for a vector (the storage area can be on the stack). The values in the register or storage area are assigned sequentially to register S4 and passed to the scalar routine.

NOTE

The return values from the scalar routines correspond with the vector registers specified. For example, if only vec_1 is specified, the return value of S1 from the scalar routine is stored. If vec_1 and vec_2 are specified, the values of S1 and S2 returned by the scalar routine are both stored.

LENGTH=*length*

The length of the vector to be processed. *length* can be passed as a constant or in an A, S, or B register. (Registers A0 and S0 should not be used). If *length* is not specified, the length is assumed to be the value of the VL register when the macro is invoked.

NOTE

The macro assumes that if vector registers are used as input, they are preserved across the call to the scalar routine. If the scalar routine being called uses any of the vector registers, these registers must be saved to memory and the name of the save area passed to the macro. The macro does not generate storage space for the vector registers. The user must define this space and store the vectors before invoking the macro.

$CYCLES MACRO- GENERATE TIMING-RELATED SYMBOLS AND CONSTANTS

The $CYCLES macro generates timing-related symbols and constants. This macro generates the number of CPU clock periods or cycles in a given

interval. When the system is assembled, the configuration parameters in the CONFIG@P deck in COSTXT determine the number of cycles in each interval specified by a $CYCLES macro invocation.

Format:

Location	Result	Operand
label	$CYCLES	INTERVAL=*interval*,UNITS=*units*,TYPE=*type*

label Name of symbol or constant generated

INTERVAL=*interval*
Nonzero integer interval

UNITS=*units*
Units from the following set:

NSEC	Nanoseconds
USEC	Microseconds
MSEC	Milliseconds
SEC	Seconds
MIN	Minutes
HOUR	Hours
HOURS	Hours (alternate form of HOUR)
DAY	Days
DAYS	Days (alternate form of DAY)

TYPE=*type* SYMBOL if a symbol is to be generated.
CONSTANT if an integer constant is to be generated.
FPCONST if a floating-point constant is to be generated.
RECIP if a floating-point reciprocal is to be generated.

$DECMIC MACRO - CONVERT A POSITIVE INTEGER TO A MICRO STRING

$DECMIC converts any positive integer that can be held in a CAL symbol to a MICRO string.

Format:

Location	Result	Operand
micname	$DECMIC	*constant*

micname Micro name, the same as in the DECMIC pseudo-op

constant Positive integer constant between 0 and $(2^{63-1})/10$

Example:

Location	Result	Operand	Comment
1	10	20	35
LONGMIC	$DECMIC	D'1234567890123	Converts to micro form
	DATA	'"LONGMIC"'Z	Use of micro
	DATA	'1234567890123'Z	Micro expansion

NOTE

The $DECMIC macro does not accept a count parameter. The generated micro is as long as needed to express the value of the constant.

RECIPCON MACRO - GENERATE FLOATING-POINT RECIPROCALS

The RECIPCON macro generates floating-point reciprocals of integer constants. These reciprocals are typically used in floating-point multiply operations, which are much faster than divide sequences on Cray mainframes.

Format:

Location	Result	Operand
tag	RECIPCON	*icon*

tag Location symbol to be used when generating the constant.

icon Integer constant. The floating-point reciprocal of this constant is generated at location *tag*.

NOTE

The method used to generate the reciprocal is:

itemp = 1.0E18/*icon* form reciprocal * 1E18
tag = FLOAT(*itemp*)*1.0E-18 form reciprocal

The net effect is to form 1.0/FLOAT(*icon*). The reciprocal is accurate to MIN(18-LOG10(*icon*), 14) decimal digits.

Example:

Location	Result	Operand	Comment
1	10	20	35
TENTH	RECIPCON	D'10	Forms floating-point 1/10
	.....		
	S1	FPNUM,0	(S1) = any floating-point number
	S2	TENTH,0	(S2) = floating-point 1/10
	S3	S1*FS2	(S3) = floating-point (num/10.0)

COS-DEPENDENT MACROS 7

Included with the Cray Operating System is a set of macro and opdef instructions available only when programming in the Cray Assembly Language (CAL). These instructions are processed by the assembler using macro definitions defined in the system text, COSTXT.

These COS-dependent macros and opdefs are intended for internal system users only. Users other than system programmers should avoid using them.

SYSTEM TASK OPDEFS

The system task opdefs include:

- ERDEF Generates error processing entries in the Exchange Processor
- GETDA Obtains first DAT page address
- GETNDA Obtains next DAT page address

ERDEF - GENERATE ERROR PROCESSING ENTRIES IN THE EXCHANGE PROCESSOR

The ERDEF opdef generates entries for the error processing table in the Exchange Processor (EXP) at assembly time. The entries are used during abort processing.

Format:

Location	Result	Operand
	ERDEF	*addr,fatal,class*[,DN=YES][,REPR=NO]

addr Message address for this error

fatal Bit number in the JTFEFW field of the JTA other than 0 indicates a fatal error has occurred; it is 0 if the error is nonfatal.

class Major error class. To interpret the value of this table entry, shift a rightmost 1 bit to the left as many times as specified in the table entry. If the table entry is 2, the value is 2^n where n=*class*.

DN=YES Dataset name to be included with the message; otherwise, the parameter is omitted.

REPR=NO Error is not reprievable; otherwise, the parameter is omitted.

GETDA - OBTAIN FIRST DAT PAGE ADDRESS

The GETDA opdef obtains the STP-relative address of the first DAT page, using either the DNT address, or the DNT and JTA addresses. The GETDA opdef has two formats: a short form if both the Dataset Name Table (DNT) and the JTA addresses are known, and a long form if only the DNT address is known.

****

CAUTION

This opdef destroys the contents of A0.

****

Format (long form):

Location	Result	Operand
	GETDA,A*i*,A*j*	S*k*&S*l*,A*m*,A*n*

A*i* A register to receive the STP-relative address of the DAT; cannot be A0. A*i* is 0 if no DATs are associated with the DNT.

A*j* A register to receive the STP-relative address of the JTA if the DAT resides in the JTA; cannot be A0.

Sk S register to be used by GETDA; cannot be S0.

Sl S register to be used by GETDA; cannot be S0.

Am A register to be used by GETDA; cannot be A0.

An A register containing STP-relative address of the DNT; cannot be A0.

Example:

Location	Result	Operand	Comment
1	10	20	35
	GETDA,A1,A2	S3&S4,A5,A6	

Expansion:

```
          S.3       W@DNDAT,A.6
          A.1       S.3
          A.0       A.1
          JAP       GN1
          SGET.S.4  S.3&S.4,DNJORD
          A.2       S.4
          A.5       LE@JXT
          A.2       A.2*A.5
          A.5       B@JXT
          A.2       A.2+A.5
          A.2       W@JXJTA,A.2
          A.1       A.2-A.1
GN1       =         *
```

Format (short form):

Location	Result	Operand
	GETDA,A*i*	A*j*,A*k*

A*i* A register to receive the STP-relative address of the DAT; cannot be A0. A*i* is 0 if no DATs are associated with the DNT.

A*j* A register containing the STP-relative address of the DNT; cannot be A0.

A*k* A register containing the STP-relative address of the JTA; cannot be A0.

Example:

Location	Result	Operand	Comment
1	10	20	35
	GETDA,A1	A2,A3	

Expansion:

```
          A.1         W@DNDAT,A.2
          A0          A.1
          JAP         GD1
          A.1         A.3-A.1
GD1       =           *
```

GETNDA - OBTAIN NEXT DAT PAGE ADDRESS

The GETNDA opdef obtains the STP relative address of the next DAT page, using either the current DAT page address, or the current DAT page and JTA addresses. The GETNDA has two formats: a long form, if only the current DAT page address is known, and a short form if both the current DAT page and the JTA addresses are known.

****

CAUTION

This opdef destroys the contents of A0.

****

Format (long form):

Location	Result	Operand
	GETNDA,A*i*,A*j*	S*k*&S*l*,A*m*,A*n*

A*i* — A register to receive the STP-relative address of the next DAT page; cannot be A0. A*i* is 0 if there are no further DAT pages.

A*j* — A register to receive the STP-relative address of the JTA if the DAT is in the JTA; cannot be A0.

S*k* — S register to be used by GETNDA; cannot be S0.

S*l* — S register to be used by GETNDA; cannot be S0.

Am — A register to be used by GETNDA; cannot be A0.

An — A register containing the STP-relative address of the current DAT page; cannot be A0.

Example:

Location	Result	Operand	Comment
1	10	20	35
	GETNDA,A1,A2		S3&S4,A5,A6

Expansion:

```
          S.3         W@DADAT,A.6
          A.1         S.3
          A0          A.1
          JAP         GN1
          SGET,S.4    S.3&S.4,DAJORD
          A.2         S.4
          A.5         LE@JXT
          A.2         A.2*A.5
          A.5         B@JXT
          A.2         A.2+A.5
          A.1         A.2-A.1
GN1       =           *
```

Format (short form):

Location	Result	Operand
	GETNDA,A*i*	A*j*,A*k*

A*i* — A register to receive the STP-relative address of the next DAT page; cannot be A0. A*i* will be 0 if there are no further DAT pages.

A*j* — A register containing the STP-relative address of the current DAT page; cannot be A0.

A*k* — A register containing the STP-relative address of the JTA; cannot be A0.

Example:

Location	Result	Operand	Comment
1	10	20	35
	GETNDA,A1	A2,A3	

```
Expansion:
            A.1         W@DADAT,A.2
            A0          A.1
            JAP         GD1
            A.1         A.3-A.1
  GD1       =           *
```

OVERLAY MANAGER TASK MACROS

The Overlay Manager task macros define new overlays in the operating system. The procedure for using the overlay macros is described below.

1. Define the overlay in COSTXT using the OVLDEF macro to associate an overlay identifier with the overlay and its entry points.

2. Define the overlay in the Startup task by using the DEFINOVL macro. All DEFINOVL macros are located at the end of Startup, creating a list of all overlays that exist in the system. DEFINOVL macros are also used by Startup when searching for overlays before creating the Overlay Directory Table (ODT) and moving the overlays to disk.

3. Include an OVERLAY macro after the IDENT statement for code that is to be considered an overlay. The placement of the OVERLAY macro informs CAL to build the appropriate Program Description Table (PDT) and Block Relocation Table (BRT), which in turn cause the loader to satisfy only external references.

4. Use the following macros to control loading and exiting from overlays:
   - CALLOVL
   - GOTOOVL
   - LOADOVL
   - RTNOVL

The Overlay Manager task macros include:

- CALLOVL Requests Overlay Manager task to load
- DEFINOVL Generates a list of modules
- DISABLE Prevents use of current memory resident copy
- GOTOOVL Requests Overlay Manager task to load
- LOADOVL Requests an initial overlay load

- OVERLAY Defines a module as a system overlay
- OVLDEF Defines overlay name
- RTNOVL Signals completion of overlay execution

CALLOVL - REQUEST OVERLAY MANAGER TASK TO LOAD

The CALLOVL macro requests the Overlay Manager task (OVM) to load a secondary overlay. CALLOVL makes an entry into the associated Overlay Call Stack to reload the calling overlay when the overlay being called terminates. A CALLOVL macro can appear only within an overlay module. A limit of nine nested CALLOVL requests is permitted.

****

CAUTION

This macro destroys the contents of register A3.

****

Format:

Location	Result	Operand
	CALLOVL	OVL=*id*,RTN=*addr*

OVL=*id* Overlay name of the form OV$*name*; this is a required parameter.

RTN=*addr* Parcel address where control is transferred when the called overlay completes execution. If omitted, the parcel following the macro expansion is used.

DEFINOVL - GENERATE A LIST OF MODULES

The DEFINOVL macro generates a list for Startup of the modules to be located in memory following the end of the Startup code and to be written to the system overlay dataset. One DEFINOVL macro must be present for each such module and no DEFINOVL macro can exist for which Startup is unable to locate a matching module.

Format:

Location	Result	Operand
name	DEFINOVL	

name 1- to 5-character overlay module name

DISABLE - PREVENT USE OF CURRENT MEMORY-RESIDENT COPY

The DISABLE macro prevents the Overlay Manager task (OVM) from using the current memory-resident copy of an overlay when load requests exist for that overlay. The DISABLE macro can appear only within an OVERLAY module.

Format:

Location	Result	Operand
	DISABLE	

GOTOOVL - REQUEST OVERLAY MANAGER TASK TO LOAD

The GOTOOVL macro requests the Overlay Manager task (OVM) to load a secondary overlay. Control is not returned to the calling overlay upon completion of the called overlay. A GOTOOVL macro can appear only within an OVERLAY module. No limit is placed on the number of successive GOTOOVL calls.

****

CAUTION

This macro destroys the contents of register A3.

****

Format:

Location	Result	Operand
	GOTOOVL	OVL=*id*

OVL=*id* Overlay name of the form OV$*name*; this is a required parameter.

LOADOVL - REQUEST AN INITIAL OVERLAY LOAD

The LOADOVL macro requests an initial overlay load. An assembly error occurs if a LOADOVL is encountered within an OVERLAY module.

****

CAUTION

This macro destroys the contents of register A3.

****

Format:

Location	Result	Operand
	LOADOVL	OVL=*id*,RTN=*addr*

OVL=*id* Overlay name of the form OV$*name*; this is a required parameter.

RTN=*addr* Parcel address where control is transferred when the overlay completes execution. If omitted, the parcel following the macro expansion is used.

OVERLAY - DEFINE A MODULE AS A SYSTEM OVERLAY

The OVERLAY macro defines a module as a system overlay. It causes the module type in the loader-directed Program Description Table (PDT) to be set to relocatable overlay and generates symbolic names for the overlay and any specified entry points.

Format:

Location	Result	Operand
name	OVERLAY	*entry-point-list*

name 1- to 5-character overlay identifier. A symbol of the form OV$*name* is defined. All references to the overlay use the supplied name. Word 0 of the overlay contains the overlay name and ID for verification and debugging purposes.

entry-point-list
Optional list of symbols defined within the overlay that are to be regarded as entry points, enclosed in parentheses. If none are present, the Overlay Manager uses a default entry at parcel 0, word 1.

OVLDEF - DEFINE OVERLAY NAME

The OVLDEF macro defines the name the system uses to identify the overlay and associates the overlay identifier with that name. A symbol of the form OV$*name* is defined and a symbol of the form OE$*eptname* is defined for each name present in the optional *entry-point-list*. If entry-point names are given, they must match the names supplied in the corresponding OVERLAY macro in number and position. A difference in entry-point definitions results in an assembly error or a failure to successfully construct the Overlay Directory Table during Startup.

Format:

Location	Result	Operand
name	OVLDEF	*entry-point-list*

name 1- to 5-character overlay identifier. A symbol of the form OV$*name* is defined. All references to the overlay use the supplied name. Word 0 of the overlay contains the overlay name and ID for verification and debugging purposes.

entry-point-list
Optional list of symbols defined within the overlay that are to be regarded as entry points, enclosed in parentheses. If none are present, the Overlay Manager uses a default entry at parcel 0, word 1.

RTNOVL - SIGNAL COMPLETION OF AN OVERLAY EXECUTION

The RTNOVL macro signals the completion of an overlay execution and returns control to the caller. The Overlay Manager task (OVM) determines which Overlay Call Stack to use in determining the return address. The RTNOVL macro can only occur in an OVERLAY module.

****

CAUTION

This macro destroys the contents of register A3.

****

Format:

Location	Result	Operand
	RTNOVL	

MESSAGE PROCESSOR MACRO

The message processor macros include LOGMSGM:

LOGMSGM - CONSTRUCT THE LGR CONTROL WORD

The LOGMSGM macro sets up a fixed call to the Message Processor task or sets up a skeleton for the call. In the second case, the fields must be defined prior to the request.

Format:

Location	Result	Operand
label	LOGMSGM	LOG=*log*,OVRD={OFF / ON},CLASS=*class*,TYPE=*type*,
		SUBTYPE=subtype,LENGTH=*length*,ADDRESS=*address*

label A 1- to 7-character identifier

LOG=*log* Specifies message locations with the following codes:

NOLOG Sets up an empty skeleton in log field

USER Writes message to user log only

SYS Writes message to system log only

BOTH Writes message to both user and system logs

OVRD={OFF / ON}

Determines if a message is issued, regardless of the echo status. OVRD=OFF sets up an empty skeleton; sets the override bit off. OVRD=ON overrides the echo status of a message class; sets the override bit on.

CLASS=*class*

Specifies message class with the following codes:

NOCL Sets up an empty skeleton in the class field

JCL Specifies JCL class message

ABORT Specifies abort class message

TYPE=*type* Specifies message type. NOTYP sets up an empty skeleton in the type field. See the COS EXEC/STP/CSP Internal Reference Manual, CRI publication SM-0040, for legal values of *type*. Symbolic values are listed in deck COMLG of the COSPL program library.

SUBTYPE=*subtype*

Specifies message subtype. NOSUB sets up an empty skeleton in the subtype field. See the COS EXEC/STP/CSP Internal Reference Manual, CRI publication SM-0040, for legal values of *subtype*. Symbolic values are listed in deck COMLG of the COSPL program library.

LENGTH=*length*

Specifies message length. *length* must be defined prior to the macro call. LENGTH=NOLEN sets up an empty skeleton in the length field.

ADDRESS=*address*

Specifies address of message to be written. *address* must be defined prior to the macro call. ADDRESS=NOADDR sets up an empty skeleton in the address field.

Example:

```
Location  Result    Operand                  Comment
1         10        20                       35

MESSAGE     BSS     0
            DATA    'THIS MESSAGE COMES FROM CSP'
LEN         =       W.@*-MESSAGE
WORD1       LOGMSGM LOG=SYS,OVRD=ON,CLASS=JCL,TYPE=ASCII,
                    SUBTYPE=CSP,LENGTH=LEN,ADDRESS=MESSAGE
```

In this example, the LOGMSGM macro builds a request to the Message Processor, requesting an ASCII type, CSP subtype, JCL class message at address MESSAGE to be written to the systemlog only. OVRD=ON causes the JCL class message to be issued, regardless of echo status.

COS INTERNAL SUBROUTINE LINKAGE MACRO

The COS internal subroutine linkage macros include $SUB:

$SUB - DEFINE A SUBROUTINE ENTRY POINT

The $SUB macro defines a subroutine entry point, and provides for the saving and restoring of registers using a software stack. Register A7 is reserved as the stack pointer within subroutines using the $SUB calling sequence. $RETURN is specified as the exit point for the subroutine.

Format:

```
Location  Result  Operand
TAG       $SUB    SREG=a:b,AREG=c:d
```

TAG Name of the subroutine; 1-5 characters. This name is embedded in a CON statement in the subroutine prologue, and is merged with the return address in a word on the register stack.

SREG=*a*:*b* Specifies the S registers that are saved by the macro on entry and restored on exit. For example, SREG=1 saves and restores S1. SREG=2:4 saves and restores S2, S3, and S4. Registers S0, S6, and S7 cannot be specified by the SREG parameter. The default is to save no S registers.

AREG=*c*:*d* Specifies the range of A registers that are saved by the macro on entry and restored on exit. For example, AREG=3 saves and restores A3. AREG=2:4 saves and restores A2, A3, and A4. Registers A0 and A7 cannot be specified by the AREG parameter. The default is to save no A registers.

NOTE

Subroutine calls can be made within $SUB subroutines, provided the called routines do not alter A7.

DATASET LOCKING MACRO

The COS dataset locking macro is DSPLOCK.

DSPLOCK - SET OR CLEAR LOCK IN DATASET PARAMETER AREA

The DSPLOCK macro is used by I/O routines to ensure single-threaded I/O on a dataset while multitasking. The Dataset Parameter Table (DSP) must be locked on entry to a routine, and unlocked on exit from a routine.

Format:

Location	Result	Operand
	DSPLOCK	*action*,DSP=*addr*

action Specifies the action to be taken:

SET Sets the DSP lock

CLEAR Clears the DSP lock

FLUSH Unconditionally clears the DSP lock on an error condition

NAME=*name* Name of the calling routine.

DSP=*addr* An A register which contains the STP-relative address of the DSP.

STRUCTURED PROGRAMMING MACROS 8

The structured programming macros in this section are in $SYSTXT and are available for use in programs written in CAL. Unlike the majority of macros in $SYSTXT, these are independent of the operating system.

Several of the structured programming macros presented in this section use conditional expressions. Therefore, an explanation of conditions precedes the macros descriptions.

CONDITIONS

Table 8-1 describes the conditions for structured programming macros that use such conditions.

Table 8-1. Conditions

Condition	Meaning
Conditions on A0 and S0	
AZ...	(A0) equal to 0
AN...†	(A0) not equal to 0
AP...	(A0) greater than or equal to 0
AM...	(A0) less than 0
SZ...	(S0) equal to 0
SN...†	(S0) not equal to 0
SP...	(S0) greater than or equal to 0
SM...	(S0) less than 0

† Condition means not equal to 0; conditions beginning with Ne, NE, Ng, or NG are not allowed, because they strongly suggest negative instead of nonzero.

Table 8-1. Conditions (continued)

Condition	Meaning
Conditions on A and S registers	
A_n[=*operand*],Z...	(A_n) equal to 0
A_n[=*operand*],N...†	(A_n) not equal to 0
A_n[=*operand*],P...	(A_n) greater than or equal to 0
A_n[=*operand*],M...	(A_n) less than 0
S_n[=*operand*],Z...	(S_n) equal to 0
S_n[=*operand*],N...†	(S_n) not equal to 0
S_n[=*operand*],P...	(S_n) greater than or equal to 0
S_n[=*operand*],M...	(S_n) less than 0
Bit set conditions	
constant,IN,S*m*[=*operand*]	True if bit number *constant* is set in register S*m*. Bits are numbered sequentially, with the sign bit being 0.
S*n*[=*operand*],ALLIN,S*o*[=*operand*]	True if every bit set in register S*n* is also set in register S*o*.
S*n*[=*operand*],ONEIN,S*o*[=*operand*]	True if at least 1 bit set in S*n* is also set in S*o*.
Compound conditions	
NOT,(*cond*)	*cond* is not true
(*cond*$_1$),AND,(*cond*$_2$)	*cond*$_1$ and *cond*$_2$ are both true
(*cond*$_1$),OR,(*cond*$_2$)	*cond*$_1$ is true, *cond*$_2$ is true, or both are true.
Relational conditions	
S_n[=*operand*], EQ / NE / GT / GE / LT / LE	S_m[=*operand*]

† Condition means not equal to 0; conditions beginning with Ne, NE, Ng, or NG are not allowed, because they strongly suggest negative instead of nonzero.

Table 8-1. Conditions (continued)

Condition	Meaning
Relational conditions (continued)	
A_n[=*operand*],	EQ A_m[=*operand*] NE GT GE LT LE

† Condition means not equal to 0; conditions beginning with Ne, NE, Ng, or NG are not allowed, because they strongly suggest negative instead of nonzero.

The ellipsis (...) means any number of letters (including zero). For example, AZ means the same as AZero.

m, *n* and *o* are any integers from 0 to 7 inclusive; the portion in brackets is optional.

Specifying an *operand* causes generation of an instruction that sets the indicated register to the *operand*'s value. If an *operand* contains embedded commas or blanks the entire assignment must be enclosed in parentheses.

Example 1:

A3,Minus is true if (A3) is less than 0; **A0=PS2,Zero** is true if the population count of (S2) is 0; **(S0=JOE,0),Plus** is true if the content of memory word JOE is positive.

Example 2:

D'63,IN,S0=T.JOE is true if the content of T.JOE is odd;
S2=<3,ALLIN,S3 is true if the last 3 (low order) bits of S3 are set;
S1,ONEIN,(S2=ERROR,0) is true if any bit set in S1 is also set in memory word ERROR.

constant is any CAL expression yielding an integer constant.

cond, $cond_1$, and $cond_2$ are any conditions, simple or compound. The parentheses are required.

Example:

NOT,(S1,EQ,S2) is true if (S1) is not equal to S2; (AMinus),OR,(SMinus) is true if (A0) is less than 0 or (S0) is less than 0 or both; and **((A1,GE,A2='A'R),AND,(A1,LE,A2='Z'R)),OR,((A1,GE,A2='a'R),AND,(A1,LE,A2='z'R))** is true if A1 contains a letter.

MACRO DESCRIPTIONS

The following macros are in $SYSTXT and are available for use in programs written in CAL. Unlike the majority of macros in $SYSTXT, these are independent of the operating system. The macros are described in the following order.

$GOTO
$GOSUB
$IF, $ELSEIF, $ELSE, and $ENDIF
$JUMP
$LOOP, $EXITLP, and $ENDLOOP
$RETURN
$SUBR

$GOTO MACRO - COMPUTES A GOTO STATEMENT

The $GOTO macro offers CAL users a computed GO TO statement.

NOTE

Unlike the 1-based FORTRAN computed GO TO, this GO TO statement is 0-based.

Format:

Location	Result	Operand
	$GOTO	A*i*,(*label*$_0$, *label*$_1$...*label*$_n$),A*j*

Register A*i* is a scratch register, and register A*j* holds a value that determines to which label the jump takes place. For instance, if A*j*=1 the jump is to label$_1$. If A*j* is greater than *n*, no jump takes place, and control falls through to the next instruction.

$GOSUB MACRO - CALL LOCAL SUBROUTINE

The $GOSUB macro calls a subroutine that is local to the current assembly module. A CALL or CALLV macro should be used when the subroutine being called is external. This macro differs from the CALL and CALLV macros in that the routine being called is not declared external by the macro and the call can be made conditionally. This macro does not use the standard CFT subroutine linkage and the call is not reported in the traceback if an error occurs. Only a return jump is generated if no conditions are specified for this macro.

Format:

Location	Result	Operand
	$GOSUB	*label,cond*

label Name of the entry point of the local subroutine. This should be declared using a $SUBR macro.

cond Conditional expression that must be true for the call to be executed. Refer to table 8-1. This expression is optional.

$IF, $ELSEIF, $ELSE, AND $ENDIF MACROS - CONDITIONAL MACROS

The $IF macro operates in the same manner as the similar structure in FORTRAN when used with the attendant $ELSEIF, $ELSE, and $ENDIF macros. The $ELSEIF and $ELSE macros are optional. If both are included, an $ELSEIF macro cannot follow an $ELSE macro. At most, one $ELSE macro may be used between an $IF macro and its $ENDIF macro.

Format:

Location	Result	Operand
	$IF	*cond*
	$ELSEIF	*cond*
	$ELSE	
	$ENDIF	

See table 8-1 for the conditions that can be used with $IF or $ELSEIF.

$IF groups can be nested within other $IF groups up to a level of 10 deep.

The value of an $IF or $ELSEIF condition is treated as either true or false. If true, the block that follows is executed; if false, it is skipped. The $ELSE macro, if present, must follow any $ELSEIF macros that belong to the same $IF group. Within each $IF group, no more than one block is executed (once a block is executed, the remaining blocks in the same $IF group are skipped). If none of the blocks in a group have been executed when an $ELSE macro is encountered, then the $ELSE block is executed if present. A block can be null, that is, it can contain no statements to be executed.

Examples of conditions used with $IF and $ELSEIF follow.

Example 1:

Location	Result	Operand	Comment
1	10	20	35
	$IF	*cond*	
	assembly code		This code is executed if the $IF condition is true.
	$ELSEIF	*cond*	
	assembly code		If the $IF condition is false and the $ELSEIF condition is true, this code is executed.
	$ELSE		
	assembly code		If both of the above conditions are false, this code is executed.
	$ENDIF		

Example 2:

Location	Result	Operand	Comment
1	10	20	35
	$IF	AZ	
	S1	A3	
	S2	A4	
	$ELSEIF	SZ	
	A1	S2	
	A2	S3	
	$IF	AP	
	A1	A2	
	A3	S4	
	$ENDIF		
	$ELSE		
	S1	S2*FS3	
	$ENDIF		

Example 3:

Location	Result	Operand	Comment
1	10	20	35
	$IF	S2,LT,S4	
	A1	2	
	$ELSEIF	A5,GE,A1	
	A1	5	
	$ELSEIF	S1,EQ,S7=123	
	A2	6	
	$ELSE		
	$IF	A2,NE,A5=ABC	
	A3	4	
	$ELSEIF	S5,GT,S7=LABEL	
	A1	5	
	$ENDIF		
	$ENDIF		

$JUMP MACRO - ACCEPT ANY $IF OR $ELSEIF PARAMETERS

The $JUMP macro can be used instead of structured programming macros. An example of such a use is a routine performing extensive validation of entry parameters with a single error return.

$JUMP accepts any parameters that are valid on the $IF-$ELSEIF macros. If the condition holds, flow passes to the destination address. If the condition does not hold, control flow continues on the line following the macro call.

Format:

Location	Result	Operand
	$JUMP	*dest,cond*

dest Destination address. Control flow passes to this address if the specified condition holds.

cond Conditions where control is to pass to the indicated address. This expression is of the same form as conditional expressions described in table 8-1.

Examples:

Location	Result	Operand	Comment
1	10	20	35
	$JUMP	ERROR,SM	
	$JUMP	BADDAY,S1,LT,S2=1	

$LOOP, $EXITLP, AND $ENDLOOP MACROS - DEFINE PROGRAM LOOPS

The $LOOP, $EXITLP, and $ENDLOOP macros define program loops whose iteration conditions are determined at execution time. Each loop begins with a $LOOP macro call which defines the starting parcel address for the loop, and optionally tests a loop top condition. Any inner loop can contain one or more $EXITLP macro calls that cause an immediate exit from the loop structure. Each loop ends with a $ENDLOOP macro call.

$ENDLOOP defines the loop exit address and causes the loop body to be repeated.

The $LOOP and $ENDLOOP statements can be nested to any depth, though nesting more than three deep is not recommended due to the problems in achieving clarity with deep nesting. Note that $EXITLP statements apply only to innermost $LOOP structures.

Format:

Location	Result	Operand
	$LOOP	*cond*
	$EXITLP	*cond*
	$ENDLOOP	

cond This expression is of the same form as conditional expressions described in table 8-1.

Example 1: $EXITLP conditions

Location	Result	Operand	Comment
1	10	20	35
	$EXITLP	SM	Exits if sign of S0 set
	$EXITLP	S1,M	Exits if sign of S1 set
	$EXITLP	S1,NE,S2	Exits if S1<>S2

Example 2: $LOOP-$ENDLOOP structure using $EXITLP to exit loop:

Location	Result	Operand	Comment
1	10	20	35
*	Loops to copy (A1) words from A to B		
*	Destroys A1.		
*	Always moves at least one word.		
*			
	$LOOP		
	A1	A1-1	Decrements word counter
	S1	A,A1	Gets word
	B,A1	S1	Moves it
	$EXITLP	A1,ZR	Exits loop when all words copied
	$ENDLOOP		Repeats body of loop
*			
*	Here when copy has been completed		

Example 3: $LOOP-$ENDLOOP structure with test at top of loop:

```
Location Result     Operand    Comment
1        10         20         35
*
*    Loops to copy a message from X to Y.  Copy terminates
*    on an all-zero word.  Count of words copied is
*    maintained in A7.  If null message, loop never executes.
*
         A7         0          Presets count of words copied
         S7         X,A7       Prefetches 1st word of buffer
         $LOOP      S7,NZ      While word to be copied is
                               nonzero, stores
           Y,A7     S7         in destination buffer
           A7       A7+1       Increments word counter
           S7       X,A7       Prefetches next word for
                               test/copy
         $ENDLOOP              Unconditionally jumps to loop
                               top
*
*    Here when copy has been completed
```

$RETURN MACRO - RETURN FROM LOCAL SUBROUTINE

The $RETURN macro returns from a local subroutine. This macro can also specify the name of a B register that contains the return address. Because this macro is intended to be used with the $SUBR macro the same B register name should be specified on both macros. This macro does not use the standard CFT subroutine linkage and is not reported in the traceback if an error occurs.

Format:

Location	Result	Operand
	$RETURN	*Bregname*

Bregname Name of a B register containing the return address. If this is omitted, then B00 is used for the return. The B register name specified should be the same name as that used on $SUBR macro used to enter this routine and should be defined using a DEFB macro. Note that only the name should be specified and not the full B register designation (for example, RETADDR not B.RETADDR).

$SUBR MACRO - DECLARE LOCAL SUBROUTINE ENTRY POINT

The $SUBR macro declares a local subroutine entry point. This macro can also be used to specify a B register for storing the return address. If a B register is specified to save the return address, then register A0 is used to load the return address from B00 and store it in the specified B register. This macro does not declare the entry point with a CAL ENTRY pseudo-op. This macro also does not use the standard CFT subroutine linkage and is not reported in the traceback if an error occurs.

Format:

Location	Result	Operand
label	$SUBR	*Bregname*

label Name of this entry point of the local subroutine

Bregname Name of a B register to save the return address. If this is omitted, the return address is only contained in B00, which must not be destroyed if a proper return is to be made. The B register name should be defined using the DEFB macro. Note that only the name should be specified and not the full B register designation (for example, RETADDR not B.RETADDR).

APPENDIX SECTION

OUTMODED FEATURES

A

BREG - ASSIGN NAMES TO B REGISTERS (OBSOLETE)

The BREG macro assigns numerical values to symbols for use as B register names. It also checks that no more registers are used than are declared on the ENTER macro. The register names are assigned after any registers used by the calling sequence linkage. This macro must be used only after an ENTER macro. This macro has been replaced by the DEFB macro, which is used before the ENTER macro.

Format:

Location	Result	Operand
name	BREG	

name Symbolic name to designate a B register as in B.*name*. BREG assigns numerical values to names in sequence beginning with the first available register after those used by the calling sequence.

Example:

Location	Result	Operand	Comment
1	10	20	35
ARPTR	BREG		Assigns ARPTR a value to be used as a B register designator

TREG - ASSIGN NAMES TO T REGISTERS (OBSOLETE)

The TREG macro assigns numerical values to symbols for use as T register names. It also checks that no more registers are used than are declared on the ENTER macro. The register names are assigned after any registers used by the calling sequence linkage. This macro must be used only after an ENTER macro. This macro has been replaced by the DEFT macro which is used before the ENTER macro.

Format:

Location	Result	Operand
name	TREG	

name Symbolic name designating a T register as in T.*name*. TREG assigns numerical values to names in sequence beginning with the first register available after those used by the calling sequence.

Example:

Location	Result	Operand	Comment
1	10	20	35
IPART	TREG		Assigns IPART a value to be used as a T register designator

INDEX

INDEX

READERS COMMENT FORM

Macros and Opdefs Reference Manual SR-0012

Your comments help us to improve the quality and usefulness of our publications. Please use the space provided below to share with us your comments. When possible, please give specific page and paragraph references.

NAME _____

JOB TITLE _____

FIRM _____

ADDRESS _____

CITY _____ STATE _____ ZIP _____

CRAY RESEARCH, INC.

CUT ALONG THIS LINE

FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY CARD

FIRST CLASS PERMIT NO 6184 ST PAUL, MN

POSTAGE WILL BE PAID BY ADDRESSEE

CRAY RESEARCH, INC.

Attention:
PUBLICATIONS

1440 Northland Drive
Mendota Heights, MN 55120
U.S.A.

FOLD

STAPLE

READERS COMMENT FORM

Macros and Opdefs Reference Manual SR-0012

Your comments help us to improve the quality and usefulness of our publications. Please use the space provided below to share with us your comments. When possible, please give specific page and paragraph references.

NAME _____

JOB TITLE _____

FIRM _____

ADDRESS _____

CITY _____ STATE _____ ZIP _____

CRAY RESEARCH, INC.

CUT ALONG THIS LINE

FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY CARD
FIRST CLASS PERMIT NO 6184 ST PAUL, MN

POSTAGE WILL BE PAID BY ADDRESSEE

CRAY RESEARCH, INC.

Attention:
PUBLICATIONS

1440 Northland Drive
Mendota Heights, MN 55120
U.S.A.

FOLD

STAPLE

To Examine the CAL listing or
CFT listing go to SPF 1 (browse)

For data set name type:

JOI1.MSS1.V113LIST

The member name for CFT is:

CFTLS

for CAL is:

CALLS

Cray Research, Inc.
Publications Department
1440 Northland Drive
Mendota Heights, MN 55120
612-452-6650
TLX 298444