HIGH-LEVEL LANGUAGE COMPUTER ARCHITECTURE

Edited by

Yaohan Chu

High-Level Language Computer Architecture

CONTRIBUTORS

Howard M. Bloom
Carl R. Carlson
Yaohan Chu
Robert W. Doran
Theodore A. Laliotis
Bernard J. Robinet

High-Level Language Computer Architecture

Edited by

Yaohan Chu

Department of Computer Science
and
Department of Electrical Engineering
University of Maryland
College Park, Maryland

ACADEMIC PRESS New York San Francisco London 1975

A Subsidiary of Harcourt Brace Jovanovich, Publishers

Figure 1, p. 32, adapted by permission from Lonergan and King, *Datamation* 7, 28–32, Copyright 1961 Association for Computing Machinery, Inc.
Figures 6 and 7, pp. 42, 43, adapted by permission from Sugimoto, *Proc. ACM*, 519–538, Copyright 1969 Association for Computing Machinery, Inc.
Figure 9, p. 51, adapted by permission from Nissen and Wallach, *Proc. Symp. High-Level Language Computer Architecture*, 43–51, Copyright 1973 Association for Computing Machinery, Inc.

ACADEMIC PRESS, INC.
111 Fifth Avenue, New York, New York 10003

United Kingdom Edition published by
ACADEMIC PRESS, INC. (LONDON) LTD.
24/28 Oval Road, London NW1

Library of Congress Cataloging in Publication Data
Main entry under title:

High-level language computer architecture.

Includes bibliographical references and index.
1. Electronic digital computers–Programming.
2. Programming languages (Electronic computers)
3. Electronic digital computers–Design and construction. I. Chu, Yaohan, (date)
QA76.6.H53 001.6'42 75-3580
ISBN 0–12–174150–8

PRINTED IN THE UNITED STATES OF AMERICA

To REX RICE and WILLIAM R. SMITH,
whose team at Fairchild designed and constructed
the revolutionary SYMBOL computer system.

This page intentionally left blank

Contents

List of Contributors

Numbers in parentheses indicate the pages on which the author's contributions begin.

HOWARD M. BLOOM (187), Management Information Systems Group, Harry Diamond Laboratories, Adelphi, Maryland

CARL R. CARLSON (31), Computer Sciences Department, Northwestern University, Evanston, Illinois

YAOHAN CHU (1, 15), Department of Computer Science and Department of Electrical Engineering, University of Maryland, College Park, Maryland

ROBERT W. DORAN (63), Department of Computer Science, Massey University, Palmerston North, New Zealand

THEODORE A. LALIOTIS (109), Systems Technology Division, Fairchild Camera and Instrument Corporation, Palo Alto, California

BERNARD J. ROBINET (243), Institut de Programmation, Université Pierre et Marie Curie, Paris, France

This page intentionally left blank

Preface

As W. M. McKeeman pointed out, it was an accident that the digital computer was organized like a desk calculator. As a result, the digital computer of today requires a multitude of software. As software gets less reliable and consumes more memory, it needs more time to debug, takes more system programmers, and spends more overhead computer time. The progress of computer utilization and applications is now greatly limited to the development of software. Yet, the progress of software development is handicapped by the desk-calculator-like organization. The computer community, as also pointed out by McKeeman, could be worse off if the digital computer were organized like a Turing machine. In this case, a lot more programmers would have been needed. A great deal more effort would have been spent in improving arithmetic routines, in discovering ways of tape memory addressing and allocation, in creating more levels of programming languages, and in facing an unmanageable maze of software. Obviously, the airplane today would not fly at sonic speeds or supersonic speeds if we were continuously improving piston engines. Likewise, the problem of air pollution by tens of millions of today's automobiles could not be resolved if we continue to build the conventional internal combustion engines. Thus, could we expect carefully engineered, very fast, automatic desk calculators to be very good for implementing compilers or operating systems?

A modern digital computer consists of a data structure, a control structure, and a processing structure for executing algorithms. If the computer is going to execute programs written in a high-level programming language such as FORTRAN or ALGOL, there is no need to have the structures for executing programs written in an assembly-type language (i.e., a conventional machine language) or to compile the high-level language program into an assembly-type language before execution. The structures of

the computer might as well have been conceived and designed for directly accepting and executing programs written in the high-level language. Such a computer architecture is called a *high-level language computer architecture.* A high-level language architecture mirrors closely the data, control, and processing structures of the high-level programming language or languages, and the high-level programming language is indeed the machine language.

This volume aims to fill a current need of tutorial material on high-level language computer architecture. The first chapter presents a classification of high-level language computer architecture according to the proximity of the machine language and the programming language. This classification gives four types: von Neumann architecture, syntax-oriented architecture, indirect execution architecture, and direct execution architecture. In order to illustrate the possible evolution of computer architecture, design concepts of Japanese-language data processing systems are chosen as an example and presented in the next chapter.

The chapter by Carlson surveys the high-level language computer architecture. That by Doran describes the syntax-oriented architecture. The chapter by Laliotis is a tutorial on the historical SYMBOL computer system that was developed by Fairchild Corporation and is now being evaluated by the Iowa State University. The SYMBOL system makes use of an indirect execution architecture. The chapter by Bloom presents design concepts of direct-execution architecture for the ALGOL 60 language. Lastly, the chapter by Robinet describes the architecture for the processor for an APL subset.

The editor would like to acknowledge the help of Ms. Joanie Fort in the preparation of the manuscript.

Concepts of High-Level Language Computer Architecture

Yaohan Chu

Department of Computer Science
and
Department of Electrical Engineering
University of Maryland
College Park, Maryland

1. INTRODUCTION

In this volume, the term "high-level language" (HLL) refers to those computer programming languages that not only allow the use of symbolic operators to signify operations and of symbolic names to represent data and data structures, but also are structured with syntax and semantics to describe the computing algorithm. Examples of such programming languages are FORTRAN, ALGOL, and COBOL.

1.1 What Is a HLL Computer System?

A high-level language computer system is one that can accept and execute a high-level language program. There are many high-level language computer systems today, such as those IBM, CDC, and UNIVAC computer systems that are provided with compilers for these and other high-level programming languages. There are others, such as the Burroughs B5700/6700, that are claimed to be more efficient in handling some high-level programming languages. Finally, there are research computer systems such as the SYMBOL computer system developed by Fairchild and now being evaluated at Iowa State University and those now being studied at the University of Maryland.

1.2 Is the HLL Computer a General-Purpose Computer?

The general-purpose digital computer of today is a stored-program computer that has an instruction set (commonly called the machine language) and a hierarchy of storages for storing the program and the data. The instruction set is capable of describing algorithms for solving a problem in a large class of application areas. It is limited by the capacity of the storage and by the program that can be written with the instruction set. Therefore, if storage capacity is adequate, the question to ask of general-purpose computers is how universal is the instruction set.

High-level programming languages can be as general purpose as the conventional general-purpose computer instruction set. For example, FORTRAN has been commonly used for solving scientific problems, while COBOL has been widely used for solving data processing problems. Another high-level programming language, SNOBOL, has been developed for describing symbol string manipulations, while ESPOL (a dialect of ALGOL) was developed and used to describe the operating system of the Burroughs B5500 computer system. High-level programming language PL/1, which incorporates language features of FORTRAN, ALGOL, and COBOL, has been developed for both scientific computing and data processing and has also been used to describe an operating system. High-level language CDL (Chu [1965, 1972]) has been used to describe the processor, control, and storage structures of a computer. Therefore, the implementation of a high-level language as the machine language instead of a conventional machine language could make the HLL computer system general purpose. In other words, it is the machine language(s) that determine whether the computer system is general purpose or not.