Chapter 1

DPEAC — A Quick Introduction

The DPEAC instruction set supports the floating-point arithmetic accelerator hardware of the CM-5. The accelerator hardware consists of four vector units (VU) on each processing node (PN), supplementing the normal SPARC Instruction Unit (IU) for the PN. (See Figure 1.)

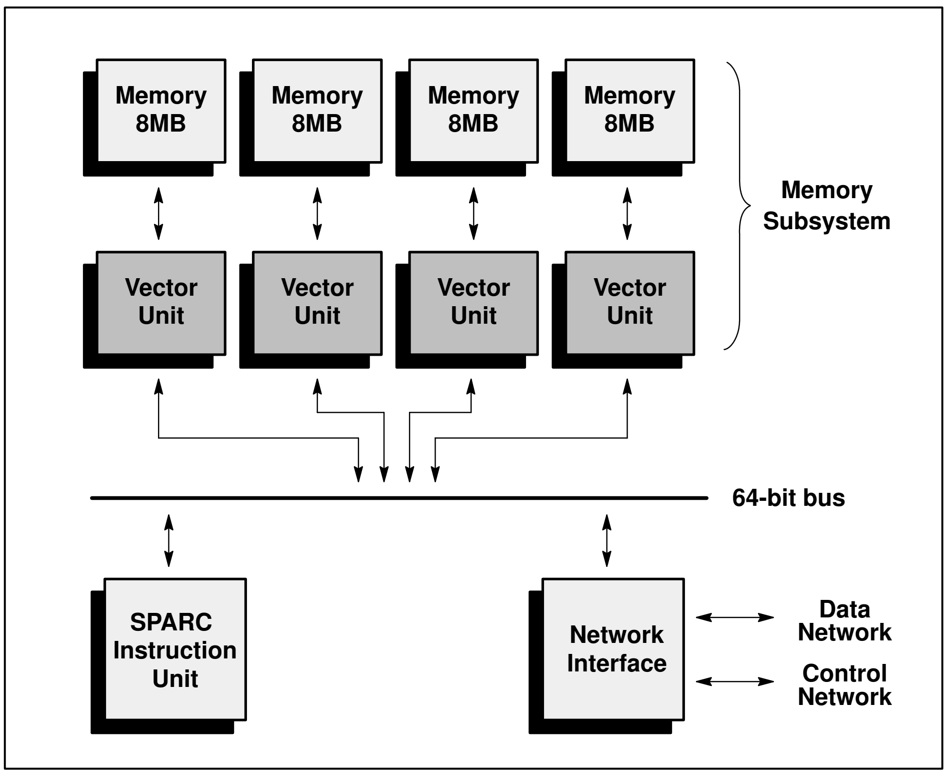

Figure 1. Components of a typical processing node (PN)

DPEAC is a superset of SPARC assembly code. A DPEAC source file contains SPARC assembly code into which instructions for controlling the VU hardware have been intermixed.

DPEAC is designed for three purposes:

- A PN object stream representation for the back ends of the compilers.
- Hand-coding algorithms for the IU/VU/MEMORY subsystem.
- Coding tests and diagnostics for the accelerator hardware.

1.1 The DPAS Assembler

DPEAC is implemented as an assembler, **`dpas`**, which acts as a fully compatible replacement for the the IU (SPARC) assembler, **`as`**. The **`dpas`** assembler reads DPEAC sources and outputs UNIX object (**`.o`**) files. It does this by preprocessing its source; translating the DPEAC-specific statements into SPARC assembly source; and running the resulting text through the **`as`** assembler.

(**Terminology Note:** The accelerator hardware was once known as the Datapath; hence the prevalence of the abbreviation “**`dp`**” throughout the **`dpas`** assembler. The name “**`dpas`**” itself reflects this old naming convention; in a sense, **`dpas`** is an “enhanced” version of **`as`**, specialized for the CM-5 accelerator hardware.)

The **`dpas`** assembler supports the **`-P`** switch of **`as`**, allowing source text to be initially passed through the **`cpp`** preprocessor. As an alternative, **`dpas`** provides an internal lexical translator, similar in syntax and functionality to **`cpp`** but with extended macro support. (In either case, the preprocessing syntax is not considered part of the DPEAC instruction set.)

Thus, assembly of DPEAC sources into UNIX object (**`.o`**) files is performed in three phases:

- Preprocessing of the DPEAC source to process the lexical directives.
- Translation of embedded DPEAC statements into SPARC assembler.
- Assembly of the resulting source code using **`as`**.

The output of the assembly is a UNIX object (**`.o`**) file, ready to be linked with other PN objects. (See Figure 2.)

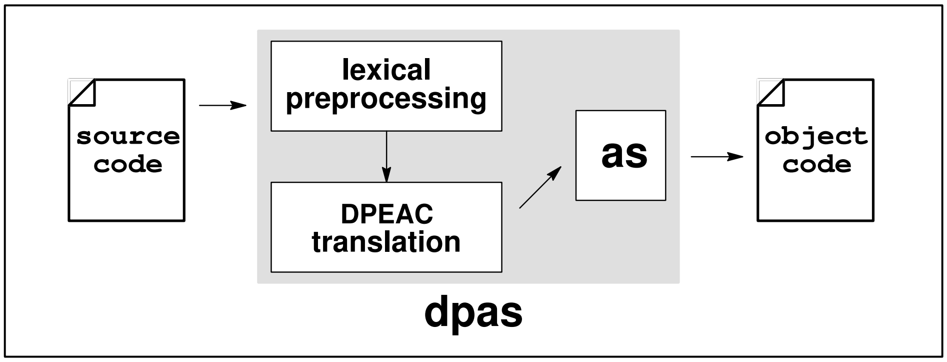

Figure 2. The `dpas` assember adds two extra steps to code processing

The internal lexical directives are described in a later section, after the description of the DPEAC language. The **`cpp`** preprocessor is described in the UNIX documentation.

The DPEAC instruction set consists of two groups of instructions:

- VU Instructions
- Control Instructions (Assembled into SPARC code; don't affect the VU.)

All lines of the source not recognized as an instruction belonging to one of these groups are ignored by the DPEAC translation mechanism and passed unchanged to **`as`**. Recognition is based on the first word of the line (following the label if present), the instruction mnemonic. As a result, misspellings of instruction mnemonics will not be recognized until the assembler phase.

1.2 Organization of This Manual

1.2.1 Structure of Chapters

Chapters 3 through 7 present the syntax and formats of DPEAC, and Chapter 10 provides useful performance information. Chapters 2 and 9 describe aspects of the hardware implementation of the accelerator chip as they relate to DPEAC programming. Finally, Chapter 12 provides a sample of DPEAC code.

Chapter 8 describes the preprocessor component of the **`dpas`** assembler, and Chapter 11 describes the command line interface of **`dpas`**.

1.2.2 Meta-Syntax Used in This Manual

The following meta-syntactic characters and forms are used throughout this manual to describe the format, options, and modifiers of instructions:

{`a`,`b`,`c`} indicates a choice of exactly one of **`a`**, **`b`**, or **`c`**.

{[`a`,`b`,`c`]} is the same but indicates that the selection is optional, and can be omitted altogether.

Literal braces (`[]`) appear in DPEAC code as written, with the exception of braces used to indicate an optional suffix of a command, for example **`ld`**[**`d`**].

Rxx is a placeholder for an operand of a DPEAC instruction. The *xx* suffix indicates the purpose of the operand:

Rs1, *Rs2*	—	Source registers
Rd	—	Destination register
Ria	—	Indirect Address register
Rls	—	Load/Store register

Note: Actual register names are given as **`R0`**, **`R1`**, ... **`R127`**. (See Chapter 3.)

Vxx is similar to *Rxx*, but indicates that the operand must be one of the vector register designators (**`Vn`**) or the first **`Rnn`** register of an eight–register vector: **`R0`**, **`R8`**, **`R16`**, etc. (This is also described in Chapter 3.)

Sxx is also similar to *Rxx*, and indicates the operand must be a scalar register. For single precision operations, scalar registers are named **`S0`** through **`S15`** and are equivalent to the registers **`R0`** through **`R15`**. For double precision, the scalar registers are the even-numbered registers in the range **`S0`** through **`S30`** (or, equivalently, **`R0`** through **`R30`**).