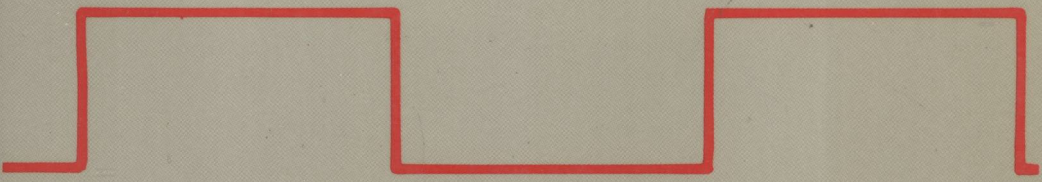


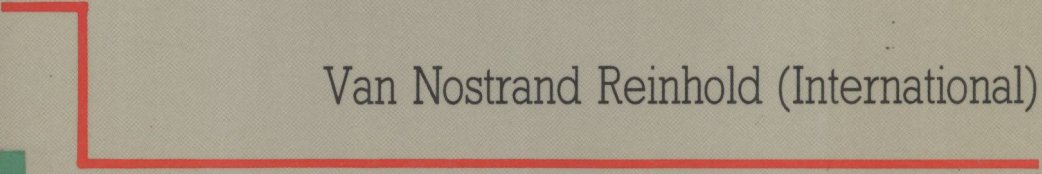
Computers



from logic to architecture



R.D. Dowsing and F.W.D. Woodhams



Van Nostrand Reinhold (International)

TP3
D752

9061357

COMPUTERS

from logic to architecture

R.D. DOWSING and F.W.D. WOODHAMS

*School of Information Systems
University of East Anglia
NORWICH*



E9061357



International

Van Nostrand Reinhold (International)

Published in 1990 by
Van Nostrand Reinhold (International) Co. Ltd
11 New Fetter Lane, London EC4P 4EE

First published in 1985 as Computer Architecture:
A first course

© 1990 R.D. Dowsing and F.W.D. Woodhams

Typeset in 10/12 Times by
Best-Set Typesetter Limited, Hong Kong
Printed in Great Britain by
TJ Press Ltd, Padstow, Cornwall

ISBN 0 278 00093 2

This paperback edition is sold subject to the condition that it shall not, by way of trade or otherwise, be lent, resold, hired out, or otherwise circulated without the publisher's prior consent in any form of binding or cover other than that in which it is published and without a similar condition including this condition being imposed on the subsequent purchaser.

All rights reserved. No part of this book may be reprinted or reproduced, or utilized in any form or by any electronic, mechanical or other means, now known or hereafter invented, including photocopying and recording, or in any information storage and retrieval system, without permission in writing from the publisher.

British Library Cataloguing in Publication Data

Dowsing, R. (Roy)
[Computer architecture]. Computers: from
logic to architecture.
1. Computer systems
I. [Computer architecture] II. Title
III. Woodhams, Frank
004

ISBN 0-278-00093-2

COMPUTERS

Preface

With the ever increasing use of computers there is an increasing need for students to be trained in the techniques of computing. One specific area which has been targeted by many colleges, polytechnics and universities is that of computer systems engineering; the interface area between computing and electronics. The specific requirements of this area of study are a knowledge of the interaction of the software and hardware of computers, often with reference to microprocessors and their applications.

This book is aimed at providing a first course in computer architecture: the interaction of hardware and software. The reader does not require any specific prior knowledge but the student who has at least a little experience of programming in a high-level language will find the latter half of the book easier reading. The book covers the spectrum of computer architecture topics from technology through to systems software and communications.

As in our previous book, we had the problem of deciding what hardware to use for examples. We decided, eventually, to use two systems, the 8-bit Intel 8085 and the 16/32-bit Motorola 68000 to show the differences between a simple 8-bit system and a sophisticated 16/32-bit machine. Also the use of two different microprocessors enables the student to see the different design choices made by the designers.

The book takes a bottom-up approach to the subject, starting at the lowest level, logic, and building up the hardware and software architecture of the computer from this basis.

Chapter 1 introduces some of the important concepts in understanding computer architecture whilst the next chapter introduces some of the concepts required to understand logic and logic design. It deals with Boolean algebra, truth tables and the different types of electronic logic component. Chapter 3 deals with combinatorial logic design, i.e. the design of circuits whose output depends solely on their inputs. It shows how the combinatorial elements of a computer, such as decoders and multiplexers, can be formed

x Preface

from simple logic networks. Chapter 4 describes sequential logic elements, those whose output is determined by past actions as well as present inputs. Latches, registers and memory elements are discussed in this chapter. The structure of a computer is discussed in the next chapter, showing how the elements of the computer are interconnected. Details of specific computer components are given as well as the general architecture.

Chapter 6 considers the different types of memory available, such as RAM and ROM, and the organization of memory into a hierarchy. Input–output processing techniques such as polling, interrupts and DMA are considered in the next chapter. Microprogramming and how it can be used to implement the control unit of a processor is considered in Chapter 8, and the following chapter presents the design of two small computer systems based on the 8085 and 68000 processors and components described in previous chapters. Chapter 10 describes the different types of data that can be manipulated in a computer and the operations that can be performed on them. Detailed discussion of number systems is dealt with here. Instruction sets and addressing modes found in computers with specific reference to the 8085 and 68000 are considered in the next chapter. Many examples are given and some complete programs are given at the end of the chapter. A chapter introducing system software such as assemblers, linkers and loaders follows. Of necessity, the discussion is brief but most of the software falling into this category is covered. Data communications and the way in which the development of communication networks has led to the introduction of distributed computing is the subject of Chapter 13, whilst the final chapter considers some new approaches to the design of computers, specifically RISC designs and the transputer.

Many people have contributed to the material in this book, including the numerous students to whom we have taught the material. We would like to thank them all, especially our colleagues Ian Marshall and George Turner, for their helpful comments and criticisms of drafts of the manuscript.

R.D. Dowsing
F.W.D. Woodhams

Glossary

- Absolute address** actual memory address used to access data or instructions in memory
- Access time** delay between time supplying address to memory and receiving data
- Accumulator** special register in the CPU, often the destination of arithmetic and logical operations and, sometimes, one of the source operands
- Addressing mode** method of specifying the address of an operand from the address bits in the instruction
- Algorithm** sequence of steps required to solve a problem
- ALU** arithmetic and logic unit
- ASCII** American Standard Code for Information Interchange
- Arithmetic and logic unit** a hardware unit which performs operations such as addition on its operands according to the function code supplied
- Assembler** the program which converts input in assembly language to machine code
- Assembly code** representation of machine instructions where bit patterns are replaced by symbols
- Base** the radix of the number system in use, for example, 2 for binary
- BCD** binary coded decimal
- Binary** representation of numbers in base 2, that is, using the digits 0 and 1 only
- Binary coded decimal** a method of number representation where each decimal digit is encoded into 4 bits
- Bit** binary digit, taking one of the values 0 or 1
- Bridge** a node which connects two networks using similar protocols
- Bus** a group of wires carrying information between subsystems
- Byte** a group of 8 bits
- Cache** small fast memory between processor and main memory
- Central processing unit** the arithmetic and logic unit together with registers and control logic for decoding and obeying instructions

xii Glossary

- Channel** any medium which carries data
- Circuit switching** a switching technique in which a route is first set up, then used for data transmission and finally closed down
- Clock** source of regular pulses to control the system operation
- Combinatorial** a circuit whose output depends only on its current inputs
- Compiler** a program which translates a high-level language program into a lower-level one, frequently machine code
- Condition flags** flags normally used to indicate carry, sign, overflow and zero as the result of the last instruction
- Complement** 1's complement – inverting all the bits of the binary value; 2's complement – 1's complement + 1
- CPU** central processing unit
- Cycle stealing** using memory time slots not used by the CPU
- Data selector** a programmable switch
- Deadlock** a situation where no processing can proceed because two or more processes are waiting for each other cyclically
- Direct memory access** method of transferring large quantities of information between memory and an input–output device without intervention from the CPU
- Distributed computing** computing spread over several processors
- DMA** direct memory access
- Fetch** that part of the instruction cycle concerned with bringing the next instruction to be executed from memory into the instruction register
- Flag** a single-bit hardware marker indicating something about the state of the computer
- Flip-flop** a single-bit memory device
- Full adder** a circuit that accepts two single-bit operands and a carry producing their sum and a carry out
- Gateway** a node which connects together two dissimilar networks
- Half adder** a circuit that accepts two single-bit operands and produces their sum and carry
- Handshake** one or more signals controlling (synchronizing) the transfer of data between sender and receiver
- Hardware** that part of a computer implemented by electronic and mechanical components
- Hexadecimal** number system using base 16 with symbols digits 0 to 9 and letters A to F
- High-level language** a language where each statement corresponds to several machine-code instructions; a language which is more expressive than machine code
- Indirect address** an address which refers to a location containing the address of the required value
- Input–output** the interface and devices by means of which the computer communicates with the outside world
- Instruction** a collection of bits containing an operation code and, possibly,

one or more operands

Instruction register register in the CPU used for holding the current instruction whilst it is being decoded

Instruction set the repertoire of instructions available on a particular computer

Interpreter a program which directly executes statements in a language without prior translation

Interrupt a method of a device informing the CPU that it needs attention

I/O input–output

K 1024, e.g. $2K = 2 \times 1024 = 2048$

Link editor program which fills in the cross references between separately compiled subprograms

Literal any symbolic value representing a constant

Loader program which loads a binary program into memory

Local area network a network which extends over a small area such as a building or single site

LSI large scale integration – integrated circuits large enough to hold a microprocessor on a single chip

Macroprocessor a program which performs text substitution, replacing one input statement by several output statements; can be used for language translation

Machine code a representation of the bit pattern of an instruction, often in hexadecimal

Memory mapped I/O an addressing scheme where the registers concerned with input–output have addresses in the normal memory address space

Microprocessor a CPU implemented in LSI or VLSI

Microprogram a set of instructions, normally in read-only memory, which are used to implement the instruction set of the computer

Mnemonic in computing, a symbol representing a bit string

MOS metal oxide semiconductor

Multiplexer a switch which allows several inputs to share the same output, but not at the same time

Multiprogramming the running of several processes on a single processor by time-division multiplexing

Operand a value to be operated on by the opcode in an instruction

Operation code opcode

Opcode that part of an instruction which defines the operation to be performed

OSI model a standard model for protocol levels in networks

Packet switching a technique for sending messages across networks as fixed-sized units

Page a contiguous block of memory space

Paging a mechanism for swapping information between main memory and backing store

Peripherals input–output devices

xiv Glossary

- PLA** programmable logic array – a regular array of AND and OR gates which may be connected together (programmed) to produce the required logic function
- Polling** interrogation of devices to find their status
- Port** an external entry or exit point from an interface
- Process** a program or subprogram in execution
- Program counter** register in the CPU holding the address of the next instruction to be executed
- PROM** programmable read-only memory – a reusable programmable memory which can be programmed by special equipment and erased by ultra-violet light
- Protocol** a set of rules which sender and receiver have to obey in order to communicate
- RAM** random access memory, often used for read–write memory; access to any location takes the same time irrespective of address
- Refresh** a process whereby a dynamic memory which loses information after a short time has its memory contents rewritten
- Register** a fast memory location, often in the CPU
- Relative address** an address relative to the current contents of the program counter
- Relocation** the process of moving a program from one part of memory to another
- RISC** reduced instruction set computer
- ROM** read-only memory
- Rotate** *see* Shift
- Routing** the process of directing a message through a network
- RS232** a standard for communication
- Sequential** performing in sequence, that is, one after another
- Serial** one after another, often with reference to communication
- Shift** to move sideways in a register
- Software** that part of the computer implemented by a program
- Stack** memory that is used in last-in–first-out fashion
- Stack pointer** register which points to the memory location currently acting as the top of the stack
- Static RAM** memory that needs no refreshing
- Status** condition
- Synchronization** the co-ordination of actions between two or more entities
- Transputer** a RISC machine designed for distributed computing designed by Inmos Ltd
- Tristate** three state, usually logic 0, logic 1 and high impedance
- VDU** visual display unit
- VLSI** very large scale integration
- Wide area network** a network over a large geographical area such as a country or a continent
- Word** a group of bits, usually the size of the data bus

Contents

| | | |
|----------|---|-----------|
| | Preface | ix |
| | Glossary | xi |
| 1 | Introduction | 1 |
| 1.1 | Approaches to computer architecture | 1 |
| 1.2 | Translation and interpretation | 4 |
| 1.3 | Languages and abstract machines | 5 |
| 1.4 | Sequentiality, concurrency and distribution | 6 |
| 1.5 | Summary | 7 |
| 2 | Introduction to digital logic | 8 |
| 2.1 | Introduction | 8 |
| 2.2 | Combinatorial and sequential logic | 8 |
| 2.3 | Basic logic gates | 9 |
| 2.4 | Example: a 1-bit half adder | 15 |
| 2.5 | Boolean algebra | 16 |
| 2.6 | NAND/NOR logic | 19 |
| 2.7 | Positive and negative logic | 22 |
| 2.8 | Logic implementation | 23 |
| 2.9 | Logic families | 27 |
| 2.10 | Summary | 37 |
| | Exercises | 38 |
| 3 | Combinatorial logic design | 39 |
| 3.1 | Introduction | 39 |
| 3.2 | Problem specification | 39 |
| 3.3 | Design example: a parity generator | 40 |
| 3.4 | Design example: a 7-segment decimal decoder | 43 |

vi Contents

| | | |
|------------|---|------------|
| 3.5 | Minimization of Boolean functions | 45 |
| 3.6 | Medium scale integrated functions | 50 |
| 3.7 | Programmable logic arrays | 57 |
| 3.8 | Read-only memories | 60 |
| 3.9 | Summary | 61 |
| | Exercises | 62 |
| | | |
| 4 | Sequential logic design | 63 |
| 4.1 | Introduction | 63 |
| 4.2 | Synchronous and asynchronous sequential circuits | 64 |
| 4.3 | State diagrams and state variables | 65 |
| 4.4 | Memory elements | 69 |
| 4.5 | Implementing sequential logic with D-type flip-flops | 74 |
| 4.6 | Implementing sequential logic circuits with read-only memories | 77 |
| 4.7 | Counter design | 82 |
| 4.8 | Design of a register bank | 86 |
| 4.9 | Summary | 89 |
| | Exercises | 91 |
| | | |
| 5 | The structure of a computer | 92 |
| 5.1 | The operation of a computer | 92 |
| 5.2 | The bus | 94 |
| 5.3 | The central processing unit | 95 |
| 5.4 | The arithmetic and logic unit | 96 |
| 5.5 | Examples of processors – the Intel 8085 and the Motorola 68000 | 102 |
| 5.6 | Input and output | 107 |
| 5.7 | Summary | 107 |
| | Exercises | 108 |
| | | |
| 6 | Memory systems | 109 |
| 6.1 | Memory | 109 |
| 6.2 | Architectural considerations and memory | 117 |
| 6.3 | Summary | 125 |
| | Exercises | 126 |
| | | |
| 7 | Input–output | 127 |
| 7.1 | Input–output interfaces | 127 |
| 7.2 | Controlling input–output devices | 136 |
| | Exercises | 142 |

| | | |
|-------------|---|------------|
| 8 | Control and microprogramming | 143 |
| 8.1 | The fetch cycle | 143 |
| 8.2 | The execute cycle | 146 |
| 8.3 | The control unit | 147 |
| 8.4 | Horizontal and vertical microcoding | 152 |
| 8.5 | Emulation | 153 |
| 8.6 | Summary | 154 |
| | Exercises | 154 |
| 9 | Design of a small computer system | 155 |
| 9.1 | Connecting the components together | 155 |
| 9.2 | A minimal Intel 8085 system | 158 |
| 9.3 | A Motorola 68000-based microcomputer | 161 |
| 9.4 | Summary | 164 |
| | Exercises | 165 |
| 10 | Data representation and manipulation | 166 |
| 10.1 | Introduction | 166 |
| 10.2 | Number systems | 166 |
| 10.3 | Data representation | 169 |
| 10.4 | Differing representations | 176 |
| 10.5 | Summary | 178 |
| | Exercises | 179 |
| 11 | Instruction sets and addressing modes | 180 |
| 11.1 | Instruction sets | 180 |
| 11.2 | The programmer's models | 183 |
| 11.3 | Instruction types | 186 |
| 11.4 | Operands | 195 |
| 11.5 | Addressing modes | 199 |
| 11.6 | Instruction encoding | 204 |
| 11.7 | Use of addressing modes | 206 |
| 11.8 | Examples to illustrate assembly code and addressing modes | 210 |
| 11.9 | Summary | 216 |
| | Exercises | 217 |
| 12 | Introduction to system software | 219 |
| 12.1 | Introduction | 219 |
| 12.2 | Assembly language and assemblers | 219 |
| 12.3 | Macros | 223 |
| 12.4 | Link editors | 223 |
| 12.5 | Loading | 227 |
| 12.6 | High-level languages | 227 |

viii Contents

| | | |
|-------------|---|------------|
| 12.7 | Documentation and debugging | 231 |
| 12.8 | Operating systems | 233 |
| 12.9 | Summary | 236 |
| | Exercises | 237 |
| 13 | Data communications | 239 |
| 13.1 | Communication | 239 |
| 13.2 | Addressing | 241 |
| 13.3 | Communication structure | 242 |
| 13.4 | Computer networks | 243 |
| 13.5 | Distributed computing | 253 |
| 13.6 | Summary | 254 |
| | Exercises | 255 |
| 14 | New directions in architecture | 256 |
| 14.1 | Introduction | 256 |
| 14.2 | Architectural considerations and processor design | 256 |
| 14.3 | RISC architectures | 257 |
| 14.4 | CISCs versus RISCs | 260 |
| 14.5 | The transputer | 262 |
| 14.6 | Summary | 265 |
| | Exercises | 266 |
| | Answers to selected exercises | 267 |
| | Further reading | 281 |
| | Index | 287 |

1

Introduction

1.1 Approaches to computer architecture

The way in which a human being understands a complex topic is known as ‘divide and conquer’. Any complex topic is decomposed into a set of less complex subparts hierarchically until the subparts are simple enough to be understood. Once these simple subparts are understood they are then composed into more complex units whose behaviour can be explained by the action of the subparts and their interaction. This process is continued hierarchically until the complex topic can be explained. This process is illustrated in Fig. 1.1.

Computer architecture is a complex topic and hence the way to understanding is to subdivide it into smaller topics. There are many different ways of subdivision and hence the many different ways of approaching the topic found in textbooks. Here two approaches to computer architecture are considered to show why a mixture of these two approaches is used in this book.

1.1.1 Layered approach

One approach to computer architecture is to consider a computer system as consisting of a set of layers of abstraction, a subset of which is used to implement a given computer system. For example, one or more of the lower layers in this model are concerned with the logic elements used to implement an architecture. One level could be solely concerned with the types of element used, for example **AND** and **OR** gates, and the design method for combining these logic elements. If the system was to be implemented in very large scale integration (VLSI) circuits then a level might be appropriate where the concerns are for physical parameters such as the width of tracks and type of technology. At a somewhat higher level is the machine level which is normally the lowest level available to the computer user. At this level

2 Introduction

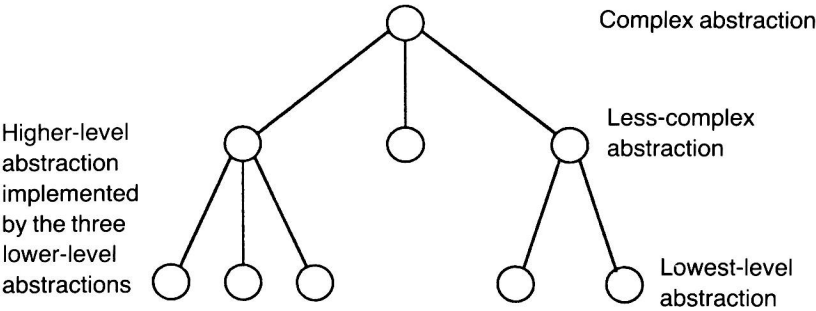


Fig. 1.1 A hierarchy of abstractions.

bit strings are interpreted as instructions and data. Levels above this are concerned with the operating system, high-level programming languages and applications. One possible hierarchy of levels of abstraction is given in Fig. 1.2. Higher levels have higher complexity than lower levels and each level relies on the level below it to implement its primitive operations. Each level has its own set of primitive components and its own set of design methods to interconnect these primitive components. In addition each level may have its own basic theory which underpins the construction method. For example, the logic level has propositional calculus and Boolean algebra as the basis of the synthesis and analysis of logic circuits. This layered approach is appealing

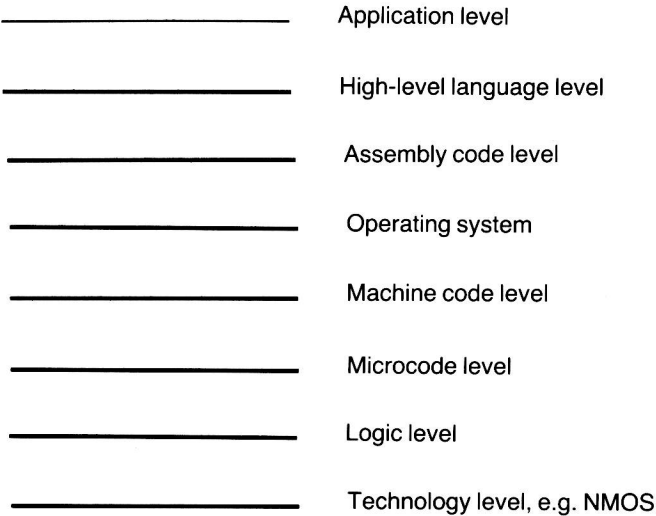


Fig. 1.2 Levels of abstraction.

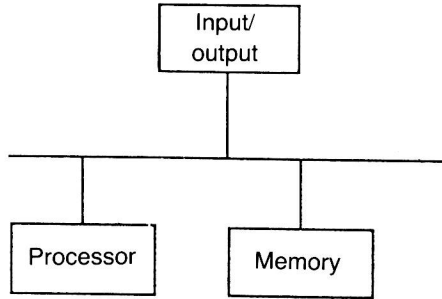


Fig. 1.3 Functional block diagram of computer.

since it relegates details of the synthesis and analysis of components to the appropriate level thus giving rise to a hierarchically structured approach.

1.1.2 Functional decomposition

Another approach to understanding computers is to adopt a functional decomposition, where the computer structure is split up into functional components and each functional block is considered separately. A typical computer could be split up into the basic functional blocks shown in Fig. 1.3, where, for example, memory is considered to be a functional block. Memory can be further subdivided into different types, such as random access memory (RAM) and read-only memory (ROM), and each of these types considered separately. In effect, this again gives a hierarchical structure, a tree, where each subtree consists of functionally related items. This decomposition is attractive since concerns about physically related components are grouped together and hence are easy to compare.

To illustrate the advantage of the functional decomposition, consider the topic of memory. In functional decomposition this would occupy a large subtree and, in terms of a book, would appear in a single chapter or group of chapters. In the layered approach different aspects of memory would be found in different layers. For example, the implementation of a memory would be found at one of the logic levels, whilst virtual memory would be considered at the operating system level. It would thus seem that the functional approach is the best way to understand computer architecture. However, there are many cases, especially at the lower levels of abstraction, where consideration of an abstraction level rather than separate functions becomes attractive. For example, the topic of logic design would appear as a single entity in the layered approach but would be scattered throughout all the components in the functional approach. Since there is an underlying rationale to logic design it makes sense to consider this as a topic in its own right. Since