CRAY RESEARCH, INC.

CRAY X-MP AND CRAY-1® COMPUTER SYSTEMS

MACROS AND OPDEFS REFERENCE MANUAL

SR-0012

RECORD OF REVISION

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.
A	November, 1984 - This rewrite supports the 1.14 COS release. All previous versions are obsolete. The manual has been reorganized and is now divided into two parts: user aids and system aids. The information on table and semaphore manipulation previously grouped in one section has been divided into four separate sections: normal table management, complex table management, translate table construction macros, and semaphore manipulation. Macro expansions are now grouped in Appendix B. The following new macros have been added for this printing: interjob communication (IJCOM), event recall (ERECALL), user channel access (DRIVER), unconditionally clearing a semaphore (WAIT$CLR), convert an integer string to an octal string ($OCTMIC), and positioning for beginning (BOV) and ending (EOV) of volumes.

PUBLICATION CHANGE NOTICE

November, 1984

TITLE: Macros and Opdefs Reference Manual

PUBLICATION NO. SR-0012 **REV.** A

This rewrite supports the 1.14 COS release. All previous versions are obsolete.

PREFACE

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

The contents of this manual presuppose that you are familiar with the CRAY-OS Version 1 Reference Manual, publication SR-0011. Likewise, the manual's contents assume that you have experience coding CAL as described in the CAL Assembler Version 1 Reference Manual, CRI publication SR-0000.

CRI publications SR-0014, the Library Reference Manual and SM-0045, the COS Tables Description Reference Manual contain reference information needed by the macro and opdefs instructions user.

The contents of this manual are organized to reflect the two main categories of macros and opdefs available to COS users:

Part 1 USER AIDS MACROS AND OPDEFS

This part includes a general description of macros and their coding requirements. It also describes macros and opdefs intended for all COS users.

Part 2 SYSTEM AIDS MACROS AND OPDEFS

Macros and opdefs which are primarily meant for the use of internal system users are described in this part.

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

CONTENTS

PART TWO - SYSTEM AID MACROS AND OPDEFS

PART 1

USER AID MACROS AND OPDEFS

INTRODUCTION 1

A set of macro and opdef instructions is included with the Cray Operating System (COS). These macros and opdefs provide you with a convenient way to handle tables and communicate with COS without need to be concerned with implementation details.

The macro and opdef instructions described in this manual are available only when you program in Cray Assembly Language (CAL). The assembler processes these macros by means of macro definitions in the system text, $SYSTXT. The exception is macros described under part 2, section 5, COS-dependent macros, which are defined in COSTXT.

CATEGORIES OF MACROS AND OPDEFS

The two main categories of macros and opdefs share a basic instruction format and are distributed as illustrated in figure 1-1.

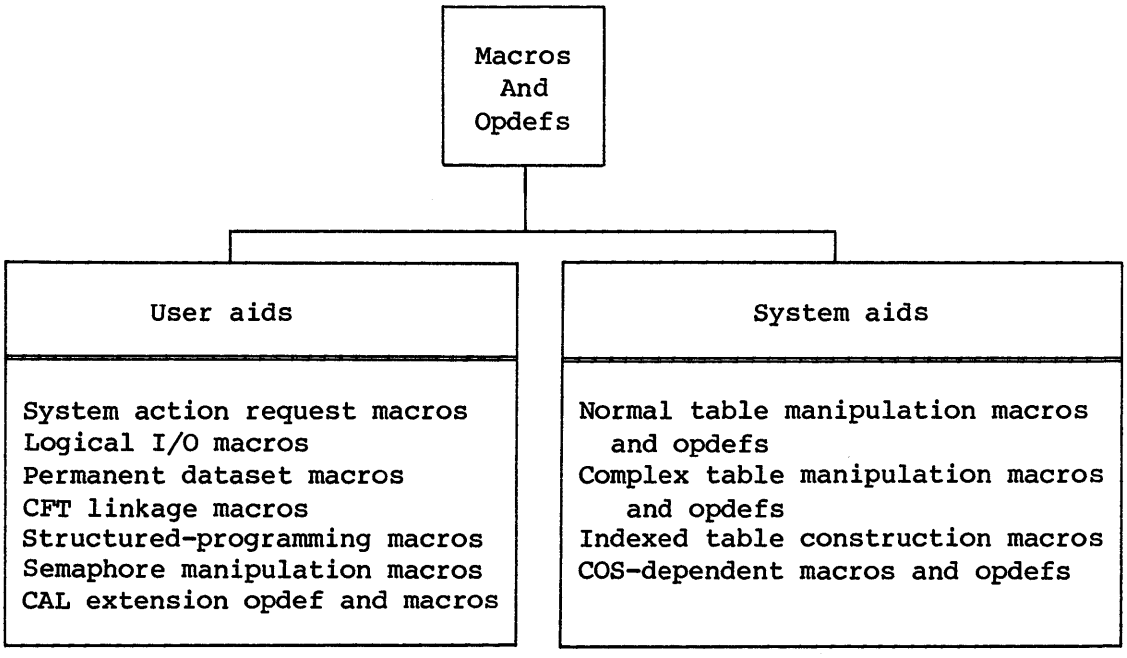


Figure 1-1. Categories of macros and opdefs

The macros in part 1 of this manual, the user aids group, are meant to meet the needs of external users. Examples of specific purposes of each type of macros and opdefs described in this manual include:

- System action request macros: job control macros, dataset management macros, time and date request macros, debugging aids, installation-defined macros
- Logical I/O macros: synchronous, asynchronous I/O, unblocked I/O, dataset positioning, user tape volume processing, Fortran-like I/O, SKOL-like I/O
- Permanent dataset macros: permanent dataset definition, permanent dataset management
- CFT linkage macros: entry block design, subroutine linkage, argument list information, local variable storage
- Structured-programming macros: conditions, macro descriptions
- Semaphore management: define semaphore name, get semaphore bit status, set semaphore without waiting, clear semaphore without waiting, test semaphore and wait to set, clear semaphore bit after waiting
- CAL extension opdef and macros: provide a precoded divide routine, pass elements of a vector register to a scalar routine, generate timing-related symbols and constants, convert an integer to a decimal micro string, convert an integer to an octal micro string, generate floating-point reciprocals
- Subsystem Support macros: interjob communication (IJCOM), user channel access (DRIVER), event recall (ERECALL)

The macros in part 2 of this manual, the systems aids group, are primarily intended for internal system users. Examples of specific purposes of each type of macros and opdefs described in this manual include:

- Normal table manipulation macros and opdefs: table definition macros, run-time field management opdefs, miscellaneous run-time field opdefs
- Complex table manipulation macros and opdefs: table definition macros, run-time table management opdefs
- Indexed table construction: jump vectors, indexed records
- COS-dependent opdefs and macros: system task opdefs, message processor macro, COS internal subroutine linkage macro

FORMAT

The format for a macro instruction is:

Location	Result	Operand
loc	*name*	$a_1,a_2,\ldots,a_j,f_1=b_1,f_2=b_2,\ldots,f_k=b_k,\text{SV=}\left\{\begin{matrix}\text{YES}\\\text{NO}\end{matrix}\right\}$

loc Location field argument. Certain macros require an entry in this field. For other macros, *loc* is an optional symbolic program address and is labeled as *oplabel*. Macro labels that generate a table are assigned a word address; macro labels 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. Positional parameters must preceed keyword parameters.

$f_j=b_j$ Keyword and actual argument; these entries can be in any order. A space or comma following the equal sign indicates a null string. A space also terminates the parameter list scan unless you enclose the space in parentheses.

$\text{SV=}\left\{\begin{matrix}\text{YES}\\\text{NO}\end{matrix}\right\}$ 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 before column 73 of the first line. You can include blanks in a parameter by enclosing the parameter in parentheses.

NOTE

Be careful when specifying A0 and S0 as macro parameters. A0 and S0 become special syntax values in certain CAL instructions, so be sure you know how the macro you are using handles A0 and S0 references.

SYSTEM ACTION REQUEST MACROS 2

The system action request macros are a subset of the system function requests. You can request system actions for job control, dataset management, timestamps, and debugging aids. Job control system actions, for example, can be useful to you in specifying abnormal termination of a job step, or to set a job step reprieve. Likewise, you can call upon the system to execute dataset management actions such as closing, releasing or disposing datasets.

Each macro generates an instruction sequence that is a call to the Cray Operating System (COS). The octal function code 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 by convention normally have a zero value when the call is completed if there were no errors. If there were errors, the job may either return an error code, or it may abort without returning an error code. 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
- Dataset management
- Time and date
- Debugging aids
- Installation defined macro

Unless specifically noted, all macros that generate executable code allow labels. The label is assigned the parcel address of the first executable instruction.

JOB CONTROL MACROS

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

ABORT - ABORT PROGRAM

The ABORT request provides for abnormal termination of the current job step without terminating the entire job. Processing resumes with the job control statement that follows the first EXIT statement after the aborted job step. If no EXIT statement exists, the job is terminated.

Format:

Location	Result	Operand
oplabel	ABORT	

oplabel Optional label

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, B, T, V and VL register values are taken from the user-supplied Exchange Package.

Format:

Location	Result	Operand
oplabel	CONTRPV	*address*

oplabel Optional label

address A symbol, or an A, S, or T register (except S0, S1, S2), that points to an Exchange Package in the your area that is used to continue processing. This is a required parameter.

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 for more information about the ECHO control statement. The location field is ignored at assembly time.

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 has elapsed.

Format:

Location	Result	Operand
oplabel	DELAY	*address*

oplabel Optional label

address A symbol or an A, S, or T register (not A0 or S0) 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
oplabel	DUMPJOB	DN=*dn*

oplabel Optional label

DN=*dn* A symbol or an A, S, or T register (not A0 or S0) containing 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
oplabel	ENDRPV	

oplabel Optional label

GETMODE - GET MODE SETTING

The GETMODE macro returns 1 to register S1 if the contents of parameter JCEFI=1; all other values return 0. GETMODE is obsolete and does not return reliable values for multitasking jobs.

Format:

Location	Result	Operand
	GETMODE	

GETSWS - GET SWITCH SETTING

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

Format:

Location	Result	Operand
oplabel	GETSWS	*n*

oplabel Optional label

n Number of the switch (1 through 6) to be tested. This parameter is required.

IOAREA - CONTROL USER ACCESS TO I/O AREA

The IOAREA request instructs the system to allow or deny access to the user's system managed I/O buffer and Dataset Parameter Table (DSP) areas.

See the subsection entitled dataset management macros, later in this section, for a detailed description of the DSP macro.

The IOAREA request can also restore the status of the I/O buffer and DSP areas to their initial status. Initially, the user I/O area is unlocked.

NOTE

The IOAREA macro does not protect I/O buffers and DSPs that have been allocated within the user's heap space. This applies only to those who use the stack-based version of the COS libraries.

Format:

Location	Result	Operand
oplabel	IOAREA	*key,save*

oplabel Optional label

key One of the following is required:

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. (Exchanges to the operating system are not required for all logical I/O calls.)

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 to used. The current status of *key* is stored in 1 word.

JTIME - REQUEST ACCUMULATED CPU TIME FOR JOB

JTIME requests that the accumulated CPU time for the job be returned to the location specified in the macro call. The time in seconds is expressed in floating-point form.

Format:

Location	Result	Operand
oplabel	JTIME	*address*

oplabel Optional label

address A symbol or an A, S, or T register (not A0 or S0) 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.

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.

Format:

Location	Result	Operand
oplabel	MEMORY	*code,value*

oplabel Optional label

code *code* is required and 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 you specify FL without *value*, the new field length is set to the maximum allowed the job.

You can change the job's field length with 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, and *value* is ignored. When you specify USER code, the job is placed in user mode until a subsequent request returns the job to automatic mode.

AUTO The job is put in automatic field length reduction mode, and *value* is ignored. When you specify AUTO code, 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*.

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 (not A0 or S0) that must contain a value when *code* is UC and may contain a value when code is FL. The system returns a value when *code* is CURFL, MAXFL, or TOTAL to the A, S, or T register specified in the macro statement.

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 5 words.

MESSAGE - ENTER MESSAGE IN LOGFILE

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

Format:

Location	Result	Operand
oplabel	MESSAGE	*address,dest,msgclass,override*

oplabel Optional label

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

dest Destination for message; this parameter is optional and can be any of the following:

U User logfile only; specified register=1.
S System logfile only; specified register=2.
US User and system logfiles; specified register=3. This is the default if *oplabel* is unspecified.

msgclass Assign the message to class *msgclass*. *msgclass* can be a symbol or an A, S, or T register (except A0, S0, S2, S3, or S4) containing the message class. See the ECHO control statment in the CRAY-OS Version 1 Reference Manual, publication SR-0011, for the available message classes. This parameter is optional.

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. This parameter is optional.

MODE - SET OPERATING MODE

The MODE macro allows you to set or clear mode flags in the job's Exchange Package.

Format:

Location	Result	Operand
oplabel	MODE	FI=*option*,BT=*option*,EMA=*option*, AVL=option,ORI=option

Parameters are in keyword form. You must specify at least one of the following parameters:

oplabel Optional label

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 is an installation option that enables or disables bidirectional memory transfers. If you specify BT=DISABLE, block reads and writes are not performed concurrently. The BT parameter is operational on CRAY X-MP Computer Systems only.

EMA=*option*†

Extended memory addressing mode. *option* can be:

ENABLE	Enables extended memory addressing
DISABLE	Disables extended memory addressing

The default is an installation option, released as EMA=ENABLE. This feature is available only on later model CRAY X-MP Computer Systems. See the CRAY X-MP Series Model 48 Mainframe Reference Manual, publication HR-0097, for a complete description of extended memory addressing.

AVL=*option*†

Additional vector logical functional unit. *option* an be:

ENABLE	Enables additional vector logical functional unit
DISABLE	Disables additional vector logical functional unit

The default is an installation option, released as AVL=DISABLE. This feature is available only on later model CRAY X-MP Computer Systems. When enabled, the primary path for vector logical operations is through the vector multiply functional unit, and a secondary path is available through the vector logical functional unit. When disabled, only one path is available for vector logical operations.

ORI=*option*

Operand range interrupt mode. *option* can be:

ENABLE	Enables operand range interrupts
DISABLE	Disables operand range interrupts

The default is an installation option, released as ORI=ENABLE. This feature is available only on CRAY X-MP Computer Systems. When enabled, operand range errors cause a program interrupt; when disabled, operand range errors are ignored by the hardware. This feature is also available through the ERI (002300) and DRI (002400) instructions, but executing these instructions does not notify the operating system of the mode change.

---

† Referred to as enhanced addressing mode in the the CRAY X-MP Series Model 48 Mainframe Reference Manual, publication HR-0097.

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 nonrerunnable without affecting the current rerunnability of the job. The default value is determined by the installation.

Format:

Location	Result	Operand
oplabel	NORERUN	*option*

oplabel Optional label

option One of the following is required:

parameter A symbol identifying a location or an A, S, or T register (not A0 or S0) 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 macro does not affect the existing rerunnability of the job. The functions that make a job nonrerunnable 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
oplabel	RECALL	*address*

oplabel Optional label

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 nonrerunnable regardless of functions previously performed. The future monitoring of nonrerunnable functions is not affected.

Format:

Location	Result	Operand
oplabel	RERUN	*option*

oplabel Optional label

option One of the following is required:

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 nonrerunnable

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
- Release any nonpermanent dataset
- Perform queued I/O on a dataset

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

Format:

Location	Result	Operand
oplabel	ROLL	

oplabel Optional label

SETRPV - SET JOB STEP REPRIEVE

The SETRPV request enables your current job step to maintain control when a job step abort error condition occurs or upon normal termination of the job step. Once you enable reprieve processing, it remains in effect until the job step terminates, a selected error condition occurs, or you disable reprieve processing. See ENDRPV at the beginning of this section for information on terminating reprieve processing. For other CAL and CFT techniques, see the Library Reference Manual, CRI publication number SR-0014.

If a selected error condition occurs, the job step is reprieved from the normal or abnormal job step termination. The reprieve processing code transfers control to a user specified address. This address must be executable code to ensure proper execution.

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. You can reprieve nonfatal error conditions any number of times per job step. 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
oplabel	SETRPV	*entry,xpsave,mask*

oplabel Optional label

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

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 parameter is required.

mask A user-specified octal value indicating the classes of conditions to enable reprieve processing. Any number of classes is 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

Class (Octal mask value)	Reprievable condition
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, you must issue another SETRPV request. You cannot issue a SETRPV request for second occurrences of errors defined as fatal.

SWITCH - SET OR CLEAR SENSE SWITCH

The SWITCH macro allows you to turn ON (set) or turn OFF (clear) pseudo sense switches. GETSWS returns the setting of the switch number specified in the S1 register (see SETSWS at the end of this section).

Format:

Location	Result	Operand
oplabel	SWITCH	*n,x*

oplabel Optional label

n Number of switch (1 through 6) to be set or cleared; required parameter.

† Fatal error

x Switch position; *x* is a required parameter and can be:

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

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 2 words; the first contains the COS revision level in ASCII and the second contains the COS assembly date in ASCII. For example:

COS *x.xx*
mm/dd/yy

Format:

Location	Result	Operand
oplabel	SYSID	*address*

oplabel Optional label

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

DATASET MANAGEMENT

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

CLOSE - CLOSE DATASET

CLOSE releases the buffer, the Logical File Table (LFT), and the Dataset Parameter Table (DSP) for a COS-managed dataset. Disk space is not released (as opposed to RELEASE which gives up the DNT as well) and the dataset remains accessible 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.
- The dataset has COS blocked dataset structure.
- The dataset is not memory resident.

Format:

Location	Result	Operand
oplabel	CLOSE	*dn*

oplabel Optional label

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 the description of the OPEN macro later in this subsection.

DISPOSE - DISPOSE DATASET

The DISPOSE macro places a dataset in the appropriate queue as defined by the PDD macro (see the CRAY-OS Version 1 Reference Manual, publication SR-0011 for more information on the PDD macro).

Format:

Location	Result	Operand
oplabel	DISPOSE	*pddtag*

oplabel Optional label

pddtag Address of Permanent Dataset Definition (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.

You should use the DSP macro only when you need 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†) or in the user heap space, if you are using stack versions of library routines, by execution of an OPEN macro.

When using the DSP macro, you must also set up a 2-word Open Dataset Name Table (ODN). You must define ODN before using an OPEN macro specifying this dataset. For more information on ODN, see the CRAY-OS Version 1 Reference Manual, publication number SR-0011.

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. See the CRAY-OS Version 1 Reference Manual, publication SR-0011 for a detailed description of the DSP.

Format:

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

loc Symbolic address of DSP. If *loc* is not specified, a symbol is defined. The default symbolic name is generated by appending @ to the dataset name.

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

† For more information on JCHLM see the section on Job memory management in the CRAY-OS Version 1 Reference Manual, publication SR-0011.

Example:

Location	Result	Operand	Comment
1	10	20	35
x	DSP	XFIL,BUF,1	
ODN	CON	'XFIL'L	ASCII name
	CON	X	Address of DSP
	.		
	.		
	.		
	OPEN	ODN,I	

Example (default):

Location	Result	Operand	Comment
1	10	20	35
	DSP	XFIL,BUF,1	
ODN	CON	'XFIL'L	ASCII name
	CON	XFIL@	Address of DSP
	.		
	.		
	.		
	OPEN	ODN,I	

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
oplabel	DSPLOCK	*action*,NAME=*name*,DSP=*addr*

oplabel Optional label

action This parameter is required and specifies the action to be taken. *action* can be one of the following:

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 address of the DSP.

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 or in the heap, including 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
oplabel	OPEN	*dn,pd,ldt,u*

oplabel Optional label. If *oplabel* is not specified, the address of the dataset name is generated.

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 you 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. See part 1, section 4 of this manual for a complete description of the LDT macro.

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

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. For more information on JCDSP, see the CRAY-OS Version 1 Reference Manual, publication SR-0011. 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.

On the other hand, if the DSP pointer in the ODN is greater than 0, OPEN assumes the DSP field contains the address of your 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 you indicate 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 this 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
	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 RELEASE macro causes the dataset whose Dataset Parameter Table (DSP) address is at the location specified in the macro call to be 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 accessible to the job.

Format:

Location	Result	Operand
oplabel	RELEASE	*address*,HOLD

oplabel Optional label

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 (not A0 or S0) containing the ODN or DSP address. See description of OPEN and DSP macros in this section. This parameter is required.

HOLD Hold generic device. If you specify HOLD, the generic system resource (the peripheral) associated with this dataset is not returned to the system pool. This parmeter is optional.

SUBMIT - SUBMIT JOB DATASET

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

Format:

Location	Result	Operand
oplabel	SUBMIT	*pddtag*

oplabel Optional label

pddtag Address of the Permanent Dataset Definition Table (PDD). This parameter is required.

TIME AND DATE REQUEST

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

DATE - GET CURRENT DATE

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

0	8	16	24	32	40	48	56
m	*m*	/	*d*	*d*	/	*y*	*y*

Your site can change the order of the information, returned by the DATE macro, with an installation parameter. If the European format is used, information is returned as day, month, and year.

Format:

Location	Result	Operand
oplabel	DATE	*address*

oplabel Optional label

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

DTTS - CONVERT DATE AND TIME TO TIMESTAMP

DTTS converts the ASCII date and time into the corresponding system timestamp.

A timestamp is a 1-word encoding of the date and time expressed in units of nanoseconds/1.024. When used to express a date and time, a timestamp is the number of timestamp units past the system base date (1 January 1973) and the date/time to be encoded. See subroutines MTTS, TSMT, TSDT).

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

	0	8	16	24	32	40	48	56
S1	*m*	*m*	/	*d*	*d*	/	*y*	*y*
S2	*h*	*h*	:	*m*	*m*	:	*s*	*s*

Format:

Location	Result	Operand
	DTTS	

The location field of the DTTS macro is completely ignored when the macro is expanded.

On exit from the macro, S1 is the timestamp corresponding to the requested date/time. The resulting timestamp can vary from one mainframe hardware type to another. See the CRAY-OS Version 1 Reference Manual, publication SR-0011 for more information about the timestamp.

JDATE - RETURN JULIAN DATE

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

0	8	16	24	32	40	48	56
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
oplabel	JDATE	*address*

oplabel Optional label

address A symbol or an A, S, or T register (not A0 or S0) containing the destination address of the current Julian date

MTTS - CONVERT MACHINE TIME TO TIMESTAMP

MTTS converts a machine time (real-time, RT, register value) 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	

The location field of the DTTS macro is completely ignored when the macro is expanded.

S1 contains the timestamp on exit from the macro. The resulting timestamp can vary from one mainframe hardware type to another.

TIME - GET CURRENT TIME

TIME returns the current time in ASCII at the location you specify in the macro call. The format of the time is as follows:

0	8	16	24	32	40	48	56
h	*h*	:	*m*	*m*	:	*s*	*s*

Format:

Location	Result	Operand
oplabel	TIME	*address*

oplabel Optional label

address A symbol or an A, S, or T register containing the destination address of the current time. This parameter is required.

TSDT - CONVERT TIMESTAMP TO DATE AND TIME

TSDT converts a timestamp (1-word encoding of a date and time) to the corresponding date and time expressed in ASCII.

Register S1 on entry holds the timestamp. On exit, registers are set as follows (S1 holds the ASCII date, S2 holds the ASCII time, and S3 holds the ASCII fractional sectionds):

	0	8	16	24	32	40	48	56
S1	*m*	*m*	/	*d*	*d*	/	*y*	*y*
S2	*h*	*h*	:	*m*	*m*	:	*s*	*s*
S3	.	*s*	*s*	*s*	*s*	^	^	^

The information in register S3 is left-justified and blank filled (^).

Format:

Location	Result	Operand
	TSDT	

The location field of the TSDT macro is completely ignored when the macro is expanded. The source timestamp can vary from one mainframe hardware type to another.

TSMT - CONVERT TIMESTAMP TO MACHINE TIME

TSMT converts a timestamp (1-word encoding of date and time) to the corresponding machine time (real-time, RT, clock value). Register S1 on entry contains the timestamp.

Format:

Location	Result	Operand
	TSMT	

The location field of the TSMT macro is completely ignored when the macro is expanded.

S1 contains the machine time on exit from the macro. The resulting timestamp can vary from one mainframe hardware type to another.

DEBUGGING AIDS

The system action request macros in this category permit you to selectively read or write information during a program run to aid in the debugging process. Included are the DUMP, LOADREGS, SAVEREGS, and SNAP macros. You can use the label DEBUG for conditional execution of DUMP and SNAP.

DUMP - DUMP SELECTED AREAS OF MEMORY

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

The macro generates a minimal amount 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 (see the CRAY-OS Version 1 Reference Manual, publication SR-0011) or equate control statement, code generation within the macro is suppressed entirely.

Format:

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

oplabel Optional label

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 can be 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 base is decimal unless a BASE O (octal) or BASE M (mixed) is in effect. The default for BASE O and BASE M is octal.

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.

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
oplabel	LOADREGS	[*region*],INLINE={YES / NO}

oplabel Optional label

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*. However, no system checks are made, and it remains the user's responsibility.

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

Inline Code flag. If you omit INLINE, 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.

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.)

The SAVEREGS macro sets a hardware semaphore bit. You must specify a LOADREGS for every SAVEREGS to clear the semaphore bit set by SAVEREGS. Since other macros, SNAP for example, call SAVEREGS, never use an unpaired SAVEREGS with an unspecified region. SAVEREGS can be nested only if different regions are specified.

Format:

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

oplabel Optional label

region Label of the first word of a region where registers are to be saved. The default is QZH44HZQ. When you define the *region*, it must be an area containing O'1230 words; if you do not define *region*, 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 you omit INLINE, 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 which you have selected.

The macro generates a minimal amount 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, code generation within the macro is suppressed entirely unless a previously assembled SET or equate statement has set 1 to the symbol DEBUG.

Format:

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

oplabel Optional label

list A list of registers and register groups separated by commas. You do not need to enclose the list 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 base of n is decimal unless a BASE O (octal) or BASE M (mixed) is in effect. If BASE O or BASE M is specified the default is octal.

Return conditions:

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

INSFUN - INSTALLATION-DEFINED SUBFUNCTIONS

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

NOTE

Cray Research, Inc., does not support the installation-defined subfunctions which INSFUN calls.

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. This parameter is required.

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. This parameter is optional.

The location field of the INSFUN macro is completely ignored when the macro is expanded.

You should see your local site programmers for definitions of available functions.

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. The OPEN macro must have opened the datasets referenced by these logical I/O macros.

The categories of logical I/O macros are: synchronous I/O, asynchronous (buffered) I/O, unblocked I/O, dataset positioning, user tape volume processing, FORTRAN-like I/O, and SKOL-like I/O.

SYNCHRONOUS I/O

With the synchronous read/write logical I/O macros you can read and write words or characters, as well as 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. You cannot assume that A or S registers, other than those specifically mentioned, have any meaningful contents. Futhermore, you cannot assume that the registers will change values which they had 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.

Synchronous read/write logical I/O macros include READ/READP, READC/READCP, WRITE/WRITEP/ WRITEC/WRITECP, WRITED, and WRITEF.

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, nor are any compressed blanks expanded with these macros (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 to be processed at a time. 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. If requested, words are transmitted to the user's data area. Each call is terminated by reaching an EOR or by satisfying the word count, whichever occurs first. If you specify READP with a word count of 0, an EOR is forced after a series of READP calls.

No blank decompression is performed.

When EOR is reached as a result of reading in word mode, the unused bit count from the EOR 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 reference 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
oplabel	READ	*dn,uda,ct*

Location	Result	Operand
oplabel	READP	*dn,uda,ct*

oplabel Optional label

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 (registers):

(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 EOR is encountered. For partial read (READP) if EOR and end of count coincide, EOR takes precedence.

(S1) Error status:

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

(S6) Contents of the 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 to be processed at a time. 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. You 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.

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

Formats:

Location	Result	Operand
oplabel	READC	*dn,uda,ct*

Location	Result	Operand
oplabel	READCP	*dn,uda,ct*

oplabel Optional label

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 EOR is written following each WRITE. The EOR indicates how many bits in the last words are unused, if any. No blank compression is performed. When a WRITE macro has written an EOR, the user program cannot write any more data before issuing the STARTSP macro.

The WRITEP macro generates a return jump to the $WWDP subroutine. No EOR is written, nor is blank compression performed. If you specify WRITEP with a word count of 0, the request is treated as a no op. 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 EOR, use the WRITE macro with a word count of 0.

Formats:

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

Location	Result	Operand
oplabel	WRITEP	*dn,uda,ct*

oplabel Optional label

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 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 to be processed at a time. An EOR is written following each WRITEC.

The WRITECP macro generates a return jump to the $WCHP subroutine and characters are written from the user's data area without an EOR.

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 EOR, the WRITEC macro with a character count of 0 is used.

NOTE

Use WRITEC with a character count of 0 to complete a record written with WRITECP. If you write the EOR by some other means (such as with WRITE or by closing the dataset), the last seven characters of data can be lost.

Formats:

Location	Result	Operand
oplabel	WRITEC	*dn,uda,ct*

Location	Result	Operand
oplabel	WRITECP	*dn,uda,ct*

oplabel Optional label

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* 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 EOR (if not previously written), an EOF (if not previously written), and an EOD 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
oplabel	WRITED	*dn*

oplabel Optional label

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

WRITEF - WRITE END-OF-FILE

The WRITEF macro generates a return jump to the $WEOF subroutine, causing an EOR (if not previously written) and an EOF 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
oplabel	WRITEF	*dn*

oplabel Optional label

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 I/O

The asynchronous read/write logical I/O macros allow you 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 you immediately. It is your 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 (see WRITE/WRITEP in this section for more information on *uda*) specify the address of an area in the user's program. Also, the *ct* parameter (see WRITE/WRITEP in this section for more information on *ct*) 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.

Asynchronous read/write logical I/O macros include BUFCHECK, BUFEOD, BUFEOF, BUFIN/BUFINP, and BUFOUT/BUFOUTP.

BUFCHECK - CHECK BUFFERED I/O COMPLETION

With the BUFCHECK macro, you request the system to wait for the buffered I/O on a dataset to complete the transfer 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
oplabel	BUFCHECK	*dn,err*

oplabel Optional label

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 your 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 causes an EOR (if not previously written), an EOF (if not previously written), and an EOD to be written. Control optionally returns immediately to you and it is your responsibility to monitor the DPBIO field.

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

Format:

Location	Result	Operand
oplabel	BUFEOD	*dn,rcl*

oplabel Optional label

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 optionally returns immediately to your program, giving you the responsibility of monitoring the DPBIO field. An EOR is written if the dataset is at mid record.

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

Format:

Location	Result	Operand
oplabel	BUFEOF	*dn,rcl*

oplabel Optional label

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$BIO (see the CRAY-OS Version 1 Reference Manual, publication SR-0011 for more information on F$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 optionally returns to your program immediately, giving you 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, transfer continues until the specified word count or EOD is reached.

Formats:

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

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

oplabel Optional label

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. 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 EOR 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 EOR contains this unused bit count.

The BUFOUTP macro transfers the specified number of words but does not write an EOR. 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 optionally returns to your program immediately, giving you 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
oplabel	BUFOUT	*dn,uda,ct,ubc,rcl*

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

oplabel Optional label

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 I/O

The unblocked dataset read and write macros allow you 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.

****

CAUTION

Special caution should be used with registers B0, B70 through B77, and T70 through T77 as CFT and other routines use them also.

****

The READU and WRITEU macros comprise the unblocked I/O macros.

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
oplabel	READU	*dn,uda,ct*

oplabel Optional label

dn Dataset name (symbolic address of the Open Dataset Name Table (ODN) 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 (registers):

(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
oplabel	WRITEU	*dn,uda,ct*

oplabel Optional label

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 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

DATASET POSITIONING

You 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.

The positioning macros (ASETPOS, BKSP, BKSPF, CLOSEV, GETPOS, POSITION, REWIND, SETPOS, SYNCH, TAPEPOS, and TAPESTAT) described in this subsection refer to the Open Dataset Name Table (ODN). For more information about the ODN, see the CRAY-OS Version 1 Reference Manual, publication SR-0011.

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).

For a blocked dataset, the macro initiates a read to the I/O buffer before positioning occurs, unless the requested position is already in the buffer. 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
oplabel	ASETPOS	*dn,pos*

oplabel Optional label

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; register S_n=-1

BOD Position the dataset at BOD; register S_n=0

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.

If the specified register contains -1, position the dataset at EOD. For example: S_1=-1.

If the specified register contains 0, position the dataset at BOD. For example: S_1=0.

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

The 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
oplabel	BKSP	*dn*

oplabel Optional label

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

Return conditions:

(A1) DSP address

(S6) The record control word (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
oplabel	BKSPF	*dn*

oplabel Optional label

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

Return conditions:
Same as for BKSP

CLOSEV - SWITCH TO THE NEXT VOLUME

A user program uses the CLOSEV macro to switch to the next tape volume at any time. The CLOSEV macro writes an end of volume (EOV) trailer label to a mounted output tape before switching tapes. CLOSEV applies only to magnetic tape datasets.

If the tape is an input tape, you have the option of writing an EOV trailer label. An output tape job is aborted if the output buffer is not empty.

In special EOV processing the user program must issue the CLOSEV macro to switch to the next tape and perform special beginning-of-tape (BOV) processing. After the CLOSEV macro is executed, the next tape is at the beginning of the volume. The user program is permitted BOV processing at this time. After the BOV processing is completed, the user program must issue the ENDSP macro to inform the operating system that special processing is complete and to continue normal processing.

Format:

Location	Result	Operand
	CLOSEV	*odn*,L=*parmlist*,TRAILER=EOV

odn ODN address

L=*parmlist*

Parameter list address. The length of the list is defined by symbol LE@TEV; symbol LE@TEV is defined in the default system text, $SYSTXT. If the parameter list address is not coded, the storage is generated inline.

TRAILER=EOV

Indicates that the EOV trailer label will be written

Return conditions:

(S1)=0 No errors

(S1)>0 TRAILER=EOV is specified for an input tape. Mount the next volume.

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
oplabel	GETPOS	*dn*

oplabel Optional label

dn Dataset name (symbolic address of the Open Dataset Name Table (ODN) 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 0-3 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 the dataset is positioned immediately following an RCW. Bit 0-3 contains 10_8 for EOR, 16_8 for EOF, 17_8 for EOD, and 0 for mid-record.

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 DATASET

With the POSITION macro you can 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.

You need to consider the effect of POSITION during special EOV/BOV processing. If you issue the POSITION macro in the EOV/BOV special processing routine and request relative block positioning, then the positioning will start from wherever the tape is positioned, which is not necessarily the last block the user program has read or written.

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 CRAY-OS Version 1 Reference Manual, publication SR-0011.

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 specify A0, S0, S1, S2, or S3. The value of *nb* cannot be greater than 15,728,639. Possible specifications for *nb* re:

+*nb* Space *nb* blocks forward from the current position. The + sign is invalid if either V or VOL is coded for POSITION.

-*nb* Space *nb* blocks back from the current position. The - sign is invalid if either V or VOL is coded for POSITION.

nb Specifies the absolute block number on which the dataset is positioned. This command applies only to the volume the programming is processing. No volumes are skipped. If the block request does not exist on the volume, the tape is repositioned at the end of the volume, and an error is returned to the user.

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 specify A0, S0, S1, S2, or S3. *nv* cannot be greater than the volume specified by the VOL parameter.

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

+*nv* Position *nv* volumes ahead of the current volume

-*nv* Position *nv* volumes back from the current volume

nv Position to absolute volume number *nv*.

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. You can also specify *vi* 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 you do not code *pla* 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=1	Dataset is not a tape dataset
TPNR=2	Tape is not at EOR
TPNS=3	Positioning request not fully satisfied. S2 contains the number of blocks not forward- or back-spaced.
TPTM=4	Tape mark encountered

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
oplabel	REWIND	*dn*

oplabel Optional label

dn Dataset name. *dn* is the symbolic address of the Open Dataset Name Table (ODN) for this dataset.

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 macro is described in part 1, 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, unless the requested position is already in the buffer. 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
oplabel	SETPOS	*dn,pos*

oplabel Optional label

dn Dataset name. *dn* is the symbolic address of the Open Dataset Name Table (ODN) for this dataset.

dn can also be an A, B, or S register containing the Dataset Parameter Table (DSP) address or negative DSP offset. The DSP macro is described in part 1, 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 data transfer 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 CRAY-OS Version 1 Reference Manual, publication SR-0011.

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 - GET TAPE DATASET POSITION

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 LE@TPI 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 CRAY-OS Version 1 Reference Manual, publication SR-0011.

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 LE@ 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 the I/O Subsystem (IOS); for input, always zero.
TPIBC	9	0	32	Number of blocks in IOS buffer
TPRSV4	10	0	64	Reserved

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.

TAPESTAT - OBTAIN TAPE STATUS FROM DSP

You should use TAPESTAT to check for any tape status after the READ, WRITE, BUFIN, BUFOUT and SYNCH macros.

Format:

Location	Result	Operand
	TAPESTAT	*odn*

odn Open Dataset Name Table (ODN) address

Return conditions:

S0=0 if circular buffer is empty
S0>0 if circular buffer is not empty
S1=*tape status code*; where *tape status code* describes current tape status:

Code	Description
TS$EOV	EOV status
TS$TOR	Tape off reel
TS$TMS	Tape mark status
TS$BLT	Blank tape detected

The tape status bits represent the tape's actual status. For input tapes, the user program has not reached the condition that the tape status indicates if the circular buffer is not empty (S0 not zero).

USER TAPE VOLUME PROCESSING

The Cray Operating System (COS) handles multivolume (multireel) tape datasets automatically. However, the following macros allow you to perform special end of volume (EOV) and beginning-of-volume (BOV) processing.

- Use SETSP to notify COS that you wish to perform your own EOV and BOV processing. COS will then notify you whenever it reaches the end of a tape volume.

- Use STARTSP to notify COS that you are beginning your special processing.
- Use ENDSP to notify COS that you have finished your special processing.

The user tape volume processing macros include ENDSP, SETSP, and STARTSP.

ENDSP - SPECIAL EOV AND BOV PROCESSING IS COMPLETE

The ENDSP macro indicates to the operating system that special end-of-volume (EOV) and beginning-of-volume (BOV) processing is complete.

ENDSP does not switch volumes; when the user program wants to switch to the next tape, the CLOSEV macro must be issued. Furthermore, data in the input/output (IOP) buffer is not written to tape until the ENDSP macro is issued at the end of BOV processing on the next tape. When the BOV processing is done, the user program must issue the ENDSP macro to terminate special processing. After the ENDSP macro is issued, the user program can continue to process the tape dataset.

Format:

Location	Result	Operand
	ENDSP	*odn*,L=*pla*

odn Open Dataset Name Table (ODN) address

L=*pla* Parameter list address. Address of storage area of length LE@TEV. If you specify L, the macro generates the list at address *pla*. If you do not code it, the parameter list is generated at an address determined by the macro.

SETSP - REQUEST NOTIFICATION AT END-OF-TAPE VOLUME

The SETSP macro informs the operating system that you wish to perform extra processing when the end of a tape volume is reached. You must use the SYNCH macro to ensure all data is written to tape before issuing SETSP.

After the user program has executed the SETSP macro, the EOV condition is set when the tape is positioned after the last data block. For an input tape the EOV condition is set after the system has read the last data block on the volume. For an output dataset, however, the EOV condition is set when end-of-tape (EOT) status is detected.

Automatic volume switching is not done by COS following the successful execution of the SETSP macro with the ON option. If you want to control volume switching, use the CLOSEV macro.

Format:

Location	Result	Operand
	SETSP	*odn*, {ON / OFF}, L=*pla*

odn Address of the Open Dataset Name Table (ODN)

{ON / OFF} If you specify ON, the user program is notified at EOV. Otherwise, OFF specifies that the user program no longer needs to be notified at EOV.

L=*pla* Parameter list address. Address of storage area of length LE@TEV. If you specify L, the macro generates the list at address *pla*. If you do not code it, the parameter list is generated at an address determined by the macro.

STARTSP - BEGIN USER EOV AND BOV PROCESSING

The STARTSP macro starts special end-of volume (EOV) and beginning-of-volume (BOV) processing. No special-processing I/O to the tape takes place until this macro has been executed: the user program must inform the Cray Operating System (COS) that it intends to reposition or perform special I/O to the tape by executing the STARTSP macro.

After issuing the STARTSP macro the user program can issue the READ, WRITE and POSITION macros. When processing is done, the user program must issue the ENDSP macro to inform COS that special processing is done. STARTSP does not switch volumes; when the user program wants to switch to the next tape, you have to issue the CLOSEV macro. Moreover, after you issue the STARTSP macro and before you issue the ENDSP macro, the CLOSEV macro is the only macro that performs volume switching for the user program.

You must issue the SYNCH macro before issuing the STARTSP macro. The data in the buffer is not written to tape until the ENDSP macro is issued at the end of BOV processing on the next tape.

Format:

Location	Result	Operand
	STARTSP	*odn*,L=*pla*

odn Address of the Open Dataset Name Table (ODN)

L=*pla* Parameter list address. Address of storage area of length LE@TEV. If you specify L, the macro generates the list at address *pla*. If you do not code it, the parameter list is generated at an address determined by the macro.

FORTRAN-LIKE I/O

The FORTRAN-like I/O macros allow you to perform formatted and unformatted reads and writes using FORTRAN-like syntax in a CAL program. The FORTRAN-like I/O macros include: FREAD, FWRITE, UFREAD, UFWRITE.

FREAD - READ DATA

The FREAD macro permits a FORTRAN-like read statement that can make use of a FORTRAN-like 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:

```
fmt DATA     '(F10.0)'
```

The character string is left-justified, and the first character in the string is the left parenthesis.

((*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, $W.177_8$).

SV={YES / NO} Save flag. The default (SV=NO) does not invoke the SAVEREGS and LOADREGS macros. If SV=YES, all registers are saved and restored. Always specify SV=YES to protect the contents of the registers.

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=$\left\{\begin{matrix}\text{YES}\\\text{NO}\end{matrix}\right\}$,UNIT=*unit*

fmt Format; takes one of the following forms:

faddr Address of a format, possibly defined with the DATA pseudo instruction, as in:

```
fmt DATA     '(F10.0,''TEXT'')'
```

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

The default is (5O25).

(*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, $W.177_8$).

SV={YES / NO} Save flag. The default (SV=NO) does not invoke the SAVEREGS and LOADREGS macros. If SV=YES, all registers are saved and restored. Always specify SV=YES to protect the contents of the registers.

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. See the CRAY-OS Version 1 Reference Manual, publication SR-0011 for more detailed information about $OUT.

UFREAD - UNFORMATTED READ

The UFREAD macro performs a FORTRAN-like unformatted read.

Format:

Location	Result	Operand
	UFREAD	*unit*,(*list*),SV={YES / NO},END=*addr*,ERR=*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, $W.177_8$).

SV={YES / NO} Save flag. The default (SV=NO) does not invoke the SAVEREGS and LOADREGS macros. If SV=YES, all registers are saved and restored. Always specify SV=YES to protect the contents of the registers.

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

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={YES / NO}

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, $W.177_8$).

SV={YES / NO} Save flag. The default (SV=NO) does not invoke the SAVEREGS and LOADREGS macros. If SV=YES, all registers are saved and restored. Always specify SV=YES to protect the contents of the registers.

SKOL-LIKE I/O

The SKOL-like I/O macros allow you to perform formatted and unformatted reads and writes using SKOL-like syntax in a CAL program. The SKOL-like macros include: INPUT and OUTOUT.

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 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={YES / NO},IN={YES / NO},UNIT=*unit*, STRING=*string*,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 can be 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 the following:

: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 $W.177_8$).

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 @ 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 3 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 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).

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 subroutine created by the macro. In the latter case, a minimal amount of code is 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*::O22

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

The variable can refer to a single word, an array, a single register, or 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 $W.177_8$).

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 @ 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*.

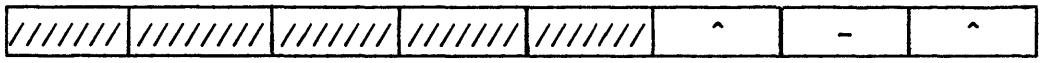
SV={YES / NO} Save flag. If SV=NO, the SAVEREGS and LOADREGS macros are not invoked, and registers cannot be used for output; IN=YES must also be specified if SV=NO. The default is SV=YES, which saves and restores all registers.

IN=$\begin{Bmatrix} YES \\ NO \end{Bmatrix}$ 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 a minimal amount of code is 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 ^ is a blank space. If the first eight characters do not match the above, OUTPUT inserts the following 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).

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 used later in such a call. The function code value is stored in register S0; S1 and S2 provide optional arguments. The function code is processed when the program exit instruction is executed.

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 Permanemt Definition Dataset Table (PDD). For more information on the PDD, see the CRAY-OS Version 1 Reference Manual, publication SR-0011. Thus, you must use the PDD macro with the permanent dataset management macros. For a description of the DISPOSE and SUBMIT macros, see Dataset Management Macros, part 1, 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. You must use the LDT macro with the PDD and ACCESS macros in a program accessing a labeled tape dataset if you want label processing.

LDT - CREATE LABEL DEFINITION TABLE

The LDT macro creates a table called the Label Definition Table (LDT). This macro does not produce executable code; moreover, you should not place it in in-line code. You must use the LDT macro with 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*,RS=*rs*,MBS=*mbs*

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=(*vsn*$_i$)
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 Control Data Corporation (CDC) tape datasets, *rf* values are:

IIW	SCOPE 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={YES / NO},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. Parameters apply only to mass storage datasets, unless otherwise noted.

DN=*dn* Dataset name. DN is a required parameter. This applies to mass storage or tape datasets.

PDN=*pdn* Permanent dataset name. The default value is *dn*. This applies to mass storage or tape datasets.

SDN=*sdn* Staged dataset name; 1- to 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- to 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. This applies to mass storage or tape datasets.

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- to 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- to 8-alphanumeric characters. SF is defined by the needs of the front-end system. Consult site operations 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; from 1- to 8-alphanumeric characters assigned by the dataset creator. The default is no read control word.

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

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

DT=*dt* Tape dataset generic device name. This parameter is required for tape datasets; it is ignored if you use it for mass storage datasets. See site operations for legal values and meanings.

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

BLP Bypass label processing†
FSL Field IBM standard labeled tapes
FNL Field unlabeled tapes; default.
FAL Field ANSI standard labeled tapes
SL IBM standard labeled tapes
NL Unlabeled tapes; default.
Al ANSI standard labeled tapes

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 you use it for mass storage datasets. *ltd* must match *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, and applies only to mass storage.

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

UQ={ON / OFF} Unique access. If you specify UQ, 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). UQ applies only to mass storage.

WAIT={ON / OFF}

Job wait/nowait. If you specify WAIT=ON, the job waits for the dataset to be transferred to the front-end system. If the transfer is canceled, the job is aborted. However, if you specify WAIT=OFF, 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 you omit the parameter, an installation default parameter is used. WAIT applies only to mass storage.

† Deferred implementation

DEFER=$\left\{\begin{matrix}\text{ON}\\\text{OFF}\end{matrix}\right\}$

Deferred disposition. When you specify DEFER, 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, and applies only to mass storage.

NRLS=$\left\{\begin{matrix}\text{ON}\\\text{OFF}\end{matrix}\right\}$

No release. When you specify NRLS=ON, the dataset remains local to the job after a DISPOSE request has been processed. The default is NRLS=OFF, and applies only to mass storage.

NOTE

The dataset is available only for reading when NRLS=ON is specified, until all dataset staging is complete.

EXO=$\left\{\begin{matrix}\text{ON}\\\text{OFF}\end{matrix}\right\}$ Execute-only dataset. EXO=ON sets the execute-only status of a dataset. EXO=OFF clears the execute-only status. The status is ignored if you omit EXO.

SID=*mf* Default source mainframe identifier: 2 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: 2 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 job's ownership value.

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

Partial delete option (for more information on the partial delete option, see CRAY-OS Version 1 Reference Manual, publication SR-0011); 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=$\left\{\begin{matrix}\text{YES}\\\text{NO}\end{matrix}\right\}$ Access tracking option (for more information on the access tracking option see the CRAY-OS Version 1 Reference Manual, publication SR-0011); default is installation defined.

RP=$\left\{\begin{matrix}\text{YES}\\\text{NO}\end{matrix}\right\}$ Remove permit option; default is NO.

USR=*ov* Dataset user number

PERMANENT DATASET MANAGEMENT MACROS

You can access, save, adjust, delete, and control other users' access to your permanent datasets by using the permanent dataset management macros. The PDD macro must accompany all of these macros 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, including an ASSIGN control statement.

At least one of read, write, or maintenance permissions must be granted, either by default or by specifying the corresponding control word. If no permissions are granted, access is denied. If permanent dataset privacy is enabled, permission must be granted via either the PAM setting or the PERMIT mechanism for you to access the dataset if the ownership value is not the same as that of your job.

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 at least write permission 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. The dataset:

- Must be currently opened for output,
- Has not had an EOD written,
- Is being written sequentially,
- Has COS blocked dataset structure, and
- 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 you use it 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 at least maintenance permission and unique access before you issue a DELETE. If the dataset is a tape dataset, a request is made to the servicing front-end, if any, to remove the dataset from its catalog.

Format:

Location	Result	Operand
	DELETE	*pddtag*

pddtag Address of PDD

PERMIT - EXPLICITLY CONTROL ACCESS TO DATASET

The PERMIT macro allows you 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 you to issue a PERMIT. However, if you use an ADN parameter to define 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, ownership, and edition number. If the dataset is a tape dataset, a request is sent to the servicing front-end, if any, 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,
- Has not had an EOD written,
- Is being written sequentially,
- Has COS blocked dataset structure, and
- 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.

You can use these macros to:

- 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.

ENTRY BLOCK DESIGN

With the use of the CFT linkage macros, you can make the entry section of a CAL routine to resemble the entry area of a CFT subroutine. The DEFARG, DEFB, DEFT, ALLOC, MXCALLEN, and PROGRAM 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.

When you use these macros, you must specify them in the following order:

1. DEFARG macro
2. DEFB macro
3. DEFT macro
4. ALLOC macro
5. MXCALLEN macro
6. PROGRAM or ENTER 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 specified by the first DEFARG is assigned to the first argument, the symbol specified by the second DEFARG is assigned to the second argument, and so on. These symbols can be used in the ARGADD macro, which is defined later. The ENTER macro also uses the number of arguments defined to control the generation of the entry sequence.

Format:

Location	Result	Operand
name	DEFARG	

name Required 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. Using these registers does not cause overhead on entry or exit. You should not use temporary B registers if you are calling lower level routines, since the registers can be destroyed during the call.

Format:

Location	Result	Operand
name	DEFB	*explicit register designator*

name Required 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*.

This parameter is optional. 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 you do not assign T registers explicitly, the DEFT macro uses the next available T register. You can assign the T registers from either of two classes, temporary or nontemporary.

Temporary registers are neither saved nor preserved across calls. Eight temporary T registers are available (T70 through T77). Temporary T registers are not saved on entry to a subroutine; using these registers causes no overhead on entry or exit. Do not use temporary T registers if you are calling lower level routines, because the call may destroy the registers.

Format:

Location	Result	Operand
name	DEFT	*explicit register designator*

name Required 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*.

This parameter is optional. 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

Example (continued):

Location	Result	Operand	Comment
1	10	20	35
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 (stack) 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. In most cases, ALLOC uses the CAL BSS pseudo instruction (see CAL Assembler Version 1 Reference Manual, CRI publication SR-0000) to dynamically allocate stacks at run time. ALLOC does not use the BSS pseudo instruction when reentrant code is to be generated. 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. This parameter is required.

size Size in words of the area to be allocated. This field can be any valid CAL expression. This parameter is optional; the default is 1 word.

Example:

Location	Result	Operand	Comment
1	10	20	35
WIDTH	ALLOC		Allocates 1 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. You can calculate the length by checking all calls to call-by-address routines and determining the maximum argument list length used by any call. You only need this macro when you use dynamic stack management.

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. This parameter is required.

Example:

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

PROGRAM - DECLARE PROGRAM 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 you use this macro with stacks, it 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. This parameter is required.

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). This parameter is optional; the default for the stack pointer reqister is A7.

NOTE

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

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 be different from the register you specify for the STKPTR parameter. This is an optional parameter; the default value is A5.

SUBROUTINE LINKAGE

The subroutine linkage macros perform the following functions:

- CALL and CALLV external subroutines using the correct calling sequences for Cray products. These macros provide traceback information for error processing.
- ENTER sets up a subroutine entry point for CALL or CALLV to use. ENTER helps design the entry block (see ENTRY block design earlier in this section) and provides traceback information.
- EXIT performs any cleanup needed before returning to the calling routine, then returns control to the subroutine caller.

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. This is a required parameter.

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 you include several items in the argument list, you must separate the items with commas and enclose them in parentheses. This is an optional parameter.

NOTE

If you specify registers as arguments, the contents of these registers are stored in the argument list passed to the called routine. For example, if you specify 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 is an optional parameter; required 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, you must code a USE parameter to force a reload of the stack pointer value. If the USE parameter specifies an A register, do not use that register in the argument list. Likewise, do not use the USE and STKPTR parameters at the same time, because STKPTR specifies that the stack pointer is already available and USE specifies that the stack pointer must be reloaded.

If you specify neither STKPTR nor USE, 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. This is an optional parameter; required 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. 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 subroutine. 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. This parameter is required.

arglist A 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, separate the items with commas and enclose them in parentheses. This parameter is optional.

STKPTR=*stkptr*

An A register currently containing the value of the stack pointer. CALLV uses this A register as an index to compute memory addresses. This parameter prevents the stack pointer from being reloaded. This parameter is optional; required 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, you have to code a USE parameter to force a reload of the stack pointer value. If you specify an A register for the USE parameter, do not use that register in the argument list. Likewise, do not code the USE and STKPTR parameters 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 you specify neither STKPTR nor USE, 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 is needed to compute the addresses of local variables defined by the ALLOC macro. This is an optional parameter; required 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 arguments 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.

****

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. It also generates code conditionally for both types of CFT calling sequences and for reentrant entry sequences. The ENTER macro also optionally forces loads of argument values 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={ON / OFF},INSRTMAC=(*macname*,(*maclist*))

name Symbolic name associated with the entry point. This parameter is requried.

NP=*np* Number of parameters expected to be passed to the routine. This parameter is optional; 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* is the number of B registers used in addition to those used by the calling sequence. This parameter is optional; 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. This parameter is optional; the default is the number of nontemporary T registers defined using the DEFT macro.

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

USER mode entry is default.

LIBRARY mode entry is intended for special use in the libraries. It is a somewhat faster method, but is also more restrictive because LIBRARY mode makes more assumptions about registers being returned.

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. This parameter is optional; 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. This parameter is optional. If you specify this parameter, the named 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.

The NB and NT parameters cannot be used when you use SHARED. 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 you specify for the SHARED parameter must precede the current entry. You cannot use any unshared entries 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. This parameter is optional; 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. This parameter is optional; the default is ONEWORD.

When you specify ARGSIZE=ONEWORD, 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.

However, when you specify ARGSIZE=TWOWORD, arguments are assumed to consist of 2 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.

If you specify ARGSIZE=THREEWORD, preloading is similar to 2-word arguments, with the exception 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. This parameter is optional; 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). This parameter is optional; the default value for the stack pointer register is A7.

NOTE

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

SCR=*scr* Specifies an A register to be used as a scratch register during the entry sequence. This parameter is optional. The valid options are A1 through A5 and A7. The scratch register must be different from the register you specify for 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. You can code an SCR value on each of the macros, which overrides the value on the ENTER macro, or code the default of A5 for that macro invocation only.

COPYIN=$\begin{Bmatrix}ON\\OFF\end{Bmatrix}$

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 is optional and 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*))

This parameter is optional and is only required when you need special processing performed before the standard entry sequence is executed. Cases where you may need special processing include those such as 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. You 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; however, if you include it, *maclist* must follow *macname* and be enclosed in parentheses, nested if necessary.

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 2 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.

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. This parameter is optional; 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. This parameter is optional; the default is the number of T registers saved by the last ENTER macro.

MODE=*mode* Exit sequence to be generated. *mode* is optional and can be one of the following: USER, LIBRARY, and BASELVL; USER mode exit is default.

LIBRARY mode exit is for special use in the libraries. It is a faster exit method, because LIBRARY mode makes more assumptions about registers being returned.

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. This parameter is optional. If you use MODE=BASELVL on entry, then on exit you have to use MODE=BASELVL also.

****

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. This parameter is optional; 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 you code KEEP=ON, 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. This parameter is optional; 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. This parameter is optional; the default is to return to the calling routine.

INSRTMAC=(*macname*,(*maclist*))
INSRTMAC is optional and is required only when special processing is to be performed before the standard exit sequence is executed. Cases where this might be necessary include those such as restoring registers before exit or clearing a hardware semaphore. The code to be inserted is placed immediately following the standard exit sequence and immediately preceding the jump to B00. You 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. *maclist* is optional; however, if you include it, have it follow the *macname* and enclose it 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 performs 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.

Examples (continued):

Location	Result	Operand	Comment
1	10	20	35
	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.

ARGUMENT LIST INFORMATION

Two macros provide information about passed-in dummy arguments for call-by-address routines: ARGADD and NUMARG. The ARGADD macro returns the addresses of passed in arguments from the calling list, whereas 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. You can only use the ARGADD macro when you have used an ENTER macro first. You should refer to arguments 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. This parameter is required. 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 leftmost 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* is required and 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 must 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. This parameter is optional.

ARGPTR=*argptr*
An A register currently containing the address of the argument list. This parameter is optional and 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 you specify neither the ARGPTR nor the USE parameter, the macro attempts to use the result register to recover the argument list pointer. If the result register is A0 or an S register, the macro 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 the calling routine properly used the CALL or CALLV macro (see explanation earlier in this section). You can only use the NUMARG macro when you have used the ENTER macro first for defining 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. This parameter is required.

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

ARGPTR=*argptr*
An A register currently containing the address of the argument list. This parameter is optional and 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 you specify neither the ARGPTR nor the USE parameter, the macro attempts to use the result register to recover the argument list pointer. If the result register is A0 or an S register, the macro 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).

Example (continued):

Location	Result	Operand	Comment
1	10	20	35
	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.

LOCAL VARIABLE STORAGE

Three macros reference temporary memory storage space defined by the ALLOC macro, explained previously 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. However, the space is acquired at run time from a general pool when stacks are being used. You can use the three macros, LOAD, STORE, and VARADD, 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 MEMORY VALUE

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. This parameter is required.

dataname Name of the memory location to be loaded. This parameter is required and 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. This parameter is optional.

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 optional and is required 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 you specify a B register, this 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 you specify neither STKPTR nor USE, 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. This parameter is optional and is required 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 parameter is optional and is required only when vector registers are to be loaded or with the INDEX parameter. If no register is specified in these cases, a warning message is issued and the scratch register defined by the ENTER macro is used; the default is A5.

SKIP=*skip* Specifies an A register containing the skip distance for a vector load. This parameter is optional; the default value 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 VALUE 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. This parameter is required.

dataname Name of the destination memory location. This parameter is required and 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. This parameter is optional.

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 optional and is required 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. This parameter is optional and is required 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. You only need this register when you want vector registers stored or when you use an INDEX parameter. This parameter is optional; if you do not specify a register in these cases, a warning message is issued and the scratch register defined by the ENTER macro is used. The default for SCR is A5.

SKIP=*skip* Skip distance specifies an A register containing the skip distance for a vector store. This parameter is optional; the default 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 - GET MEMORY ADDRESS

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. This parameter is required.

dataname Name of the memory location address to be found. This required 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 optional and is required 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, you must code a USE parameter to force a reload of the stack pointer value. If you specify an A register for the USE parameter, that register contains the current stack base address after the macro has been executed. On the other hand, if you specify a B register, the B register is used as temporary storage to recover the stack pointer. Its contents are indeterminate after the VARADD macro.

Do not code the USE and STKPTR parameters 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 you code neither STKPTR nor USE, 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. This parameter is optional and is required 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 parameter is optional and is required only when S registers or register A0 are to receive the memory address. If you do not specify a register 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.

STRUCTURED PROGRAMMING MACROS 6

The structured programming macros in this section 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 6-1 describes the conditions for structured programming macros that use such conditions.

Table 6-1. Conditions for structured programming macros

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 6-1. Conditions for structured programming macros (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* isalso 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.

† Specify the A micro (S micro) or the A*n* (S*n*) format, because the A.*symbol* (S.*symbol*) format causes assembly problems.

†† 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 6-1. Conditions for structured programming macros (continued)

Condition		Meaning
Relational conditions		
S_n[=*operand*], *cond*,Sm[=*operand*]		
	EQ	S_n EQ S_m
	NE	S_n NE S_m
	GT	S_n GT S_m
	GE	S_n GE S_m
	LT	S_n LT S_m
	LE	S_n LE S_m
A_n[=*operand*], *cond*,A_m[=*operand*]		
	EQ	A_n EQ A_m
	NE	A_n NE A_m
	GT	A_n GT A_m
	GE	A_n GE A_m
	LT	A_n LT A_m
	LE	A_n LE A_m

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 COS-independent structured programming macros presented in this section include: $GOTO; $IF, $ELSEIF, $ELSE, and $ENDIF; $JUMP; $LOOP, EXITLP, and $ENDLOOP; $GOSUB; $RETURN; and $SUBR.

$GOTO - COMPUTE 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
	$GOTO	Ai,(*label*$_0$, *label*$_1$...*label*$_n$),Aj

Register A0 cannot be used as A*i* or 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* or less than zero, no jump takes place, and control falls through to the next instruction.

$IF, $ELSEIF, $ELSE, AND $ENDIF - FORM CONDITIONAL BLOCK

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 you include both, an $ELSEIF macro cannot follow an $ELSE macro at the same nesting level. There can be no more than one $ELSE at the same nesting level in an $IF/$ENDIF sequence.

Format:

Location	Result	Operand
	$IF	*cond*
	$ELSEIF	*cond*
	$ELSE	
	$ENDIF	

See table 6-1 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 the condition is true, the block that follows is executed; otherwise, if the condition is false the block that follows 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 - JUMP CONDITIONALLY

You can use the $JUMP macro 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. However, if the condition does not hold, control flow continues on the statement 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 6-1.

Example:

```
Location  Result    Operand              Comment
1         10        20                   35

          $JUMP     ERROR,SM
          $JUMP     BADDAY,S1,LT,S2=1
```

$LOOP, $EXITLP, AND $ENDLOOP - DEFINE PROGRAM LOOP

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. $EXITLP exits the loop at its nesting level to the next executable instruction, but does not exit the entire 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.

Format:

Location	Result	Operand
	$LOOP	*cond*
	$EXITLP	*cond*
	$ENDLOOP	
cond	This expression is of the same form as conditional expressions	described in table 6-1.

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 first 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
```

$GOSUB - CALL LOCAL SUBROUTINE

The $GOSUB macro calls a subroutine that is local to the current assembly module. Use a CALL or CALLV macro when the subroutine you are calling is external. This macro differs from the CALL and CALLV macros, because 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 $GOSUB.

Format:

Location	Result	Operand
	$GOSUB	*label,cond*

label	Required name of the entry point of the local subroutine. Declare *label* using a $SUBR macro.
cond	Optional conditional expression that must be true for the call to be executed. (See table 6-1.) This expression is optional.

$RETURN - 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. See also $GOSUB previously in this section.

Format:

Location	Result	Operand
	$RETURN	*Bregname*

Bregname	Optional name of a B register containing the return address. If you omit *Bregname*, then B00 is used for the return. The B register name you specify should be the same name as the one you specify on the $SUBR macro used to enter this routine; furthermore, the name should be defined using a DEFB macro. Note that you should only specify the name and not the full B register designation (for example, RETADDR instead of B.RETADDR).

$SUBR - DECLARE LOCAL SUBROUTINE ENTRY POINT

The $SUBR macro declares a local subroutine entry point. You can also use this macro to specify a B register for storing the return address. If you indicate a B register 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. Also, this macro neither uses the standard CFT subroutine linkage nor is reported in the traceback if an error occurs. See also $GOSUB previously in this section.

Format:

Location	Result	Operand
label	$SUBR	*Bregname*

label Required name of this entry point of the local subroutine

Bregname Optional name of a B register to save the return address. If you omit *Bregname*, the return address is only contained in B00, which must not be destroyed if a proper return is to be made. Define the B register name using the DEFB macro. Note that you should only specify the name and not the full B register designation (for example, RETADDR instead of B.RETADDR).

SEMAPHORE MANIPULATION MACROS 7

Hardware semaphore (SM) registers provide interprocessor communication and coordination for all models of CRAY X-MP Computer Systems. EXEC uses these semaphore registers to ensure that only one CPU is active in either EXEC or in the STP area. (See the COS EXEC/STP/CSP Internal Reference Manual, publication SM-0040, for additional information on SM registers.)

With the semaphore manipulation macros you can define semaphore names symbolically and use the CRAY X-MP hardware semaphores. These macros generate executable code on CRAY X-MP systems. Code generation is controlled conditionally by use of the CPU= parameter on the CAL control statement. Indicate CPU='CRAY-XMP' to produce executable code.

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
- GETSM — Gets current status of semaphore bit
- SETSM — Unconditionally sets a semaphore
- CLRSM — Unconditionally clears a semaphore
- TEST$SET — Tests and sets semaphore
- WAIT$CLR — Unconditionally clears a semaphore after waiting for memory references to complete

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 you do not specify a number 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. Semaphore bit 16 is assigned if a DEFSM with no number specified is the first DEFSM to be used. 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 The required symbol to be assigned to the semaphore bit. This must be 5 characters or less.

number An optional number designating the semaphore bit to be assigned. This must be a number in the range 0 to 31. If you do not specify a number, the semaphore bit following the last previous bit defined is assigned. If this is the first use of a DEFSM macro and number unspecified, 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

GETSM - GET SEMAPHORE BIT STATUS

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 another processor before you can check its value. Do not use the GETSM macro in place of a TEST$SET macro; GETSM is intended for signaling purposes.

Format:

Location	Result	Operand
	GETSM	*name*,REG=*reg*

name The required name of the semaphore to interrogate. You must have defined this name using the DEFSM macro.

REG=*reg* An optional 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 you do not specify a REG= clause, the result is returned in S0.

Example:

Location	Result	Operand	Comment
1	10	20	35
	GETSM	SIGNL	Gets signal value in S0

SETSM - SET SEMAPHORE WITHOUT WAITING

The SETSM macro causes a semaphore set instruction to be generated for the CRAY X-MP computer. This instruction sets a semaphore bit to 1 unconditionally, without regard to its prior state. You must have defined the named semaphore using the DEFSM macro.

Format:

Location	Result	Operand
	SETSM	*name*

name Required 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

CLRSM - CLEAR SEMAPHORE WITHOUT WAITING

The CLRSM macro causes a semaphore clear instruction to be generated for the CRAY X-MP computer. This instruction clears a semaphore bit to 0 unconditionally, 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 Required name of the semaphore to clear. You must have defined this name using the DEFSM macro.

Example:

Location	Result	Operand	Comment
1	10	20	35
	CLRSM	CODLK	Allows other processor to enter

TEST$SET - TEST SEMAPHORE AND WAIT TO SET

The TEST$SET macro generates a CRAY X-MP semaphore test and set instruction. The test and set instruction causes a processor to set the semaphore if it is clear, or to stop processing instructions until the sempahore is clear, at which time the semaphore is set and instruction processing resumes. This macro generates test and set instructions for the semaphores defined by the DEFSM macro.

Format:

Location	Result	Operand
	TEST$SET	*name*

name Required name of the semaphore to test and set. You must have defined this name using the DEFSM macro.

Example:

Location	Result	Operand	Comment
1	10	20	35
	TEST$SET	CODLK	Stops the processor issuing the instruction until SM.CODLK is 0

WAIT$CLR - CLEAR A SEMAPHORE BIT AFTER WAITING

The WAIT$CLR macro causes a CMR instruction and generates a semaphore clear instruction for the CRAY X-MP Computer System. This instruction unconditionally clears (sets to 0) a semaphore bit after waiting for memory references to complete without regard to its prior state. The semaphore to clear must have been defined using the DEFSM macro.

Format:

Location	Result	Operand
	WAIT$CLR	*name*

name Required name of the semaphore to clear. *name* must have been defined using the DEFSM macro.

Example:

Location	Result	Operand	Comment
1	10	20	35
	WAIT$CLR	CODLK	Clear code lock

CAL EXTENSION OPDEF AND MACROS 8

CAL extension opdef and macros provide various means for enhancing the Cray Assembly Language's capabilities. This opdef and group of macros include:

- DIVIDE Provides a precoded divide routine
- PVEC Passes elements of a vector register to a scalar routine
- $CYCLES Generates timing-related symbols and constants
- $DECMIC Converts an integer to a decimal micro string
- $OCTMIC Converts an integer to an octal micro string
- RECIPCON Generates floating-point reciprocals

DIVIDE OPDEF - PROVIDE A PRECODED DIVIDE ROUTINE

The DIVIDE opdef enhances the instruction repertoire of the CAL assembler by providing a precoded floating-point 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 and must contain normalized floating-point values.

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 available for 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, which is necessary when there are no purely vectorized versions of arithmetic algorithms. 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 Required name of the scalar routine to call

vec$_1$ An optional vector register or 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$_2$ An optional vector register or 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 An optional vector register or 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 An optional vector register or 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 you only specify vec_1, the return value of S1 from the scalar routine is stored. If you specify vec_1 and vec_2, the values of S1 and S2 returned by the scalar routine are both stored.

LENGTH=*length*

Optional. The length of the vector to be processed. *length* can be passed as a constant or in an A, S, or B register. (Do not use registers A0 and S0.) If you do not specify *length*, 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. You 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 common deck in COSTXT or $SYSTXT 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* *type* can be:
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 AN INTEGER TO A MICRO STRING

$DECMIC converts any integer that can be held in a CAL symbol to a MICRO string.

Format:

Location	Result	Operand
micname	$DECMIC	*value*[,length]

micname Required micro string name. *micname* is assigned the character string that corresponds to the decimal value of the expression supplied to the macro. If the value is negative, this string has a leading minus sign.

value A required expression that is converted into a decimal micro string.

length Optional micro string length. If length is unspecified, the leading zeros are truncated, and only the significant digits are returned.

If a length is specified, the result string is zero filled to the specified length, If the result string is too small to contain the converted value, a warning message is issued. If the length is greater than 20, a warning message is issued, and the length is reduced to 20 characters.

Example:

Location	Result	Operand	Comment
1	10	20	35
LONGMIC	$DECMIC	D'1234567,D'15	Converts to micro form
	DATA	'"LONGMIC"'Z	Use of micro
	DATA	'0000000001234567'Z	Micro expansion

$OCTMIC MACRO - CONVERT AN INTEGER TO AN OCTAL MICRO STRING

$OCTMIC converts any integer that can be held in a CAL symbol to a MICRO string.

Format:

Location	Result	Operand
micname	$OCTMIC	*value*[,*length*]

micname Required micro string name. *micname* is assigned the character string that corresponds to the octal value of the expression supplied to the macro. If the value is negative, it is represented in two's complement form.

value A required expression that is converted into an octal micro string.

length Optional micro string length. If length is unspecified, the leading zeros are truncated, and only the significant digits are returned.

If a length is specified, the result string is zero filled to the specified length, If the result string is too small to contain the converted value, a warning message is issued. If the length is greater than 22, a warning message is issued, and the length is reduced to 22 characters.

Example:

Location	Result	Operand	Comment
1	10	20	35
LONGMIC	$OCTMIC	O'7654321,D'10	Converts to micro form
	DATA	'"LONGMIC"'Z	Use of micro
	DATA	'0007654321'Z	Micro expansion

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)

SUBSYSTEM SUPPORT MACROS 9

Subsystem Support macros provide a mechanism for the user to develop code that would otherwise have to be incorporated as part of the Cray Operating System (COS). Examples of this kind of code are networking packages and online diagnostics.

Subsystem Support macros are a collection of independent functions whose use may be restricted to jobs granted the necessary privilege by COS.

This section describes the following Subsystem Support macros:

- Interjob communication (IJCOM)
- User channel access (DRIVER)
- Event recall (ERECALL)

INTERJOB COMMUNICATION (IJCOM)

The interjob communication (IJCOM) macro allows a job to communicate with another job. This feature is available to all single-tasking jobs; it is prohibited to multitasking jobs.

Format:

Location	Result	Operand
	IJCOM	*ijpbadd*

ijpbadd A symbol or an A (not A0), S (not S0), or T register containing the address of the interjob communication parameter block (IJPB). For a complete description of the IJPB parameters, see appendix A of the CRAY-OS Version 1 Reference Manual, publication SR-0011. Up to an installation defined maximum number of parameter blocks (I@MPBS) can be linked together by setting IJLINK in IJPB. In all cases IJFUNC, IJRID, and IJPLEN are required in IJPB. IJSTAT is returned in IJPB by the system. *ijpbadd* is a required parameter.

Table 9-1 lists interjob communication (IJCOM) functions enabled by the IJFUNC parameter of the IJPB and briefly describes the operation of each.

Table 9-1. IJCOM functions

Function	Description
IJM$NOP	Is ignored
IJM$REC	Marks the job as receptive. IJRCB is required in IJPB; all other fields are ignored.
IJM$OPEN	Initiates an attempt to open a communication path with another job. IJHLEN, IJTID, and IJNCB are required in IJPB; all other fields are ignored.
IJM$ACCE	Accepts a request from another job to opencommunication. IJTID, IJHLEN, and IJNCB are required in IJPB; all other fields are ignored.
IJM$REJE	Rejects a request from another job to open communication. IJTID is required in IJPB; all other fields are ignored.
IJM$SNDM	Sends a message to another job. IJNCB, IJTID IJBADD and IJBLEN are required in IJPB; all other fields are ignored.
IJM$SNDL	Sends a message to an attached job's logfile. This is a privileged function. IJTID, IJOVR, IJFCS, IJFCU, IJCLS, and IJBADD are required in IJPB; all other fields are ignored.
IJM$CLOS	Closes a communication path. Either IJNCB and IJTID or neither, are required in IJPB. All other fields are ignored. If IJNCB and IJTID are specified only the path determined by IJRID and IJTID is closed; otherwise all communication paths with IJRID are closed.
IJM$END	Ends a job's receptivity. All other fields are ignored. Existing communication paths are not affected.

On return from the IJCOM macro, the following status conditions can exist:

Register	Description
S0=0	No errors occurred
S0≠0	IJSTAT contains the error code. If linked parameter blocks are used, all IJSTAT fields must be examined if S0≠0.

USER CHANNEL ACCESS (DRIVER)

Use the DRIVER macro to directly access a Cray channel on the I/O Subsytem (IOS) MIOP. This feature is available to privileged single-tasking jobs, and prohibited from multi-tasking jobs.

Format:

Location	Result	Operand
	DRIVER	*drpbad*

drpbad A symbol or an A (not A0), S (not S0), or T register containing the DRIVER parameter block (DRPB). For a complete description of the DRPB parameters, see the CRAY-OS Version 1 Reference Manual, publication SR-0011. In all cases DRFUNC, DRPLEN, and DRLN are required in DRPB and DRCOSS is returned to the user. See individual driver specifications for the use of this word and other field requirements. *drpbad* is a required parameter.

Table 9-2 lists the DRIVER functions enabled by the DRFUNC parameter of the DRPB and briefly describes the operation of each.

Table 9-2. DRIVER functions

Function	Description
CFN$OPE	Opens a channel; a job cannot access a channel until it opens the channel. DRDRNM, DRTO, DRDIR, and DROPD are required; all other fields are ignored.
CFN$CLS	Closes a channel. Any open channels are closed during job termination. DRDIR is required; all other fields are ignored.
CFN$RD	Reads data. DRBAD and DRLN are required; DRTLN is returned.
CFN$RDH	Reads and holds data. DRBAD and DRLN are required; DRTLN is returned. A read and hold makes a second read from the channel, and holds the data in Buffer Memory for a subsequent read.
CFN$RDD	Reads and rereads data. DRBAD and DRLN are required; DRLN is returned. A read and reread makes a second read to Central Memory.

On return from the DRIVER macro, the following conditions can exist:

Register	Description
S0=0	No errors occurred. The job must poll DRCOMS for nonzero indicating the driver has completed the request. When DRCOMS is nonzero, the driver status is in DRDRS. See the individual driver specifications for driver statuses.
S0≠0	DRCOSS contains the error code and the request is not sent to the driver.

EVENT RECALL (ERECALL)

The ERECALL macro allows a job to suspend itself until one or more selected events occur. This feature is available to all single-tasking jobs; it is prohibited to multi-tasking jobs.

When event monitoring is enabled, the system monitors selected events for a job, keeping track of the events that have occurred. Monitoring is disabled at the beginning of each job step and can be enabled by making a system request specifying the events to monitor. Once monitoring is enabled, a job can make a system request to do the following: change the events that are to be monitored, get a map indicating the events that have occurred, go into event recall, or disable monitoring.

When monitoring is enabled, a map of occurred-events is returned to the user and discarded by the system. If monitoring was disabled when the enable occurred the map is zero.

When events to be monitored are changed or a map of occured events is requested, a map of occured events is returned to the user and discarded by the system.

When recall is requested and the map of occurred events is zero, the job is suspended for an event until one of the events occurs. If the map is nonzero, the map is returned to the user immediately and discarded by the system.

When recall is disabled, the map of occurred-events is discarded by the system.

Format:

Location	Result	Operand
	ERECALL	*erpbadd*

erpbadd A symbol or an A (not A0), S (not S0), or T register containing the address of the event recall parameter block (ERPB). For a complete description of the ERPB parameters, see the CRAY-OS Version 1 Reference Manual, publication SR-0011. In all cases ERFUNC is required in ERPB. *erbpadd* is required.

Table 9-3 lists the event recall (ERECALL) functions enabled by the ERFUNC parameter of the ERPB and briefly describes the operation of each.

Table 9-3. ERECALL functions

Function	Description
ERCL$DIS	Disables event monitoring; all other fields are ignored.
ERCL$ENA	Enables event monitoring or changes the events to be monitored. ERMASK is required. If ERMASK=0, timeout is the only enabled event. Timeout is always enabled in order to prevent a job hanging forever in recall. ERMAP is returned by the system. ERTO is ignored.
ERCL$RCL	Places the job in recall. An error is returned in S0 if monitoring is disabled. ERTO is required, ERMASK is ignored, and ERMAP is set by the system. If ERTO=0, an installation-defined default value, I@TODEF, is used. If ERTO is specified but is less than I@TOMIN, the installation-defined minimum is used without notification. If ERMAP=0 on return, timeout is the only event that occurred.
ERCL$RET	Requests the system to set ERMAP; all other fields are ignored. An error is returned in S0 if monitoring is disabled.

PART 2

SYSTEM AID MACROS AND OPDEFS

NORMAL TABLE MANIPULATION 1

A table is an area within a program that provides a means of associating various bits of memory to fields or subfields. Tables consist of one or more fields. A field, moreover, is a contiguous set of bits in Central Memory.

When you use CAL, you can simplify access to data structures by defining tables. You can access a field with only one macro instruction by using the table manipulation macros and opdefs. The fields of a table are then accessed by name rather than by address.

The table manipulation macros and opdefs described in this section allow you to orient the fields to Central Memory word boundaries. The tables, which are oriented to the physical size of a Central Memory word (that is, 64 bits), are referred to as normal tables and are governed by the normal table macros and opdefs.

Tables that allow fields to span Central Memory word boundaries are called complex tables. Complex tables are defined by the complex table macros and opdefs described in part 2, section 2.

Normal table manipulation macros and are divided into the following main classes:

- Table definition macros
- Run-time field management opdefs
- Miscellaneous run-time field management opdefs

TABLE DEFINITION MACROS

Tables are defined during the program's assembly. You define the table's parts by using a macro that corresponds to the specific aspect to be defined. The macros in turn define assembly symbols that reflect the characteristics of what has been defined. These symbols are then used by the construction and manipulation macros and opdefs.

Both normal and complex tables consist of three basic categories: a header, one or more entries, and overall characteristics. However, the normal table macros and the complex table macros must be used separately for definition and table structure maintenance. Mixing normal and complex table macros results in assembly errors. Complex tables are described in part 2, section 2.

You can construct tables that use 64-bit words by using the following macros:

- TABLE — Defines the overall table attributes
- CAPTION — Prints a caption underneath a diagram
- ENDTABLE — Designates the end of a table definition
- FIELD — Defines the field within the current table structure, or sets up special labels for fields
- FIELD@ — Defines a field with attributes identical to a previously defined field
- ENDFIELD — Designates the end of a field spanning more than 1 word
- SUBFIELD — Identifies fields contained within a larger field
- 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
- BUILD — Constructs a table at assembly time

TABLE - DEFINE TABLE ATTRIBUTES

The TABLE macro defines symbols that relate to a table; however, TABLE neither creates the table (see BUILD), nor checks for the presence of a label. This macro defines special symbols for the length of the table header (LH@*name*), the length of each entry in the table (LE@*name*), the number of entries in the table (NE@*name*), and the size of the table (SZ@*name* and L@*name*).

Format:

Location	Result	Operand
name	TABLE	LH=*lh*,LE=*le*,NE=*ne*,{L=*lt* | SZ=*st*}

All parameters are optional. Nonetheless, note that without a name, coding any parameter can produce unexpected symbols.

name The 1- to 5-character name of the table being described. This parameter is required if LH, LE, NE, L, or SZ is specified.

LH=*lh* Length of table header in words. If the table has a header, LH generates an assembler label, LH@*name*, which contains the length of the header. This parameter is optional.

LE=*le* Length of entry in words. This parameter is optional.

NE=*ne* Number of entries in the table. Coding NE for the TABLE macro can be helpful only if NE has not been defined already in a systems text. If NE has been specified already, with a value different from the one given here, CAL generates a double-defined error. This parameter is optional.

{L=*lt* \| SZ=*st*} Table length in words. This parameter is optional.

Example:

Location	Result	Operand	Comment
1	10	20	35
DRT	TABLE	LH=3,LE=67,NE=2,SZ=LH@DRT+LE@DRT*NE@DRT	

You can use the TABLE macro to introduce a table for the Table Diagram Generator (TDG). See the Table Diagram Generator (TDG) Reference Manual, CRI publication SM-0075 for more information about the TDG.

CAPTION - DECLARE TABLE TITLE

For each table diagram drawn by the Table Diagram Generator (TDG), the TDG prints a caption underneath the diagram and at the bottom of each page from which the diagram is continued. You provide the caption by coding a CAPTION statement†.

A CAPTION statement is required to inform the TDG that the records following are to be considered part of a table. An ENDTABLE or ENDTEXT statement causes the TDG to ignore records up to the next CAPTION statement. A TABLE statement can also turn on TDG processing, but the resulting table's diagram contains inadequate labeling information because of a lack of a CAPTION statement. For more information about the TDG, see the Table Diagram Generator (TDG) Reference Manual, CRI publication SM-0075.

Format:

Location	Result	Operand
mne	CAPTION	*pfx*,'*caption text*'[,WORD='*ttt*']

mne The table mnemonic. *mne* should be the same for each CAPTION statement that is included within a single table definition. In many cases, the *mne* value should be the same as the value of the location field of the preceding TABLE statement that starts the table definition.

pfx The table prefix. *pfx* is a 2 or 3 character alphanumeric prefix that precedes every field label in the table.

'*caption text*'
A quoted string describing the table. The first caption statement for a table is used in page headers for all pages of the table.

WORD='*ttt*'
The WORD keyword parameter prefixes the string '*ttt*' to each word number printed by the TDG. This applies to tables which have variable length components, in whose case it is appropriate to refer to words as, for example, x+0 or x+1, relative to a given starting word x.

† The CAPTION statement may also be used in complex tables.

Example:

The following is a typical CAPTION entry for a table:

Location	Result	Operand
APT	CAPTION	AP,'ANY PACKET TABLE'

ENDTABLE - END TABLE DEFINITION

The ENDTABLE macro specifies the end of a table description. If you did not define a LE@*name*, 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 referenced table 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. This parameter is optional. This parameter is optional; if *name* is specified, LE@*name* is specified.

The ENDTABLE statement terminates all preceding TABLE statements for the Table Diagram Generator (TDG). See the Table Diagram Generator (TDG) Reference Manual, CRI publication SM-0075 for a detailed description of the TDG. Until a new TABLE statement is encountered, the following statements after an ENDTABLE statement are flagged as errors: FIELD, SUBFIELD, NEXTWORD, REDEFINE and ENDTABLE.

FIELD - NAME FIELD WITHIN TABLE

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 (*sbit* and *number*) and may not overlap. 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*

name A 1- to 6-character name of the field being described. The first few characters should be the table prefix, as required by the BUILD macro. For processing with the Table Diagram Generator (TDG)†, it is not necessary to start the name with the table prefix. Nonetheless, the table diagram's appearance improves when the proper prefix is used. This parameter is required.

word Relative 64-bit word index in table or table entry in which the field resides. This parameter is required. 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. This parameter is required. * suppresses the S@*name* and N@*name* definitions; *number* must be omitted if * is used.

† For a complete description of the TDG, see the Table Diagram Generator (TDG) Reference Manual, CRI publication SM-0075.

For the TDG, if you code *sbit* as '*' the length of the field is ambiguous. The TDG defers drawing the field in the diagram until a CAL ENDFIELD† directive is encountered. The TDG calculates the length of the ambiguous field to cover all bits defined by FIELD or NEXTWORD, between the ambiguous FIELD statement and the ENDFIELD statement.

number Field size in number of bits. This parameter is required when *sbit* has been given a numeric value.

Example:

Location	Result	Operand	Comment
1	10	20	35
COMDN	TEXT	'DN Demonstration table - DNT'	
DNT	TABLE		
DNBSZ	FIELD	0,6,24	BSZ field
DNDAT	FIELD	1,24,40	DAT field
DNBFZ	FIELD	3,12,15	BFZ field
DNUSR	FIELD	16,*	USR field
DNXYA	FIELD	*,24,40	Part of USR
DNUSR	ENDFIELD		Required
DNT	ENDTABLE		
	ENDTEXT		

FIELD@ - EQUATE A NEW FIELD TO A PREVIOUSLY DEFINED FIELD

The FIELD@ macro sets up the special labels for a field from values previously defined for another field. Only the special labels already defined are set up for the new field. This macro is used to ensure that two field definitions are identical.

Location	Result	Operand
name	FIELD@	*name*1

Expansion:

W@*name*=W@*name*1
S@*name*=S@*name*1
N@*name*=N@*name*1

† The ENDFIELD statement may also be used in complex tables.

name A 1- to 6-character name of the field being described. This parameter is required.

*name*1 A 1- to 6- character name of a previously defined field. This parameter is required.

ENDFIELD - DEFINE THE END OF A FIELD

The ENDFIELD macro terminates a field definition that begins with a special form FIELD statement. See the Table Diagram Generator (TDG) Reference Manual, CRI publication SM-0075 for more information about the ENDFIELD and special form FIELD statements.

Format:

Location	Result	Operand
	ENDFIELD	

SUBFIELD - NAME PART OF A FIELD

The SUBFIELD macro allows fields to overlap when two or more fields use the same bit range. Use this macro only when overlapping fields are logical subfields. The DPERR field, for example, 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 you define a subfield using the FIELD macro instead of the SUBFIELD macro, the result is an assembly error.

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. This parameter is required.

sbit Number identifying the leftmost bit of the field. This parameter is required.

length Number identifying the field length in bits. This parameter is optional. If it is not specified, the default is 1.

W@*name*, S@*name*, and N@*name* are defined. W@*name* uses the word number from the most recent FIELD macro.

Comment processing for SUBFIELD statements is equivalent to that for the FIELD statement. The diagram annotation for subfields is indented two spaces relative to other annotations.

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

NEXTWORD - ADVANCE SPECIFIED NUMBER OF WORDS

The NEXTWORD macro allows you to add 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*. Use this macro with a special form of the FIELD macro to maintain the current word number automatically.

Format:

Location	Result	Operand
name	NEXTWORD	*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 you omit *length*, or if it is 1, and *name* is present, S@*name* is defined as 0 and N@*name* is defined as 64. No symbols are defined unless you indicate *name*.

word Optional word number to reset counters. If you omit *word*, counters are set to the next sequential word resulting from a previous NEXTWORD macro or FIELD macro. The default value is the current value of the current word counter, plus the *length* from the most recently encountered NEXTWORD or REDEFINE macro. The default value can also be the current value of the current word counter +1 if a FIELD macro is encountered which specifies a numeric word value or a +, since the NEXTWORD or REDEFINE.

length Optional size of block being defined. If *length* is omitted, 1 word is assumed. *length* is used when a block larger than 1 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, a following NEXTWORD or a following FIELD macro with the + option 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
COMXX	TEXT	'XX Demonstration table - TAB'	
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.

Location	Result	Operand	Comment
1	10	20	35
	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 SPECIFIED NUMBER OF WORDS

Use the REDEFINE macro to redefine the words within a table definition that has multiple formats of one or more words. For example, you can use this macro for the Permanent Dataset Definition Table (PDD), whose formats change according to 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 you omit *length*, or if it is 1, and *name* is present, S@*name* is defined as 0 and N@*name* is defined as 64. No symbols are defined unless you indicate *name*.

word The beginning word number of the range to be defined. If you omit *word*, 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 (for example, the Dataset Name Table (DNT) portion of a System Dataset Table (SDT) entry). The default value is 1.

The REDEFINE statement terminates the current table generated by the Table Diagram Generator (TDG). The diagram and annotation up to the occurrence of the REDEFINE statement are drawn. A new diagram is then started and its first word is the one specified by the first parameter of the REDEFINE statement. To avoid redefinitions meant to define subfields, use the SUBFIELD macro described previously in this subsection. For a detailed description of the TDG, see the Table Diagram Generator (TDG) Reference Manual, CRI publication SM-0075.

BUILD - CONSTRUCT DEFINED TABLE

Use the BUILD macro 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 to set initial values (see the CAL Assembler Version 1 Reference Manual, CRI publication SR-0000). Fields you specify in the BUILD macro must not overlap or be repeated, although these conditions are not diagnosed. Any field that you do not name explicitly in the BUILD macro is initialized to 0.

Format:

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

oplabel Optional location symbol. If present, *oplabel* is defined as the location of word 0 of the table being built and has the CAL word address attribute. 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. *pfx* is the prefix used in the FIELD macro (such as, DP for DSP or EQ for EQT). This parameter is required.

NOTE

Use of the BUILD macro requires that all table 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. This parameter is required.

list Field names and values to be entered into the fields. You must enclose the list in parentheses, unless only one field name and value is specified or no field is specified. If you do not specify a field, the table is initialized with all zeros. This parameter is optional.

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 fields A and B for the table header.

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

c. These entries define fields X, Y, and Z for the table entry.

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

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=0'37)	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. The field names are described in the COS Table Descriptions Internal Reference Manual, publication SM-0045.

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)	

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   |         |
```

RUN-TIME FIELD MANAGEMENT OPDEFS

Run-time field management opdefs allow you to manage information contained in a table during program execution. These opdefs primarily allow the following:

- Field retrieval
- Field modification

Each opdef is structured so that a minimum number of instructions are generated to produce its major function or functions. You can optimize memory references still further by using the following format catagories:

- Fast format uses a minimum number of instructions and registers (omits register S*j*) to perform the opdefs' function.
- Full format performs required memory reads within the opdef.
- Quick format assumes required memory reads have been made by another opdef, usually a full opdef, using register A*n*.

To designate the desired optimization category, enter one of the following formats:

Location	Result	Operand	Comment
	name,R*i*	S*k*,*field*,A*n*	Fast category
	name,R*i*	S*j*&S*k*,*field*,A*n*	Full category
	name,R*i*	S*j*&S*k*,*field*	Quick category

name Name of the opdef to be optimized

R*i* An address register, scalar register, or an assembly symbol that receives or supplies the field value.

S*j* Base image register. On exit from the opdef, S*j* contains the memory image in which the field resides. The quick opdef requires the referenced field to reside in the register on entry into the opdef.

S*k* Scratch register. S*k* is used in masks or shifts to perform the opdefs major function. No special value is associated with this register upon entry or exit.

A*n* Entry base address register. Register A*n* specifies the memory base address of the desired table entry.

field Field names the field associated with the operation. *field* must have been previously defined in a table structure.

For all of the run-time field management opdefs (GETF, GET, SGET, PUT, SPUT, SET) and miscellaneous run-time field management opdefs (LOAD, STORE, ASSIGN, LJF, and XFER) opdefs, an optional location field can be specified. All of the other parameters are required.

When you use the quick opdefs, validate the implied field assumptions in order to avoid a run-time misinterpretation of data.

To aid programming, the run-time field management opdefs optionally compare the word offset assigned in the the base image register of full form opdefs, and the word offset of fields used by quick form opdefs. This option is controlled by the following local assembly symbol:

Location	Result	Operand
%%VERFLD	SET	*value*

value A value that can be one of the following:

Value	Description
Undefined	Disable comparison
0	Disable comparison
1	Issues a comment to the listing dataset when a descrepancy is detected
2	Issues a warning error for each descrepancy detected
3	Issues a fatal error for each descrepancy detected

FIELD RETRIEVAL

The set of run-time field retrieval opdefs retrieve (fetch) the current value of a given field into a result field (an address or scalar register). The fetched value is right-justified within the result register. Run-time field retrieval opdefs (GETF, GET, and SGET) use all of the memory optimization catagories. The catagories are designated using the following optimization formats:

- Fast format uses a minimum number of instructions and registers (omits register S*j*) to perform field retrieval. The GETF odefs use this format.
- Full format performs a required memory read of the base image register (S*j*) prior to field retrieval within the opdef. The GET opdefs use this format.
- Quick format assumes the base image register (S*j*) has the proper memory image in which the field resides. The SGET opdefs use this format.

GETF - Field retrieval (fast format)

The GETF (fast format) opdefs retrieve the value of a given field from the specified table entry. GETF opdefs expand and generate instructions depending on the registers specified and/or the location and size of the desired field within the table entry word. The first method uses scratch register S*k* as an extraction mask, while the second method extracts the field using left- and right-shift instructions.

Location	Result	Operand
	GETF,S*i*	S*k*,*field*,A*n*
	GETF,A*i*	S*k*,*field*,A*n*

S*i* or A*i* Result register. A register in the range from S0 to S7 or A0 to A7 receiving the desired value (right-justified within the register).

S*k* Scratch register. Register S*k* is used in shifts or masks to extract the desired field.

A*n* Entry address register. Register A*n* contains the base address of the desired entry within a table.

field Field entry. *field* names the field associated with the operation. This value must be previously defined in a table structure.

The GETF opdef includes the following special syntaxes:

- Using register A0 in the A*n* position takes advantage of special CAL syntax during the read of memory. A0 reads, biased by the word offset, from the programs current Exchange Package data base address (XPDBA)† for CRAY X-MP Computer Systems.
- If register S*i* is register S*k*, a partial-word field is extracted using shift instructions.
- If register S*i* is register S0, a partial-word field is extracted using shift instructions.
- If register S*k* is register S0, a partial-word field is extracted using shift instructions.

† The Exchange Package data base address is designated as XPBA for CRAY-1 Models A, B, S, and M Computer Systems.

GET - Field retrieval (full format)

The GET (full format) opdefs retrieve a field value from memory and keep the entry word read in the base image register (S*j*). The word image of the retrieved field resides in a scalar register and is available to other field management opdefs. The result register can be an address or a scalar register. The field value placed in the result register is right-justified within the register.

Location	Result	Operand
	GET,S*i*	S*j*&S*k*,*field*,A*n*
	GET,A*i*	S*j*&S*k*,*field*,A*n*

S*i* or A*i* Result register. A register in the range from S0 to S7 or A0 to A7 receiving the desired value (right-justified within the register).

S*j* Base image register. Register S*j* maintains an image of the word in which the desire field resides.

S*k* Scratch register. Register S*k* shifts or masks to extract the desired field.

A*n* Entry address register. Register A*n* contains the base address of the desired entry within a table.

field Field entry. *field* names the field associated with the operation. This value must be previously defined in a table structure.

The GET opdef includes the following special syntaxes:

- The use of A0 in the A*n* position takes advantage of the special CAL syntax during the read of memory. It reads, biased by the word offset, from the programs current XPDBA† for CRAY X-MP Computer Systems.
- S0 can be used in the S*i* position when the desired field is right-justified within the table entry word, and the length of the desired field is greater than or equal to the size of the result register (64 bits for S*i* and 24 bits for A*i*).

† The Exchange Package data base address is designated as XPBA for CRAY-1 Models A, B, S, and M Computer Systems.

- S0 can be used in the S*k* position when the desired field is right-justified within the table entry word, and the length of the desired field is greater than or equal to the size of the result register (64 bits for S*i* and 24 bits for A*i*).

- S*j* and S*k* can be the same register when the desired field is right-justified within the table entry word, and the length of the desired field is greater than or equal to the size of the result register (64 bits for S*i* and 24 bits for A*i*).

SGET - Field retrieval (quick format)

The SGET (quick format) opdefs retrieve a field value from the word image that resides in the base image register (S*j*). SGET, unlike GET, does not perform a memory read while extracting the field value. The result register can be an address or a scalar register. The field value is right-justified within the result register.

Location	Result	Operand
	SGET,S*i*	S*j*&S*k*,*field*
	SGET,A*i*	S*j*&S*k*,*field*

S*i* or A*i* Result register. A register in the range from S0 to S7 or A0 to A7 receiving the desired value (right-justified within the register).

S*j* Base image register. Register S*j* maintains an image of the word in which the desire field resides.

S*k* Scratch register. Register S*k* shifts or masks to extract the desired field. This register is assumed to have no special value upon entry to or exit from the opdef.

field Field entry. *field* names the field associated with the operation. This value must be previously defined in a table structure.

The SGET opdef includes the following special syntaxes:

- S0 can be used in the S*j* position when the desired field is right-justified within the table entry word, and the length of the desired field is greater than or equal to the size of the result register (64 bits for S*i* and 24 bits for A*i*).

- S0 can be used in the S*k* position when the desired field is right-justified within the table entry word, and the length of the desired field is greater than or equal to the size of the result register (64 bits for S*i* and 24 bits for A*i*).

- *Sj* and *Sk* can be the same register when the desired field is right-justified within the table entry word, and the length of the desired field is greater than or equal to the size of the result register (64 bits for *Si* and 24 bits for A*i*).

FIELD MODIFICATION

The set of run-time field modification opdefs update the current value of a specified field. The current value can be expressed as an assembly symbol or constant or can reside in an address or scalar register. The field modification opdefs (PUT, SPUT, and SET) use the following optimization catagories.

- Full format updates a partial word opdef after a read of memory is made of the base image register (*Sj*). The PUT opefs use this format.

- Quick format assumes the base image register (*Sj*) contains the proper memory image in which the field resides. The SPUT and SET opdefs use this format.

PUT - Field update (full format)

The PUT (full format) opdef updates a field's value in memory and keeps the entry word in the base image register (*Sj*). Register *Sj* is then available to the quick field management opdefs. The source of the value can be an assembly symbol or constant or can reside in either an address or a scalar register. If the new value resides in a register, the value must be right-justified within that register upon entry to the opdef.

Location	Result	Operand
	PUT,S*i*	S*j*&S*k*,*field*,A*n*
	PUT,A*i*	S*j*&S*k*,*field*,A*n*
	PUT,*val*	S*j*&S*k*,*field*,A*n*

Si, A*i*, or *val*

Source of the new value; can be an address register, a scalar register, or an assembly symbol or constant. If the register contains a value, the value must be right-justified within that register.

Use of registers A0 or S0 is invalid and results in assembly errors.

Sj Base image register. Register *Sj* maintains an image of the word in which the desire field resides.

Sk Scratch register. Register *Sk* shifts or masks to set the new value for the field.

An Entry address register. Register *An* contains the base address of the desired entry within a table.

field Field entry. *field* names the field associated with the operation. This value must be previously defined in a table structure.

The PUT opdef includes the following special syntaxes:

- The use of A0 in the *An* position takes advantage of the special CAL syntax during the read of memory. It reads, biased by the word offset, from the programs current XPDBA.
- S0 can be used in the *Sj* position when the length of the referenced field is 1-word long (64 bits).
- S0 can be used in the *Sk* position when the length of the referenced field is 1-word long (64 bits).
- *Sj* and *Sk* can be the same register when the length of the referenced field is 1-word long (64 bits).
- Use minus one (-1) in a result field to set all bits in the referenced field.
- Use zero (0) in the result field to clear all bits in the referenced field.

SPUT - Field update (quick format)

The SPUT (quick field) opdefs update a field's value in memory and in the base image register (*Sj*). Upon entry, register *Sj* contains the memory image in which the field resides. SPUT differs from PUT, in that, SPUT does not read memory before changing values. The source of the value can be an assembly symbol or constant or can reside in either an address or a scalar register. If the new value resides in a register, the value must be right-justified upon entry to the opdef.

Location	Result	Operand
	SPUT,S*i*	S*j*&S*k*,*field*,A*n*
	SPUT,A*i*	S*j*&S*k*,*field*,A*n*
	SPUT,*val*	S*j*&S*k*,*field*,A*n*

Si, A*i*, or *val*
: Source of the new value. This is an address register, a scalar register, or an assembly symbol or constant. If the register contains a value, the value must be right-justified within that register.

  Use of registers A0 or S0 is invalid and results in assembly errors.

S*j*
: Base image register. Register S*j* maintains an image of the word in which the desire field resides.

S*k*
: Scratch register. Register S*k* shifts or masks to set the new value for the field.

A*n*
: Entry address register. Register A*n* contains the base address of the desired entry within a table.

field
: Field entry. *field* names the field associated with the operation. This value must be previously defined in a table structure.

The SPUT opdef includes the following special syntaxes:

- The use of A0 in the A*n* position takes advantage of the special CAL syntax during the read of memory. It reads, biased by the word offset, from the programs current XPDBA† for CRAY X-MP Computer Systems.
- S0 can be used in the S*j* position when the length of the referenced field is 1-word long (64 bits).
- S0 can be used in the S*k* position when the length of the referenced field is 1-word long (64 bits).
- S*j* and S*k* can be the same register when the length of the referenced field is 1-word long (64 bits).
- Use minus one (-1) in a result field to set all bits in the referenced field.
- Use zero (0) in the result field to clear all bits in the referenced field.

† The Exchange Package data base address is designated as XPBA for CRAY-1 Models A, B, S, and M Computer Systems.

SET - Field update (quick format)

The SET (quick format) opdefs update a field's value only in the base image register (S*j*). Upon entry, register S*j* contains the memory image in which the field resides. SET differs from SPUT, in that, SET does not rewrite memory following a change to the value in the base image register.

The source of the value can be an assembly symbol or constant or can reside in either an address or a scalar register. If the new value resides in a register, the value must be right-justified within that register upon entry to the opdef.

Location	Result	Operand
	SET,S*i*	S*j*&S*k*,*field*
	SET,A*i*	S*j*&S*k*,*field*
	SET,*val*	S*j*&S*k*,*field*

S*i*, A*i*, or *val*
: Source of the new value. This is an address register, a scalar register, or an assembly symbol or constant. If the register contains a value, the value must be right-justified within that register.

  Use of registers A0 or S0 is invalid and results in assembly errors.

S*j*
: Base image register. Register S*j* maintains an image of the word in which the desire field resides.

S*k*
: Scratch register. Register S*k* shifts or masks to set the new value for the field.

field
: Field entry. *field* names the field associated with the operation. This value must be previously defined in a table structure.

The SET opdef includes the following special syntaxes:

- S0 can be used in the S*j* position when the length of the referenced field is 1-word long (64 bits).
- S0 can be used in the S*k* position when the length of the referenced field is 1-word long (64 bits).
- S*j* and S*k* can be the same register when the length of the referenced field is 1-word long (64 bits).

- Use minus one (-1) in a result field to set all bits in the referenced field.
- Use zero (0) in the result field to clear all bits in the referenced field.

MISCELLANEOUS RUN-TIME FIELD OPDEFS

The miscellaneous run-time field opdefs described in this section are helper opdefs for programs that make extensive use of run-time field management opdefs. These opdefs complement the previously defined set of quick opdefs and aid the programmer by checking the status of various field assumptions that can exist in program logic. Miscellaneous run-time field opdefs include the following:

- LOAD
- STORE
- ASSIGN
- LJF
- XFER

LOAD - PRELOAD ENTRY WORD (FULL FORMAT)

The LOAD (full format) opdef preloads the entry word into a specified base image register (S*j*). Upon exit from the LOAD opdef, the entry word loaded into register S*j* is available for use by any of the following quick opdefs: SGET, SET, SPUT, or LJF.

Location	Result	Operand
	LOAD,S*j*	*field*,A*n*

S*j* Base image register. Register S*j* maintains an image of the word in which the desire field resides.

field Field entry. *field* names the field associated with the operation. This value must be previously defined in a table structure.

A*n* Entry address register. Register A*n* contains the base address of the desired entry within a table.

The LOAD opdef includes the following special syntax:

- The use of A0 in the A*n* position takes advantage of the special CAL syntax during the read of memory. It reads, biased by the word offset, from the programs current XPDBA†.

STORE - UPDATE ENTRY WORD (FULL FORMAT)

The STORE (full format) opdef updates an entry word in memory with the contents of a specified base image register (S*j*). Any previous word offsets assigned to the base image register remain intact upon exit from the opdef.

Location	Result	Operand
	STORE,S*j*	*field*,A*n*

S*j* Base image register. Register S*j* maintains an image of the word in which the desire field resides.

A*n* Entry address register. Register A*n* contains the base address of the desired entry within a table.

field Field entry. *field* names the field associated with the operation. This value must be previously defined in a table structure.

The STORE opdef includes the following special syntax:

- The use of A0 in the A*n* position takes advantage of the special CAL syntax during the read of memory. It reads, biased by the word offset, from the programs current XPDBA.

ASSIGN - FIELD OFFSET CHANGE (FULL FORMAT)

The ASSIGN (full format) opdef changes the word offset assigned to the base image register (S*j*). The ASSIGN opdef does not generate any executable instructions and is only evaluated and processed during program assembly.

† The Exchange Package data base address is designated as XPBA for CRAY-1 Models A, B, S, and M Computer Systems.

Location	Result	Operand
	ASSIGN,S*j*	*field*

S*j* Base image register. Register S*j* maintains an image of the word in which the desire field resides.

field Field entry. *field* names the field associated with the operation. This value must be previously defined in a table structure.

The ASSIGN opdef includes no special syntaxes.

LJF - LEFT SHIFT FIELD (QUICK FORMAT)

The LJF (quick format) opdef left shifts the contents of a base image register into the specified result register. The number of bits LJF shifts is governed by the starting number of the referenced field (S@field). Upon entry, the base image register (S*j*) contains the word image in which the field resides. LJF is commonly used in the testing of single-bit fields.

Location	Result	Operand
	LJF,S*i*	S*j*<*field*

S*i* Result register. A register in the range from S0 to S7.

S*j* Base image register. Register S*j* maintains an image of the word the desired field resides in.

field Field entry. *field* names the field associated with the operation. This value must be previously defined in a table structure.

The LJF opdef includes the following special syntax:

- IF S*i* and S*j* are the same register and the field is left-justified with the entry word, no executable instructions are generated.

XFER - COPY FIELD (FAST FORMAT)

The XFER (fast format) opdef copies the field value from one entry to another.

Location	Result	Operand
	XFER,*sfld*,A*n*	S*i*&S*j*&S*k*,*dfld*,A*m*

sfld Source field. *sfld* must be previously defined in a table structure.

A*n* Source entry address register. Register A*n* contains the base address of the entry within a table.

S*i* Scratch register

S*j* Scratch register

S*k* Scratch register

dfld Destination field. *dfld* must be previously defined in a table structure.

A*m* Destination entry address register. Register A*m* contains the base address of the entry within a table.

XFER schedules memory reads and writes in the following ways:

- S*i* can be S0 if the source and destination fields are both 1-word long (64 bits).
- S*j* can be S0 if the source and destination fields are both 1-word long (64 bits).
- S*k* can be S0 if the source and destination fields are both 1-word long (64 bits).
- A*n* can be A0 if the source and destination fields are both 1-word long (64 bits).
- A*m* can be A0 is the source and destination fields are both 1-word long (64 bits).

COMPLEX TABLE MANIPULATION 2

Complex tables allow fields to span Central Memory word boundaries. Normal tables, which require each field to be wholly contained in a single memory word, are described in part 2, section 1. Since complex table manipulation is an extension of normal table manipulation, this section assumes you are familiar with the normal table manipulation macros and opdefs (see part 2, section 1).

Complex table macros and opdefs are identified by mnemonics with an initial C, as in CTABLE. The complex table macros define a table's attributes so that other macros and opdefs may refer to the table symbolically. The complex run-time management opdefs allow you to retrieve data from any field defined in the complex table and also store data any place in the table.

TABLE DEFINITION

Both complex and normal tables consist of three basic categories: a header, one or more entries, and overall characteristics. However, the normal table macros and the complex table macros, for definition and table structure maintenance, must be used separately. Mixing complex and normal table macros results in assembly errors.

You can define fields in tables that are capable of spanning 64-bit word boundaries. You define tables of this type by using the following macros:

- CTABLE Defines the overall table attributes.
- CENDTAB Designates the end of a complex table structure.
- CFIELD Defines a field within the current complex table structure.
- CSBFIELD Identifies fields contained within a larger field.
- 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.

CTABLE - DEFINE 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. When you omit any attribute from the parameter list, its corresponding symbol remains undefined. CTABLE 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 1- to 5-character name of the table being described. This parameter is required if any of the other parameters are specified.

LH=*lh* Length of the table header in 64-bit words. This is an optional parameter.

LE=*le* Length of table entry in 64-bit words. This is an optional parameter.

NE=*ne* Number of table entries. This is an optional parameter.

L=*lt* Table length in 64-bit words. This is an optional parameter.

SZ=*st* Table length in 64-bit words. This is an optional parameter.

Example:

Location	Result	Operand	Comment
1	10	20	35
VOL1	CTABLE	LH=0,LE=D'10	

Expansion:

```
LH@VOL1=0
LE@VOL1=D'10
```

CENDTAB - END TABLE DEFINITION

The CENDTAB macro terminates the definition of the current complex table and automatically assigns a value to table entry length (LE@*name*) if you have not defined it. 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. This parameter is required.

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 - NAME FIELD WITHIN 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. For successful definition of a complex field:

- The definition must be contained within a complex table definition (within a CTABLE CENDTAB sequence).
- All definitions must be in numerically ascending order with respect to word and bit positions.
- The field length must be between 1 and 64 bits, inclusive.

Depending on the operands entered, CFIELD generates symbols identifying the initial word (X@*name*), the initial bit (T@*name*), and the length of the field (O@*name*).

Format:

Location	Result	Operand
name	CFIELD	*word,sbit,length*

name A 1- to 6-character name of the field being described. If you omit *name*, the definition of the special labels is skipped. This parameter is required.

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. This parameter is required. 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. This parameter is required.

length Field length in number of bits. If you omit *length*, a default length of 1 bit is assumed. If you give *length* a numeric value, *length* must be between 1 and 64, inclusive. This parameter is required.

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
```

CSBFIELD - NAME PART OF A FIELD

Use the CSBFIELD macro to define a field (defined by CFIELD) as smaller, separate fields. As with CFIELD, fields which you define with CSBFIELD can span 64-bit word boundaries. For successful field definition:

- 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. This parameter is required.

sbit Number identifying the leftmost bit of the field. This parameter is required.

length Number identifying the field length in bits. This parameter is optional; the default is 1.

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
```

CNXTWORD - ADVANCE SPECIFIED NUMBER OF 64-BIT WORDS

The CNXTWORD macro allocates an area in a table, leaving out definitions. Use CNXTWORD primarily when the table to be constructed contains other previously defined tables as elements. An example is 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: *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. This parameter is optional.

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. This parameter is optional; if you omit *length*, 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 SPECIFIED NUMBER OF 64-BIT WORDS

The CREDEF macro allows a block of 64-bit words to be redefined within a table. Use CREDEF primary 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. This parameter is optional; when you omit *word*, 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. This parameter is optional; if you omit *length*, 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	

RUN-TIME TABLE MANAGEMENT

The run-time table management opdefs transfer data between a holding register and a field in the complex table. 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.

The complex run-time table management opdefs are:

- CGET Fetches the contents of a field into a register
- CPUT Stores the contents of a register into a field

The short forms of these macros are not provided due to hardware restrictions and the need to save on register usage.

An optional location field can be specified for the complex run-time field management opdefs (CGET and CPUT). All of the other parameters are required.

NOTE

These macros cannot be used in reentrant code.

CGET - RETRIEVE FIELD CONTENTS

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 DATA IN A FIELD

The CPUT opdef conveniently stores a field value into a table or table entry. You must have defined the field previously using the CFIELD or CSBFIELD macros. 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.

S*k* S register to be used as a scratch register. The contents of this register, upon exit, have no meaningful value. S*k*, S*i*, and S*j* must be unique registers. S0 is not a valid register.

A*l* A register containing the base address of the table or table entry. This register is not destroyed by CPUT.

INDEXED TABLE CONSTRUCTION MACROS 3

Use the indexed table construction macros (MAP, MAPTO, and ENDMAP) to construct indexed arrays of numbers, addresses, words, or data records that have a common format. Indexed table construction macros can be used as:

- Jump vectors
- Indexed records

Indexed table construction macros build any table in which an entry can be looked up by an index key. The MAP macro sets up assembly parameters to control the table. The ENDMAP macro completes the table and makes final error checks. Each invocation of the MAPTO macro generates one table entry. The variable macro or pseudo-op you specify in the MAP macro call generates each entry.

Since the generating pseudo-op is a variable parameter, MAP, ENDMAP and MAPTO are very powerful. You can use these macros to create a table of any desired structure, as long as you order the entries with a set of sequentially-numbered constant key values.

Format:

Location	Result	Operand
tname	MAP	*op*,FILL=*exp*,ORIGIN=*exp*
*key*1	MAPTO	*value*1
*key*2	MAPTO	*value*2
	.	
	.	
	.	
key_n	MAPTO	$value_n$
	ENDMAP	*limit*

tname Table name; location field symbol for first entry.

op Macro or opdef name to use in generating each entry

FILL=*exp* Value to place in unused entries. FILL=NO causes MAPTO and ENDMAP to generate an assembly error if there are any omitted entries. If FILL=*exp* is omitted, MAPTO uses a default of 0.

ORIGIN=*exp*
Value for the first key in the table. Entering a nonzero value for *exp* suppresses FILL entries at the beginning of the table. If you specify a nonzero origin, subtract this origin value from the index value before referencing the table.

key_n Entry ordinal symbol. The first *key* is 0.

$value_n$ Value to assemble into the entry

limit Highest legal key value +1

JUMP VECTORS

Jump vectors are the equivalent of the CFT computed GOTO. If you want to jump to one of a number of addresses based on the value in a variable, use MAP, MAPTO, and ENDMAP to build a table of addresses or jump instructions that are indexed by the controlling variable.

Examples:

Location	Result	Operand	Comment
GOTO	=	*	A1=index value
	A7	P.JUMPVEC	Table base address
	A7	A7+A1	Add 2 times the index
	A7	A7+A1	
	B00	A7	Entry parcel address
	J	B00	To jump instruction

Location	Result	Operand
JUMPVEC	MAP	J,FILL=UNDEF
VAL1	MAPTO	LOC1
VAL2	MAPTO	LOC2
VAL3	MAPTO	LOC3
VAL5	MAPTO	LOC5
	ENDMAP	D'10

The example macros construct a series of 32-bit jump instructions. For a value of VAL1, the routine at GOTO jumps to location LOC1, for VAL2 to LOC2 and so on. For values other than VAL1, VAL2, VAL3, or VAL5, the routine jumps to location UNDEF.

INDEXED RECORDS

To build a list of messages that associates one message with each possible value of a variable, use MAP, MAPTO, and ENDMAP as shown in the following example.

Example:

```
| Location | Result  | Operand                                  |
|==========|=========|==========================================|
|          |         |                                          |
| MSGTAB   | MAP     | DATA,ORIGIN=ERCODE1                      |
| ERCODE1  | MAPTO   | (='ERROR 1 - INVALID PARAMETER'*L)       |
| ERCODE2  | MAPTO   | (='ERROR 2 - ILLEGAL OPERATION'*L)       |
| ERCODE3  | MAPTO   | (='ERROR 3 - SYSTEM ERROR'*L)            |
|          | ENDMAP  | ERCODE3+1                                |
```

Each table entry is a word (DATA) containing the address of the corresponding message for that index value.

COS-DEPENDENT MACROS AND OPDEFS 4

The COS-dependent macros and opdefs described in this section are system aids processed by the assembler using macro definitions defined in the system text, COSTXT.

These specific macros and opdefs are intended for internal system users only. Users other than system programmers should avoid using them.

There are three categories of COS-dependent macros and opdefs:†

- System task opdefs
- Message processor macro
- COS internal subroutine linkage macro

SYSTEM TASK OPDEFS

The system task opdefs include:

- ERDEF Generates error processing entries in the Exchange Processor
- GETDA Obtains first Dataset Allocation Table (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.

† Overlay task manager macros were removed with the 1.14 COS release.

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 Job Table Area (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 the 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, omit the parameter.

REPR=NO Error is not reprievable; otherwise, omit the parameter.

GETDA - OBTAIN FIRST DAT PAGE ADDRESS

The GETDA opdef obtains the STP-relative address of the first Dataset Allocation Table (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*

Ai A register to receive the STP-relative address of the DAT; cannot be A0. Ai is 0 if no DATs are associated with the DNT.

Aj 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,Ai	Aj,Ak

Ai A register to receive the STP-relative address of the DAT; cannot be A0. Ai is 0 if no DATs are associated with the DNT.

Aj — A register containing the STP-relative address of the DNT; cannot be A0.

Ak — 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 Dataset Allocation Table (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,Ai,Aj	Sk&Sl,Am,An

Ai — A register to receive the STP-relative address of the next DAT page; cannot be A0. Ai is 0 if there are no further DAT pages.

Aj — A register to receive the STP-relative address of the JTA if the DAT is in the JTA; cannot be A0.

Sk — S register to be used by GETNDA; cannot be S0.

Sl — 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,Ai	Aj,Ak

Ai — A register to receive the STP-relative address of the next DAT page; cannot be A0. Ai will be 0 if there are no further DAT pages.

Aj — A register containing the STP-relative address of the current DAT page; cannot be A0.

Ak — 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       =           *
```

MESSAGE PROCESSOR MACRO - LOGMSGM

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, 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, 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 - $SUB

The Cray Operating System (COS) internal subroutine linkage ($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 must be specified as the exit point for the subroutine.

To call the subroutine, enter the following:

```
A7        stackaddr                    Entry condition
R         TAG          call subroutine Exit conditions
```

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.

Example:

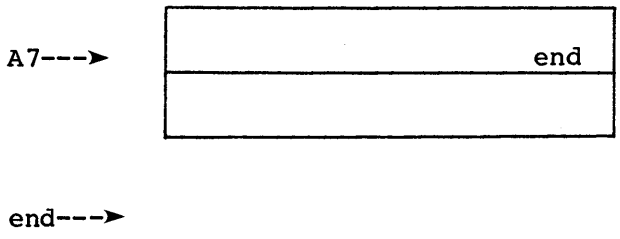
Location	Result	Operand	Comment
TAG	$SUB	SREG=*a*:*b*.AREG=*c*:*d*	
.			
	J	$RETURN	*exit coditions*

$SUB defines symbol $RETURN as the exit point for the subroutine; all exits must be made using jumps to $RETURN.

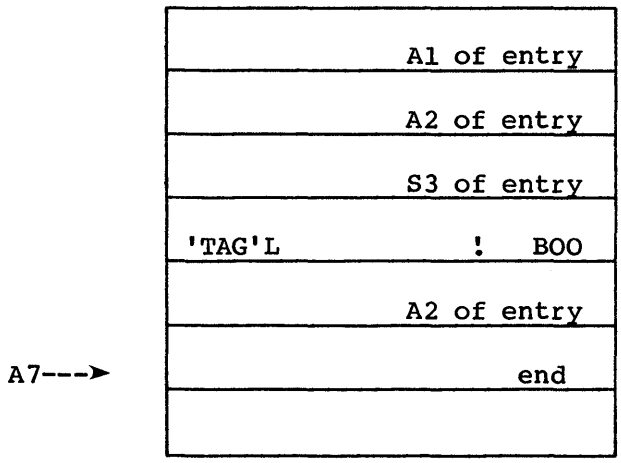
The following restrictions are in effect when using the $SUB macro:

- A, S6, and S7 must not be used as entry registers for $SUB routines.
- S6 and S7 msut not be used as exit registers for $SUB subroutines.
- BOO is save and restored within the macro; subroutine call can be made freely within $SUB subroutines, with the provision that called routines do not alter A7.
- Always exit $SUB subroutines by jumping to $RETURN; never jump to BOO when exiting $SUB.

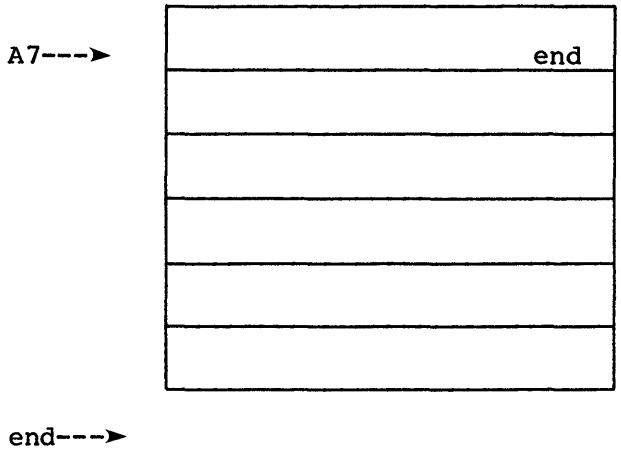
Stack format before $SUB call:



Stack format after 'TAG $SUB AREG=1:2,SREG=3'



Stack format after exit from TAG subroutine:



APPENDIX SECTION

OUTMODED FEATURES A

The following features are not supported following the release of the 1.14 version of the Cray Operating System (COS):

- BREG - Assign names to B registers
- TREG - Assign names to T registers

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 Required. 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 Required. 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

TABLE MACRO EXPANSIONS B

Macro expansions for the following field retrieval and modification opdefs are included in this section:

- GET,S*i*
- GET,A*i*
- GETF,S*i*
- GETF,A*i*
- PUT,S*i*
- PUT,A*i*
- PUT,*val*

GET,S*i* EXPANSIONS

Expansions for the GET,S*i* opdef take the following into consideration:

- Location of the field within the table word
- Length of the field with respect to the result register size

The GET,S*i* opdef has four special cases:

- The field size is equal to the size of a Cray word (example 1).
- The field size is less than a Cray word, and the field is left-justified within the table word (example 2).
- The field size is less than a Cray word, and the field is right-justified within the table word (example 3).
- The field size is less than a Cray word, and the field is neither right- nor left-justified within the the table word (example 4).

Example 1:

Location	Result	Operand
	Si	W@*field*,An
	Sj	W@*field*,An

Example 2:

Location	Result	Operand
	Sj	W@*field*,An
	Sk	>N@*field*
	Si	*Sj*&*Sk*
	Si	*Si*>D'64-S@*field*-N@*field*

Example 3:

Location	Result	Operand
	Sj	W@*field*,An
	Sk	<N@*field*
	Si	*Sj*&*Sk*

Example 4:

Location	Result	Operand
	Sj	W@*field*,An
	Sk	>N@*field*
	Sk	*Sk*>S@*field*
	Si	*Sj*&*Sk*
	Si	*Si*>D'64-S@*field*-N@*field*

GET,A*i* EXPANSIONS

Expansions for the GET,A*i* opdef take the following into consideration:

- Location of the field within the table word
- Length of the field with respect to the result register size

The GET,A*i* opdef has four special cases:

- The field size is greater than or equal to the result register size, and the field is right-justified within the table word (example 5).
- The field size is greater than or equal to the result register size, and the field is not right-justified within the table word (example 6).
- The field size is less than the size of the result register, and the field is left-justified within the table word (example 7).
- The field size is less than the size of the result register (example 8).

Example 5:

Location	Result	Operand
	A*i*	W@*field*,A*n*
	S*j*	W@*field*,A*n*

Example 6:

Location	Result	Operand
	S*k*	W@*field*,A*n*
	S*j*	W@*field*,A*n*
	S*k*	Sk>D'64-N@*field*-S@*field*
	A*i*	S*k*

Example 7:

Location	Result	Operand
	S*k*	W@*field*,A*n*
	S*j*	W@*field*,A*n*
	S*k*	Sk>D'64-N@*field*
	A*i*	S*k*

Example 8:

Location	Result	Operand
	Sk	W@*field*,A*n*
	Sj	W@*field*,A*n*
	Sk	*Sk*<S@*field*
	Sk	*Sk*>D'64-N@*field*
	A*i*	*Sk*

GETF,S*i* EXPANSIONS

Expansions for the GETF,S*i* opdef take the following into consideration:

- Location of the field within the table word
- Length of the field with respect to the result register size
- Specified registers

The GETF,S*i* opdef has four special cases:

- The field size is equal to the size of a Cray word (example 9).
- The field is right-justified within the table word; registers S*i* and S*k* must not be the same and cannot be register S0 (example 10).
- The field is left-justified within the table word (example 11).
- Miscellaneous cases (example 12).

Example 9:

Location	Result	Operand
	S*i*	W@*field*,A*n*

Example 10:

Location	Result	Operand
	Si	W@*field*,A*n*
	Sk	<N@*field*
	Si	S*j*&S*k*

Example 11:

Location	Result	Operand
	Si	W@*field*,A*n*
	Si	Si>D'64-N@*field*

Example 12:

Location	Result	Operand
	Si	W@*field*,A*n*
	Si	Si<S@*field*
	Si	Si>D'64-N@*field*

GETF,A*i* EXPANSIONS

Expansions for the GETF,A*i* opdef take the following into consideration:

- Location of the field within the table word
- Length of the field with respect to the result register size

The GETF,A*i* opdef has four special cases:

- The field size is greater than or equal to the result register sized, and the field is right-justified within the table word (example 13).
- The field size is greater than or equal to the result register size, and the field is not right-justified within the table word (example 14).

- The field size is less than the size of the result register, and the field is left-justified within the table word (example 15).
- The field size is less than the size of the result register (example 16).

Example 13:

Location	Result	Operand
	Ai	W@*field*,A*n*

Example 14:

Location	Result	Operand
	Sk	W@*field*,A*n*
	Sk	*Sk*>W@*field*-S@*field*
	Ai	*Sk*

Example 15:

Location	Result	Operand
	Sk	W@*field*,A*n*
	Sk	*Sk*>D'64-N@*field*
	Ai	*Sk*

Example 16:

Location	Result	Operand
	Sk	W@*field*,A*n*
	Sk	*Sk*<S@*field*
	Sk	*Sk*>D'64-N@*field*
	Ai	*Sk*

PUT,Si EXPANSIONS

Expansions for the PUT,S*i* opdef take the following into consideration:

- Location of the field within the table word
- Length of the field with respect to the source register size

The PUT,S*i* opdef has five special cases:

- The field size is equal to 1 word, and the source register is different from the base image register (example 17).
- The field size is equal to 1 word, and the source register is the same as the base image register (example 18).
- The field size is less than a Cray word, and the field is left-justified within the table word (example 19).
- The field size is less than a Cray word, and the field is neither right- nor left-justified within the table word (example 20).
- The field size is less than a Cray word, and the field is right-justified within the table word (example 21).

Example 17:

Location	Result	Operand
	S*j*	S*i*
	W@*field*,A*n*	S*i*

Example 18:.

Location	Result	Operand
	W@*field*,A*n*	S*i*

Example 19:

Location	Result	Operand
	Sj	W@*field*,A*n*
	Sk	>N@*field*
	Si	*Si*<D'64-S@*field*-N@*field*
	Sj	*Si*!*Sj*&*Sk*
	Si	*Si*>D'64-S@*field*-N@*field*
	W@*field*,A*n*	*Sj*

Example 20:

Location	Result	Operand
	Sj	W@*field*,A*n*
	Sk	>N@*field*
	Sk	*Sk*>S@*field*
	Si	*Si*<D'64-S@*field*-N@*field*
	Sj	*Si*!*Sj*&*Sk*
	Si	*Si*>D'64-S@*field*-N@*field*
	W@*field*,A*n*	*Sj*

Example 21:

Location	Result	Operand
	Sj	W@*field*,A*n*
	Sk	<N@*field*
	Sj	*Si*!*Sj*&*Sk*
	W@*field*,A*n*	*Sj*

PUT,A*i* EXPANSIONS

Expansions for the PUT,A*i* opdef take the following into consideration:

- Location of the field within the table word
- Length of the field with respect to the source register size

The PUT,A*i* opdef has seven special cases:

- The field size is equal to 1 word (example 22).

- The field size is greater than or equal to the source register size, and the field is right-justified within the table word (example 23).
- The field size is greater than or equal to the source register size, and the field is left-justified within the table word (example 24).
- The field size is greater than or equal to the source register size (example 25).
- The field size is less than the size of the result register, and the field is right-justified within the table word (example 26).
- The field size is less than the size of the result register, and the field is left-justified within the table word (example 27).
- The field size is less than the size of the result register (example 28).

Example 22:

Location	Result	Operand
	S*j*	A*i*
	W@*field*,A*n*	A*i*

Example 23:

Location	Result	Operand
	S*j*	W@*field*,A*n*
	S*k*	<N@*field*
	S*j*	#S*k*&S*j*
	S*k*	A*i*
	S*j*	S*j*!S*k*
	W@*field*,A*n*	S*j*

Example 24:

Location	Result	Operand
	Sj	W@field,An
	Sk	>N@field
	Sj	#Sk&Sj
	Sk	Ai
	Sk	Sk<D'64-S@field-N@field
	Sj	Sk!Sj
	W@field,An	Sj

Example 25:

Location	Result	Operand
	Sj	W@field,An
	Sk	>N@field
	Sk	Sk>S@field
	Sj	#Sk&Sj
	Sk	Ai
	Sk	Sk<D'64-S@field-N@field
	Sj	Sk!Sj
	W@field,An	Sj

Example 26:

Location	Result	Operand
	Sj	W@field,An
	Sk	<N@field
	Sj	#Sk&Sj
	Sk	Ai
	Sk	Sk<D'64-N@field
	Sk	Sk>S@field
	Sj	Sk!Sj
	W@field,An	Sj

Example 27:

Location	Result	Operand
	Sj	W@*field*,A*n*
	Sk	>N@*field*
	Sj	#*Sk*&*Sj*
	Sk	A*i*
	Sk	*Sk*<D'64-N@*field*
	Sj	*Sk*!*Sj*
	W@*field*,A*n*	*Sj*

Example 28:

Location	Result	Operand
	Sj	W@*field*,A*n*
	Sk	>N@*field*
	Sk	Sk>S@*field*
	Sj	#*Sk*&*Sj*
	Sk	A*i*
	Sk	*Sk*<D'64-N@*field*
	Sk	*Sk*>S@*field*
	Sj	*Sk*!*Sj*
	W@*field*,A*n*	*Sj*

PUT,*val* EXPANSIONS

Expansions for the PUT,*val* opdef take the following into consideration:

- Location of the field within the table word
- Value in the field

This special case opdef has values of 0 and -1. The expansions for this opdef are as follows:

- The field size is equal to 1 word (example 29).
- The value specified is -1, and the field is left-justified within the table word (example 30).
- The value specified is -1, and the field is right-justified within the table word (example 31).

- The value specified is -1, and the field is neither right- nor left-justified within the table word (example 32).
- The value specified is 0, and the field is left-justified within the table word (example 33).
- The value specified is 0, and the field is right-justified within the table word (example 34).
- The value specified is 0, and the field is neither right- nor left-justified within the table word (example 35).
- The value specified is not a special case, and the field is left-justified within the table word (example 36).
- The value specified is not a special case, and the field is right-justified within the table word (example 37).
- The value specified is not a special case, and the field is neither right- nor left-justified within the table word (example 38).

Example 29:

Location	Result	Operand
	S*j*	*val*
	W@*field*,A*n*	S*j*

Example 30:

Location	Result	Operand
	S*j*	W@*field*,A*n*
	S*k*	>N@*field*
	S*j*	S*j*!S*k*
	W@*field*,A*n*	S*j*

Example 31:

Location	Result	Operand
	S*j*	W@*field*,An
	S*k*	<N@*field*
	S*j*	S*j*!S*k*
	W@*field*,A*n*	S*j*

Example 32:

Location	Result	Operand
	Sj	W@*field*,An
	Sk	>N@*field*
	Sk	Sk>S@*field*
	Sj	Sj!Sk
	W@*field*,An	Sj

Example 33:

Location	Result	Operand
	Sj	W@*field*,An
	Sk	>N@*field*
	Sj	#Sk&Sj
	W@*field*,An	Sj

Example 34:

Location	Result	Operand
	Sj	W@*field*,An
	Sk	<N@*field*
	Sj	#Sk&Sj
	W@*field*,An	Sj

Example 35:

Location	Result	Operand
	Sj	W@*field*,An
	Sk	>N@*field*
	Sk	Sk>S@*field*
	Sj	#Sk&Sj
	W@*field*,An	Sj

Example 36:

Location	Result	Operand
	Sj	W@*field*,*An*
	Sk	>N@*field*
	Sj	#*Sk*&*Sj*
	Sk	*val*
	Sk	Sk<D'64-S@*field*-N@*field*
	Sj	*Sj*!*Sk*
	W@*field*,*An*	*Sj*

Example 37:

Location	Result	Operand
	Sj	W@*field*,*An*
	Sk	<N@*field*
	Sj	#*Sk*&*Sj*
	Sk	*val*
	Sj	*Sj*!*Sk*
	W@*field*,*An*	*Sj*

Example 38:

Location	Result	Operand
	Sj	W@*field*,*An*
	Sk	>N@*field*
	Sk	Sk>S@*field*
	Sj	#*Sk*&*Sj*
	Sk	*val*
	Sk	Sk<D'64-N@*field*
	Sj	*Sj*!*Sk*
	W@*field*,*An*	*Sj*

CRAY X-MP MODEL 48 MACHINE INSTRUCTION MACROS C

Machine instruction macros for CRAY X-MP Model 48 Computer Systems generate functions that CAL Version 2 will assemble. These macros are temporary implementations and will be replaced when CAL Version 2 becomes available. CRAY X-MP machine instruction macros emulate the following:

- CAL Version 2 machine instructions
- CAL Version 2 pseudo instruction (IFM)

MACHINE INSTRUCTIONS

CRAY X-MP Model 48 machine instruction macros generate the following machine instructions:

- CLN - Cluster number instructions
- Compress index instruction
- Extended memory addressing
- Gather/scatter instructions
- Interprocessor interrupts (CIPI/SIPI)

CLN - CLUSTER NUMBER INSTRUCTIONS

The following syntax sets the cluster number to *j* to make the following cluster selections:

CLN=0 No cluster; all shared register and semaphore operations are no-ops, (except SB, ST, or SM register reads, which return a 0 value to A*i* or S*i*).

CLN=*n* Cluster number; where *n* can be 1, 2, 3, 4, or 5. Clusters 1 through 5 each have a separate set of SM, SB, and ST registers.

Result	Operand	Description	Machine instruction
CLN	*j*	Select cluster number *j*	0014*j*3

COMPRESS INDEX INSTRUCTION

The compress index instructions create a vector mask identical to the 1750*jk* instruction (VM Vj,Z . . .). The compress index instructions, also, create a compressed index list in register V*i* that is based on the results of tests made on the contents of register V*j*.

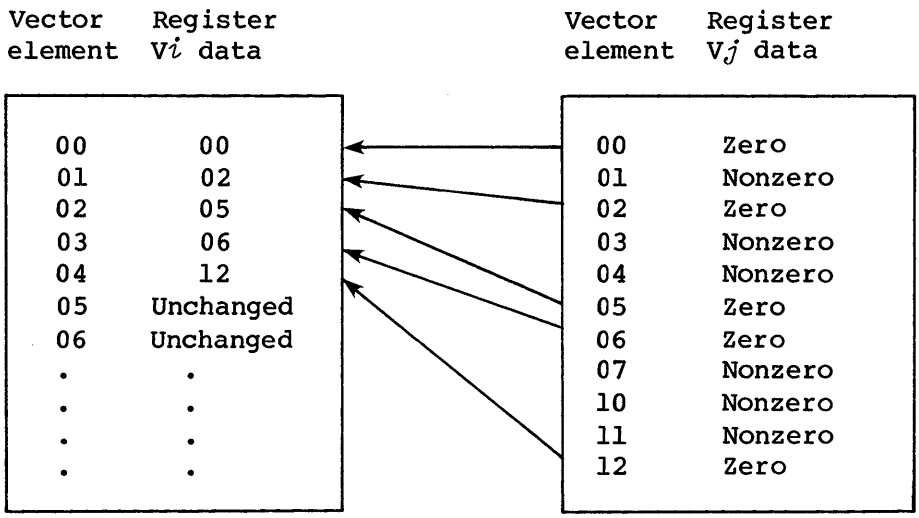
If an element of V*j* satisfies one of the following conditions (Z for zero, N for nonzero, P for positive, or M for minus), the corresponding bit in the vector mask is set, and the element number is placed in the next availabe element of V*i*.

Result	Operand	Description	Machine instruction
V*i*,VM	V*j*,Z	Compress index instruction	175*ij*4
V*i*,VM	V*j*,N	Compress index instruction	175*ij*5
V*i*,VM	V*j*,P	Compress index instruction	175*ij*6
V*i*,VM	V*j*,M	Compress index instruction	175*ij*7

Example:

This example of the compress index instruction 175*ij*4 generates the same vector mask as instruction 1750*j*0 and also generates data into vector register V*i* as follows:

Vector length=13_8



The vector length is set to 5 on completion of this operation.

EXTENDED MEMORY ADDRESSING

Extended memory addressing uses the 010-017 instructions to transmit an expression (*ijkm*) to register A*h*. The high-order bit of the *i* field determines whether the instruction is an extended load instruction or a jump instruction.

If the high-order bit in the *i*-field of the instruction is a 0, the instruction is always interpreted as a jump instruction. If the high-order bit in the *i*-field of the instruction is a 1 and is input to a CRAY X-MP Model 11, 12, 14, 22, or 24 Computer System, the instruction is interpreted as a jump instruction. If the high-order bit in the *i*-field of the instruction is a 1 and is input to a CRAY X-MP Model 48 Computer System, the instruction is interpreted as an extended load instruction with the 2 low-order *i*-field bits serving as the top 2 bits of the expression to be loaded.

NOTE

On CRAY-1 Models A, B, C, S, and M and on CRAY X-MP Models 11, 12, 14, 22, and 24, 01*hijkm* instructions are always interpreted as jump instructions by the hardware regardless of the high-order bit in the *i* field.

The following two examples illustrate the difference in the i-field between the extended load instruction and a jump instruction.

Example (jump instruction):

$$\underset{g}{\underline{0001}}\ \underset{h}{\underline{nnn}}\ \underset{i}{\underline{0nn}}\ \underset{j}{\underline{nnn}}\ \underset{k}{\underline{nnn}}\ \underset{m}{\underline{nnnnnnnnnnnnnnn}}$$

Example (extended load instruction):

$$\underset{g}{\underline{0001}}\ \underset{h}{\underline{nnn}}\ \underset{i}{\underline{1nn}}\ \underset{j}{\underline{nnn}}\ \underset{k}{\underline{nnn}}\ \underset{m}{\underline{nnnnnnnnnnnnnnn}}$$

Result	Operand	Description	Machine instruction
Ah	*exp*	Transmit $ijkm$ to Ah (high-order bit of the i-field=1)	01$hijkm$

GATHER/SCATTER INSTRUCTIONS

A0 is the base address for the load (gather) or store (scatter) of Vi. Successive elements of Vi are loaded (gathered) or stored (scattered) from addresses computed by adding the contents of A0 and Vk.

Result	Operand	Description	Machine instruction
Vi	,A0,Vk	Gather transmits (VL) words from memory to Vi elements using memory address (A0)+(V_k)	176i0k
,A0,Vk	Vj	Scatter transmits (VL) words from Vj elements to memory using memory address (A0) + (Vk elements)	1771jk

INTERPROCESSOR INTERRUPTS

Macros are available for clearing (CIPI) and setting (SIPI) interprocessor interrupts.

CIPI - Clear interprocessor interrupt

The clear interprocessor interrupt (CIPI) instruction clears the interrupt flag in the processor executing the instruction. On CRAY X-MP Model 48 Computer Systems, the *j* field ($0 \leq j \leq 3$) designates the number of the processor to be cleared.

Result	Operand	Description	Machine instruction
CIPI	*exp*	Clear interprocessor interrupt (*exp*=$0 \leq j \leq 3$)	001402

Example:

Code generated	Location	Result	Operand	Comment
	1	10	20	35
001402		CIPI		Pass (no op for CRAY X-MP Models 11, 12, and 14)
0014*j*2		CIPI	*j*	Clear interprocessor interrupt (CRAY X-MP Models 22, 24, and 48)

SIPI - Set interprocessor interrupt

The set interprocessor interrupt (SIPI) instruction sets the Interprocessor Interrupt flag in another processor. The request remains until cleared by the receiving CPU. On CRAY X-MP Computer Systems Models 22 and 24, SIPI causes an interrupt to be raised in the other processor regardless of the value of the *j* field. On CRAY X-MP Model 48 Computer Systems, the *j* field ($0 \leq j \leq 3$) desginates the number of the processor to be interrupted.

Result	Operand	Description	Machine instruction
SIPI		Set interprocessor interrupts	001401
SIPI	*exp*	Set interprocessor interrupts (exp=0≤j≤3)	0014j1

Example (CRAY X-MP Computer Systems Models 11, 12, and 14):

Code generated	Location	Result	Operand	Comment
	1	10	20	35
001401		SIPI		Pass (no op)
0014j1		SIPI	j	Pass (no op)

Example (CRAY X-MP Computer Systems Models 22 and 24):

Code generated	Location	Result	Operand	Comment
	1	10	20	35
001401		SIPI		Set inter-processor interrupt for the other processor
0014j1		SIPI	j	Set inter-processor interrupt for the other processor

Example (CRAY X-MP Computer Systems Model 48):

Code generated	Location	Result	Operand	Comment
	1	10	20	35
001401		SIPI		Set inter-processor interrupt for processor 0

Code generated	Location	Result	Operand	Comment
	1	10	20	35
0014*j*1		SIPI	*j*	Set inter-processor interrupt for processor *j*

IFM PSEUDO INSTRUCTION - TEST TARGET MACHINE ATTRIBUTES FOR ASSEMBLY CONDITION

The IFM pseudo instruction tests the hardware attributes of the target machine for assembly conditions. If the target machine has the specified hardware attribute, assembly continues with the next statement. If the attribute condition is false, subsequent statements are skipped. Skipping stops when an ENDIF or ELSE pseudo instruction with the same location field name as the name on the IFM pseudo instruction is encountered. If an assembly error is detected, assembly continues with the next instruction.

Location	Result	Operand
ifname	IFM	*attribute*

ifname Required with the ELSE or ENDIF pseudos; controls the skipping of subsequent instructions.

attribute Target machine hardware attribute name. Currently, only EMA (Extended Memory Addressing) and CIGS (Compress/Index, Gather/Scatter) hardware attributes can be tested. If the specified hardware is not present on the target machine, the condition is considered false and skipping is initiated. If the hardware is present, assembly continues until an ELSE pseudo instruction is encountered.

If a complement (#) sign preceeds a hardware attribute, subsequent lines are assembled only if the target machine does not have the specified attribute.

INDEX

INDEX

READERS COMMENT FORM

Macros and Opdefs Reference Manual SR-0012 A

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

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.

STAPLE

READERS COMMENT FORM

Macros and Opdefs Reference Manual SR-0012 A

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

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.

STAPLE