Introduction [1]

Cray Assembly Language (CAL) is a powerful symbolic language that generates object code for execution on Cray PVP systems.

Two types of CPUs are available with CRAY T90 systems. The first type of CPU uses the same type of floating-point format as all other Cray PVP systems. The second type of CPU conforms with the Institute of Electrical and Electronics Engineers (IEEE) standard 754, and except for minor differences, supports all 64-bit numeric representations, arithmetic operations, rounding modes, and exception handling.

CAL is supported on all Cray PVP systems with somewhat different instruction sets for each product line. The instruction sets for each machine are presented in appendix E, page 369.

Manual organization

1.1

This publication is organized as follows:

Section	Description
2	Describes the CAL invocation statement that executes under the UNICOS operating system. Section 2 also describes binary definition files.
3	Describes the organization of a CAL program.
4	Describes the statement syntax of the CAL program.
5	Describes the use of pseudo instructions.
6	Describes defined sequences available within CAL.
A	Lists descriptions of CAL pseudo instructions in alphabetical order.
B	Lists all CAL user messages.
C	Lists the character sets that CAL supports.

Section	Description
D	Provides tables of the symbolic machine instructions for Cray PVP systems.
E	Provides a table of all symbolic machine instructions for Cray PVP systems. The instructions are listed in numeric order by the opcode and a brief description of the instruction is given.

New features and modifications

1.2

The new instructions necessary to support IEEE floating-point format on CRAY T90 systems can be found in appendix E, page 369.

Capabilities

1.3

CAL provides the following capabilities:

- The free-field source statement format of CAL lets you control the size and location of source statement fields.
- With some exceptions, you can enter source statements in uppercase, lowercase, or mixed-case letters.
- You can assign code or data segments to specific areas to control local and common sections.
- You can use preloaded data by defining data areas during assembly and loading them with the program.
- You can designate data in integer, floating-point, and character code notation.
- You can specify addresses as either word or parcel addresses.
- You can control the content of the assembler listing.
- You can define a character string in a program and substitute the string for each occurrence of its micro name in the program by using micro coding.
- You can define sequences of code in a program or in a library and substitute the sequence for each occurrence of its macro name in the program by using macro coding.

Execution of the CAL assembler
1.4

The CAL assembler executes under the control of the UNICOS operating system. It has no hardware requirements beyond those required for the minimum system configuration.

When you specify the CAL invocation statement, the assembler is loaded and begins executing. The parameters used on the invocation statement specify the characteristics of an assembler run, such as the file containing source statements or listing output. For descriptions of the CAL command line, see section 2, page 11, or the as(1) man page.

The file containing source statements can include more than one CAL program segment. Each program segment is assembled as it is encountered in the source code. The CAL assembler makes two passes for each program segment. During the first pass, each source language instruction is read, sequences (such as macro instructions) are expanded, machine function codes are generated, and memory is assigned. During the second pass, values are substituted for symbolic operands and addresses; object code and an associated listing are generated.

The object code must be linked and loaded prior to execution to resolve references to external symbols. The absolute file that is created by the link and load process is suitable for execution.

Source statement format
1.5

CAL source programs consist of sequences of source statements. The source statement can be a symbolic machine instruction, pseudo instruction, macro instruction, or opdef instruction. The symbolic machine instructions provide a way of symbolically expressing all functions of a Cray PVP system. Pseudo instructions control the assembly process. Macros and opdefs define instruction sequences that can be called later in a program.

CAL source statements are free-format and can contain any or all of the following fields:

- Location
- Result
- Operand
- Comment

The content of each field is dependant upon the format specified by the `-f` (new format) or `-F` (old format) parameters on the `as` command line or by using the `FORMAT` pseudo instruction. Generally, fields are separated by white space (blanks or tabs). See subsection 3.2, page 36, for more information on the format of source statements.

The following is an example of a CAL source statement:

```
ABC         Si      Sj+Sk       ; Sum
```

In the preceding example `ABC` resides in the location field, S*i* in the result field, S*j*+S*k* in the operand filed, and `;Sum` in the comment field.

Assembler listings format

1.6

CAL generates a source statement listing and a cross-reference listing. You can control the format of these listings by using the listing control pseudo instructions (see subsection 5.6, page 121) or by using the `-i`, `-I`, `-l`, `-L`, `-a`, `-n`, `-h`, and `-H` options on the `as` command line (see subsection 2.1, page 12).

Each page of listing output produced by the CAL assembler contains three header lines. Figure 1 shows the format of the page header.

CAL version #	Title	Global page #
Date and time	Subtitle	Local page #
Section and qualifier	Scale (1–72 characters wide)	Cray Research system

Figure 1. Page header format

The three lines of the page header are described as follows:

- The first line contains, from left to right, the version number of CAL, the title of the program, and a page number that is global over all programs assembled by the current assembly. If you do not specify a title by using a `TITLE` pseudo instruction, the title is taken from the operand field of the `IDENT` pseudo instruction.
- The second line contains, from left to right, the date and time of assembly, a subtitle if specified with a `SUBTITLE` pseudo instruction, and a page number that is local for this listing.
- On the third line, the leftmost entry is a local section name if specified in a `SECTION` pseudo instruction. To the right of the local section name is a symbol qualifier name if specified by a `QUAL` pseudo instruction. The next field is a horizontal scale that is 72 characters wide, numbered from 1 through 72. This scale appears directly over your source code and helps you to differentiate the four fields of your source statements. On the far right of the third line is the name of the Cray Research system for which the code was generated.

Source statement listing format 1.6.1

The format of the source statement listing, as shown in Figure 2, appears directly under the page header and contains five columns of information, as follows:

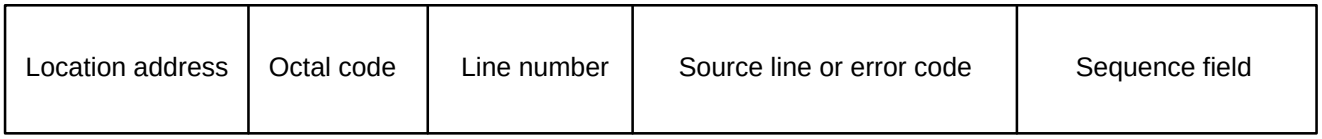

Figure 2. Source statement listing format

The five columns of the source statement listing are described as follows:

- The *location address* column contains the address of the current statement at assembly.

  If the statement is a machine instruction, it lists the parcel address with the parcel identifier `a`, `b`, `c`, or `d` appended to the word address. Parcels are lettered from left to right within a word. If the statement is not a machine instruction the address is listed as a word address.

- The *octal code* column contains the octal representation of the current instruction or numeric value.

  If the numeric value is an address, the octal code has one of the following suffixes:

  – `+` (Relocation in program block)
  – `C` (Common section)
  – `D` (Dynamic section)
  – `S` (Stack section)
  – `T` (Task common)
  – `Z` (Zero common)
  – `:` (Immobile attribute)
  – `X` (External symbol)
  – `None` (Absolute address)

  The results of several pseudo instructions can also appear in the octal code column:

  – The octal value of symbols defined by the `SET`, `MICSIZE`, or `=` pseudo instruction

  – The octal value of the number of words reserved by the `BSS` or `BSSZ` pseudo instruction

  – The octal value of the number of full parcels skipped as a result of the `ALIGN` pseudo instruction

  – The octal value of the number of characters in a micro string defined by a `MICRO`, `OCTMIC`, or `DECMIC` pseudo instruction

- The *line number* column contains the line number of the source code.

- The *source line* column is 72 characters wide and holds columns 1 through 72 of each source line.
- The *error code* column contains an error message immediately following a statement that contains an error. Error codes are described in appendix B, page 281.
- The *sequence field* column contains either an identifier or information taken from columns 73 through 90 of the source line image. It contains an identifier if the line is an expansion of a macro or opdef, or if the line was edited.

Cross-reference listing format
1.6.2

The assembler generates a cross-reference listing in the format shown in Figure 3. The assembler lists symbols alphabetically and groups them by qualifier if the QUAL pseudo instruction has declared qualifiers. If qualifiers were declared, each new qualifier appears on a fresh page. The qualifier name appears on the third line of the page header.

The cross-reference listing does not include unreferenced symbols that are defined between TEXT and ENDTEXT pseudo instructions and it does not include symbols of the form *%%xxxxxx*; x is zero or more identifier characters.

Note: The page header is nearly identical to the page header of the assembler listing; the difference is that the string *Symbol cross reference* is printed out in the middle field of the third line of the cross-reference listing.

CAL version #			Title	Global page #
Date and time			Subtitle	Local page #
Section and qualifier			"Symbol cross reference"	Cray Research system
Symbol	Value	.	Symbol references	

Figure 3. Cross-reference listing format

The information in each column is described as follows:

- The *symbol* column contains the symbol name.
- The *value* column contains the octal value of the symbol.

  A symbol with a parcel address attribute has a, b, c, or d appended to the word address. Parcels go from left to right within a word. The octal value of the symbol may have one of the following suffixes:

  – + (Relocation in program block)
  – C (Common section)
  – D (Dynamic section)
  – S (Stack section)
  – T (Task common)
  – Z (Zero common)
  – : (Immobile attribute)
  – X (External symbol)
  – None (Absolute address)

- The *period (.)* column separates the value reference field from the symbol reference fields and is called the separator.
- The *symbol references* column contains one or more references to the symbol.

  The assembler references symbols in the following format:

  page : *line x*

  page is the decimal representation of the local page number in the listing that contains the current reference. The local page number appears in parentheses at the far right end of the second line of the header.

  line is the decimal representation of the line number that contains the reference.

  x represents the type of reference as follows:

  – A *blank* column means the symbol value is used at this point.
  – D means the symbol is defined in the location field of an instruction or else by a SET, =, or EXT pseudo instruction.

- E means the symbol is an entry name.
- F means the symbol is used in an expression on an IFE, IFA, or ERRIF conditional pseudo instruction.
- R means the symbol is used in an address expression in a memory read instruction or as a B or T register symbol in an instruction that reads the B or T register.
- S means the symbol is used in an address expression in a memory store instruction or as a B or T register symbol in an instruction that stores a new value in the B or T register.

If a symbol is defined in text between TEXT and ENDTEXT pseudo instructions, the cross-reference listing reports the associated TEXT name on the line below the symbol reference.

If a symbol is defined in a binary definition file, the cross-reference listing reports the associated file name on the line below the symbol reference.