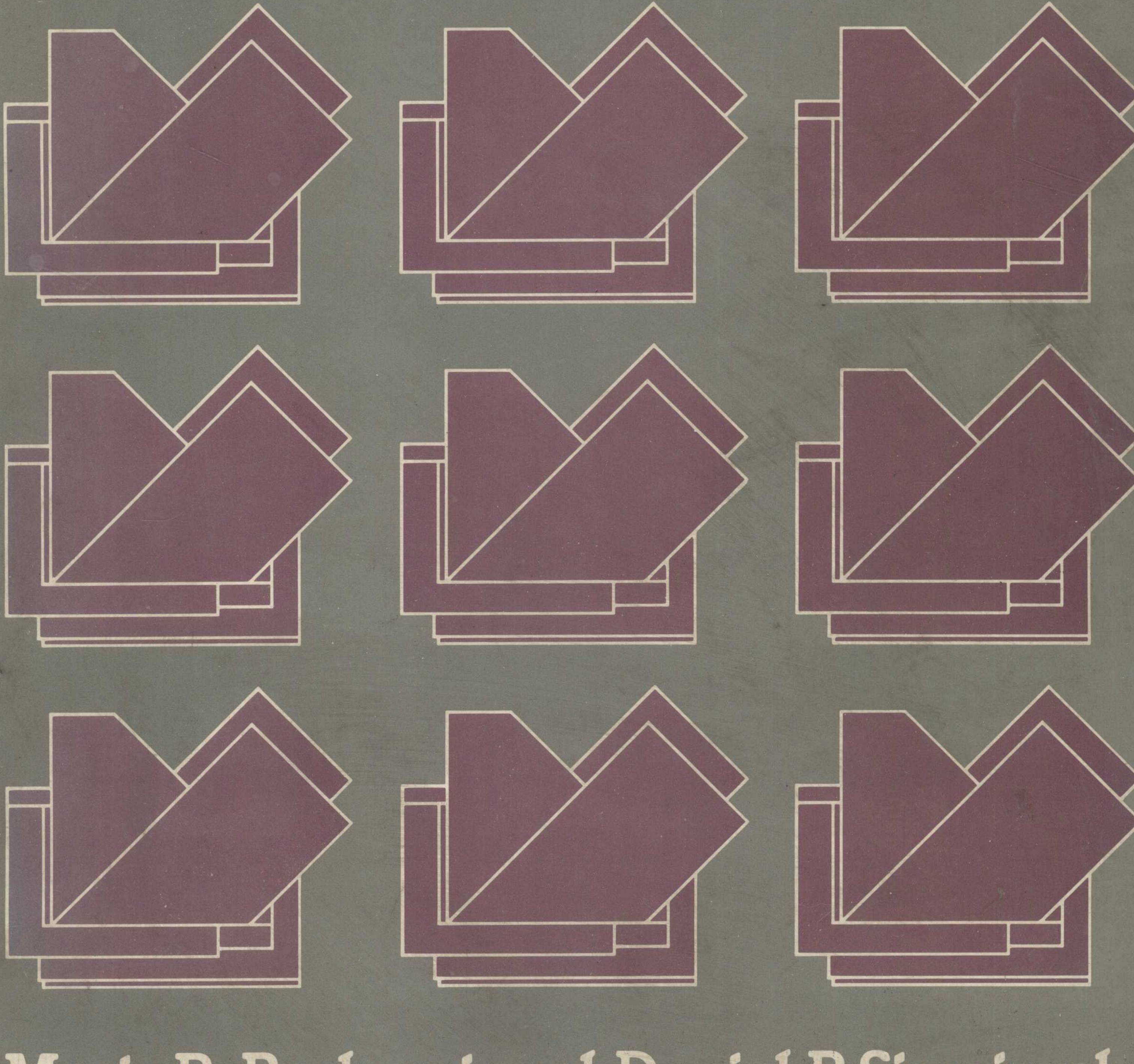
COMPUTER SCIENCE SERIES

THE DESIGN AND ANALYSIS OF INSTRUCTION SET PROCESSORS



Mario R. Barbacci and Daniel P. Siewiorek

The Design and Analysis of Instruction Set Processors

Mario R. Barbacci

Daniel P. Siewiorek

Carnegie-Mellon University

McGraw-Hill Book Company

New York St. Louis San Francisco Auckland Bogotá Hamburg Johannesburg
London Madrid Mexico Montreal New Dehli Panama Paris São Paulo Singapore
Sydney Tokyo Toronto

The Design and Analysis of Instruction Set Processors

2 3 4 5 6 7 8 9 0 S M S M 8 9 8 7 6 5 4 3 2

ISBN 0-07-057303-4

The editor was James E. Vastyan; the production supervisor was Joe Campanella.
Semline, Inc., was printer and binder.

Library of Congress Cataloging in Publication Data

Barbacci, Mario R.
The design and analysis of instruction set processors.

(McGraw-Hill computer science series)
Bibliography: p.
Includes index.
1. Computer architecture. 2. ISP (Computer program language) I. Siewiorek, Daniel P. II. Title. III. Series.
QA76.9.A73B37 621.3819'52 81-23620
ISBN 0-07-057303-4 AACR2

The Design and Analysis of Instruction Set Processors

to our parents

Preface

Computer Structure: Readings and Examples [Bell and Newell 1971] created a methodology to study and compare computer systems. One of the vehicles used in the book was a computer description notation: ISP. Since the ISP descriptions in Readings and Examples were used exclusively for presentation of the machines (i.e., 'read only'), the notation was not formally defined. Since the publication of Readings and Examples, we have gone through two iterations on the design and implementation of a computer description language based on ISP. The latest version, ISPS [Barbacci et al. 1977], is being used at many universities and companies as a design tool. Computer Structures: Principles and Examples [Siewiorek, Bell, and Newell 1982] uses ISPS as the computer description language.

This book is designed to present the student with a notation and methodology for the analysis of computer architectures. The overall motivation is to present the space of architecture features spanned by a collection of representative machines rather than presenting yet another paper machine, designed solely for pedagogical reasons.

There are several reasons why a study of real machines is a better vehicle towards an understanding of the architecture design process. Fundamentally, every architect must have an understanding of the underlying technologies used to implement a computer. Technology affects the state of the art by determining the speed and cost of the memory and central processor. These determine the basic data types and operators of the machine, the architect's building blocks. Market requirements also bias the design of instruction sets towards specific application areas, languages, or modes of operation. These two forces, together with the architect's own vision of the design space are not always in agreement and compromises must be achieved. By exploring real machines we attempt first, to understand the different dimensions of the space and second, to quantify them. It is easy to see why paper machines won't do. They are always remarkably adequate for the task on hand, a result rarely achieved in the real world. Moreover, they fail to present the complete picture: the compromises made in light of conflicting requirements, the sins committed during the design, and more important, the attempts at fixing these in later versions.

Four machines, ranging from small minicomputers to large mainframes, are used as running examples. The first minicomputer, the DEC PDP-8, serves as an example of a simple Instruction Set Processor. The DEC PDP-11 represents a sophisticated 16-bit minicomputer architecture. The IBM System/370 represents the first planned computer family. Finally, the CDC 6600 is a high performance scientific architecture.

In the process of writing complete formal descriptions, one must include many details that could be left out otherwise. Principles and Examples only included the complete descriptions for the simplest machines. By including complete descriptions, this book can also be used to complement Principles and Examples by presenting an orthogonal view of the computer space. While in Principles and Examples, chapters are organized around machines and the features implemented in their instruction set, this book is organized around features and the machines that include them. This organization is also suitable for the use of problems and exercises to test the student's comprehension of a topic. The book includes actual problems and suggestions for problems of the form: Compare feature X as implemented in machines A, B, and C. How would you add feature Y to machine D? How would you subset (eliminate) feature Z from machine N? Provide alternative mechanisms for a missing or incorrect implementation of a feature?, etc.

Each chapter of this book is meant to illustrate some aspect of the architecture space. Each feature is presented and discussed in terms of the same set of machines. The student is assumed to have some background in digital logic, as described in courses DL-1 and DL-2 of the IEEE Curriculum [IEEE 1976], as well as some background in Assembly Language programming, numeric representation in different bases, and conversion between bases, as covered in courses CS-3 and CS-4 of the 1978 ACM Curriculum [Austing et al. 1979]. This book can be used in Computer Organization or Computer Architecture courses (IEEE CO-1, IEEE CO-3, or ACM CS-6).

Many computer description languages have been proposed and one's choice must be supported by something more than pride of authorship. Initially ISP was introduced mainly for publication purposes. Its implementation as a computer language has expanded considerably its usefulness. In contrast with existing hardware description languages, ISPS is used for high level, behavioral descriptions and has been successfully used in areas outside the traditional realm of hardware descriptions: simulation and synthesis of combinational and sequential logic. ISPS has been used to drive both hardware and software generators. It has been used to evaluate computer architectures and to verify software correctness. Its use at CMU and elsewhere has produced a growing library of (real and idealized) machine descriptions, readily available for students and researchers alike [Barbacci 1981]. This achievement places ISPS in a class of its own. Finally, another incentive for its use is the availability of software (compiler, simulator, CAD systems, etc.) which can be used as laboratory tools.

Organization of the Book

We place a heavy emphasis on the ability to read and understand instruction set descriptions in ISPS and we provide in Chapter 1 a 'readers guide' to the notation. The material is not original and has appeared both in [Siewiorek, Bell, and Newell 1982] and [Bell, Mudge, and McNamara 1978]. In addition, earlier versions have been used for several years in computer architecture courses at Carnegie-Mellon University. Readers familiar with the notation can safely skip it. Nevertheless, it is advisable to read the last section of Chapter 1, in which we describe the convention used in writing the examples and full descriptions. The rules for capitalizing names, variables, and operators were introduced as a means to aid in the readability of ISPS. Of course, these are not part of the language, and are offered only as a guide towards good style.

An instruction set processor operates by interpreting bits of information stored in the memory and registers of the machine. We begin our study of the architecture space by describing and comparing the fundamental information units. Chapter 2 introduces the different data types and the operations performed by the machines, how these operations interpret data as integers, floating point numbers, characters, strings, etc. and how the information can be mapped from one format to another.

Although instructions are one more data type, they play such an important role that they deserve a separate treatment, in Chapter 3. New instruction types appear whenever a new feature is introduced. Thus, while this chapter describes the basic formats, new ones will appear throughout the book.

Chapter 4 deals with the techniques use to extend the address space of the machine. Memory management, relocation, and virtual address translation are the core of the chapter. Address faults, error recovery, and interrupts are introduced here and continued in Chapters 5 and 6.

Chapter 5 deals with a fundamental property of an instruction set processor, namely, the ability to modify its behavior by chosing alternative instruction sequences based on the result of previous instructions. Program control and subroutines are discussed. Arithmetic operations, because of the finite precision of the arithmetic units, have certain well defined exceptional conditions which result in the introduction of condition codes in the processor state. In this chapter we describe what these are and how they are computed across the different machines.

Chapter 6 places the processor in the context of a computer system. Peripheral devices for input/output and as secondary memories constitute the main topics.

Chapter 7 deals with the design of instruction sets, the symmetry and usefulness of the instructions.

Chapter 8 treats architecture measurements. Our work on evaluating computer architectures for the Department of Defense is based on the use of implementation independent figures of merit. The use of abstract architecture parameters to evaluate and compare architectures and the use of a formal computer description language to drive the collection of data is a novel topic and we believe it will prove to be one of the most successful contributions of this book.

Selected portions of the ISPS descriptions are included throughout the book. Sometimes, in order to keep an example concise or free of details not relevant to the topic on hand these portions have been slightly abridged. The full ISPS descriptions however, appear in the Appendices

We would like to thank Dorothy Josephson who typed portions of the manuscript. Gary Leive is to be commended for his effort in the writing of the ISPS descriptions. Jin Kim, Mickey Tsao, and Andy Wilson edited the ISPS descriptions and brought them in agreement with the ISPS writing style guidelines used throughout the book.

This book was edited and composed by the authors on a DECSystem10 in the Department of Computer Science at Carnegie-Mellon University. The camera-ready copy of this book was produced by the Scribe document compiler and printed on a GSI CAT-8 Photocomposer at the Campus Printing Office of Carnegie-Mellon University.

Mario R. Barbacci
Daniel P. Siewiorek

Table of Contents

List of Figures

List of Tables

1. The ISPS Notation

This chapter introduces the reader to the ISPS notation. Although some details have been excluded, it covers enough of the language to provide a *reading* capability, to permit the reading and study of complex descriptions. For a detailed explanation of the complete language the reader must consult the ISPS reference manual [Barbacci et al. 1977].

1.1. Instruction Set Processor Descriptions

To describe an Instruction Set Processor (ISP), we need to define the operations, instructions, data types, and interpretation rules used in the machine. These will be introduced gradually, as we describe the primary memory state, the processor state, and the interpretation cycle. Primary memory is not, in a strict sense, part of the Instruction Set Processor but it plays such an important role in its operation that it is typically included in the description. In general, data types (integers, floating point numbers, characters, addresses etc.) are abstractions of the contents of the machine registers and memories. One data type that requires explicit treatment is the "instruction" and we shall explore the interpretation of instructions in great detail.

We will use the DEC PDP-8 ISPS description as a source of examples.

Memory State

The description of the PDP-8 begins by specifying the primary memory that is used to store data and instructions:

```
M\Memory[0:4095]<0:11>,
```

The primary memory is declared as an array of 4096 words, each 12 bits wide. The memory has a name "M", and an alias "Memory". These aliases are a special form of a comment and are useful for indicating the meaning or usage of a register's name. As in most programming languages, ISPS identifiers consist of letters and digits, beginning with a letter. The character "." is also allowed, to increase the readability. The expression [0:4095] describes the structure of the array. It declares the size (4096 words) and the names of the words (0,1,..., 4094,4095). The expression <0:11> describes the structure of each individual word. It declares the size (12 bits) and the names of the bits (0,1,...,10,11).

It should be noted that bit and word *names* are precisely that, i.e., identifiers for the subcomponents of a memory structure. These names do not necessarily indicate the relative position of the subcomponents. Thus, R<7:3> is a valid definition of a 5-bit register. The fact that the five bits are named 7,6,5,4,3 should not be confused with the 7th, 6th, etc. positions inside the register. Thus, bit 7 is the leftmost bit, bit 6 is located in the next position towards its right, etc., while bit 3 is the rightmost bit.