The Architecture of High Performance Computers

Roland N. Ibbett

The Architecture of High Performance Computers

Roland N. Ibbett
*Reader in Computer Science,
University of Manchester*

M

First published 1982 by
THE MACMILLAN PRESS LTD
London and Basingstoke
Companies and representatives
throughout the world

Printed in Great Britain by
UNWIN BROTHERS LIMITED,
The Gresham Press, Old Woking, Surrey

ISBN 0 333 33229 6
0 333 33231 8 pbk

Preface

Computer architecture has always formed an essential part of all single honours degree courses taught in the Department of Computer Science at the University of Manchester, and the material in this book is largely based on those parts of the course which are concerned with the design of high performance uniprocessors. One of the principal aims of the course has always been to encourage students to discover the 'whys' as well as the 'hows' of computer architecture. For this reason I have tried to place the computers described here in their proper historical context, in order to show both why and how they were developed. Similarly I have not tried to impose a uniform nomenclature on to the material, but rather to use the terms which came naturally to the designers and which are to be found in the original documentation.

I am indebted to many people for advice and assistance in the preparation of this book. At Manchester numerous colleagues joined in discussions about the various topics considered here, and the computer maintenance team kept MU5 in good order while it processed the text. Elsewhere computer maintenance engineers at various sites willingly answered obscure questions about the machines in their charge, and staff at Cray Research and Control Data Corporation vetted parts of the manuscript. Particular mention should be made of Chuck Purcell of CDC for much useful information, and I would like to thank Professors T. Kilburn, D. B. G. Edwards, F. H. Sumner and D. Morris for their constant encouragement and advice over the years. Finally a word of appreciation to my small son James for his 'help' with the manuscript.

Roland N. Ibbett

June 1982

Contents

Preface

1 Introduction

1.1 Historical Developments 1
1.2 Techniques for Improving Performance 2
1.3 An Architectural Design Example 3

2 Instructions and Addresses

2.1 Three-address Systems - The CDC 6600 and 7600 7
2.2 Two-address Systems - The IBM System/360 and /370 10
2.3 One-address Systems 12
2.4 Zero-address Systems 15
2.5 The MU5 Instruction Set 17
2.6 Comparing Instruction Formats 22

3 Storage Hierarchies

3.1 Store Interleaving 26
3.2 The Atlas Paging System 29
3.3 IBM Cache Systems 33
3.4 The MU5 Name Store 37
3.5 Data Transfers in the MU5 Storage Hierarchy 44

4 Pipelines

4.1 The MU5 Primary Operand Unit Pipeline 49
4.2 Arithmetic Pipelines - The TI ASC 62
4.3 The IBM System/360 Model 91 Common Data Bus 67

5 Instruction Buffering

5.1 The IBM System/360 Model 195 Instruction Processor 72
5.2 Instruction Buffering in CDC Computers 77
5.3 The MU5 Instruction Buffer Unit 82
5.4 The CRAY-1 Instruction Buffers 87
5.5 Position of the Control Point 89

6 Parallel Functional Units

6.1 The CDC 6600 Central Processor 95
6.2 The CDC 7600 Central Processor 104
6.3 Performance 110
6.4 The CRAY-1 112

7 Vector Processors

7.1 Vector Facilities in MU5 126
7.2 String Operations in MU5 136
7.3 The CDC Star-100 142
7.4 The CDC CYBER 205 146
7.5 Future Developments 163

References 165

Index 170

1 Introduction

Computer architecture has been defined in a number of ways by different authors. Amdahl, Blaauw and Brooks [1], for example, the designers of the IBM System/360 architecture, used the term to 'describe the attributes of a system as seen by the programmer, i.e. the conceptual structure and functional behaviour, as distinct from the organisation of the data flow and controls, the logical design and the physical implementation'. Stone [2], on the other hand, states that 'the study of computer architecture is the study of the organisation and interconnection of components of computer systems'. The material presented here is better described by this wider definition, but is particularly concerned with ways in which the hardware of a computer can be organised so as to maximise performance, as measured by, for example, average instruction execution time. Thus the architect of a high performance system seeks techniques whereby judicious use of increased cost and complexity in the hardware will give a significant increase in overall system performance.

1.1 HISTORICAL DEVELOPMENTS

Designers of the earliest computers, such as the Manchester University/Ferranti Mark 1 (first produced commercially in 1951 [3]), were constrained by the available technology (valves and Williams Tube storage, for example, with their inherent problems of heat dissipation and component reliability) to build (logically) small and relatively simple systems. Even so, the introduction of B-lines and fast hardware multiplication in the Mark 1 were significant steps in the direction of cost-effective hardware enhancement of the basic design. At Manchester this trend was developed further in the Mercury computer, with the introduction of hardware to carry out floating-point addition and multiplication. This increased logical complexity was made possible by the use of semiconductor diodes and the availability of smaller and more reliable valves than those used in Mark 1, both of which helped to reduce power consumption (and hence heat dissipation) and increase overall reliability. A further increase in reliability was made in the commercial version of Mercury (first produced in 1957) by the use of the then newly developed ferrite core store.

The limitations on computer design imposed by the problems of heat dissipation and component reliability were decreased dramatically in the late 1950s and early 1960s by the commercial availability of transistors, and the first generation of 'supercomputers' (Atlas, Stretch, MULTICS and the CDC 6600, for example [4,5,6,7]) appeared. These machines incorporated features which have influenced the design of succeeding generations of computers (the paging/virtual memory system of Atlas and the multiple functional units of the 6600, for example), and also highlighted the need for sophisticated software, in the form of an operating system, intimately concerned with activities within the hardware from which the user (particularly in a multi-user environment) required protection, and vice versa! In this book we shall follow the developments of some of the ideas from these early supercomputers into present day computer designs.

1.2 TECHNIQUES FOR IMPROVING PERFORMANCE

Improvements in computer architecture arise from three principle factors

(1) technological improvements

(2) more effective use of existing technology

(3) improved software-hardware communication

Technological improvements include increases in the speed of logic circuits and storage devices, as well as increases in reliability. Thus the use of transistors in Atlas, for example, not only allowed individual circuits to operate faster than those in Mercury, but also allowed the use of parallel adders, which involved many more logic circuits but which produced results very much more quickly than the simple serial adders used in earlier machines. Corresponding changes can be seen today as the scale of integration of integrated circuits continues to increase, allowing ever more complex logical operations to be carried out within one silicon chip, and hence allowing greater complexity to be introduced into systems while keeping the chip speed and overall system reliability more or less constant.

Making more effective use of existing technology is the chief concern of computer architecture, and almost all of the techniques used can be classified under the headings of either storage hierarchies or concurrency. Storage devices range from being small and fast (but expensive) to being large and cheap (but slow); in a storage hierarchy several types, sizes and speed of storage device are combined together, using both hardware and software mechanisms, with the aim of presenting the user with the illusion that he has a fast, large (and cheap) store at his disposal.

Concurrency occurs in various forms and at various levels of system design. Low-level concurrency is typified by pipeline techniques, interleaved storage and parallel functional units, for example, all of which are largely invisible to software. High-level concurrency, on the other hand, typically involves the use of a number of processors connected together in some form of array or network, so as to act in parallel on a given computational task, and special software is generally required in order to organise the activities of these processors. In this book we shall be largely concerned with low-level concurrency techniques.

Communication between software and hardware takes place through the order code (or instruction set) of the computer, and the efficiency with which this takes place can significantly affect the overall performance of a computer system. One of the difficulties with some 'powerful' computers is that the 'power' is obtained through hardware features which can only be fully exploited either by hand coding or by the use of complex and time-consuming algorithms in compilers. This fact was recognised early in the Manchester University MU5 project, for example, where an instruction set was sought which would permit the generation of efficient object code by high-level language compilers. The design of this order code was therefore influenced not only by the anticipated organisation of the hardware, which was itself influenced by the available technology, but also by the nature of existing high-level languages. Thus the order code, hardware organisation, and available technology all interact together to produce the overall architecture of a given system. An example of such an interaction is given in the next section, based on some of the early design considerations for MU5 [8].

1.3 AN ARCHITECTURAL DESIGN EXAMPLE

At the time when the MU5 project was started (1966/7), the fastest production technology available for its construction was that being used by ICT (later to become ICL) for their 1906A computers. In this technology, MECL 2.5 small scale integrated circuits are mounted individually or in pairs on printed circuit modules, together with discrete load resistors, and up to 200 of these modules are interconnected via multi-layer platters. Platters are mounted in groups of six or nine within logic bays, with adjacent platters being joined by pressure connectors. Inter-group and inter-bay connections are via co-axial cables. The circuit delay for this technology is 2 ns, but the wiring delay of 2 ns/ft (through matched transmission line interconnections) must also be taken into account. An average connection between two integrated circuits mounted on modules on the same platter involves a 1" connection on each module and a 4" connection between modules, involving 6" in all and giving an extra 1 ns delay. Some additional delay also occurs because of the loading of an output circuit with following input circuits, and so for design purposes a figure of 5 ns was assumed for the delay between the inputs of successive gates.

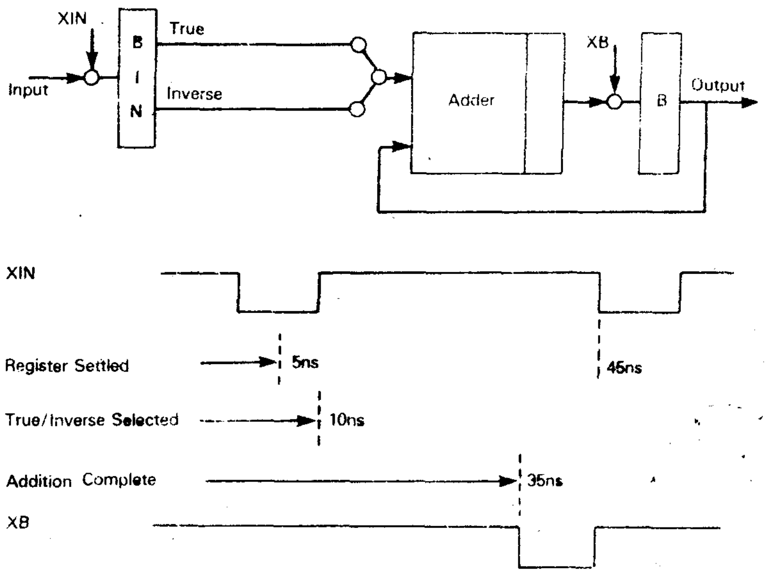


Figure 1.1 An Index Arithmetic Unit

A 32-bit fixed-point adder/subtracter constructed in this technology requires five gate delays through the adder, plus one gate delay to select the TRUE or INVERSE of one of the inputs to allow for subtraction, giving a total delay of 30 ns. This adder/subtracter can be incorporated into an index arithmetic unit as shown in figure 1.1. A 10 ns strobe XIN copies new data into register BIN, the output from which is steady after 5 ns. During the addition, information is strobed into a register which forms part of the adder, and the 10 ns strobe XB, which copies the result of the addition into the index register B, starts immediately after this internal adder strobe has finished. The earliest time at which the next XIN strobe can start is at the end of XB, so that the minimum time between successive add/subtract operations in this unit is 45 ns.

This time is very much less that the access time to the main store of MU5, a plated-wire store with a 260 ns cycle time. (A suitable replacement semiconductor store, using 16K MOS RAMs, would actually be even slower.) Thus, in the absence of some additional technique, there would be a severe mis-match between the operand accessing rate and the arithmetic execution rate for index arithmetic instructions. An examination of the operands used in high-level languages, and studies of programs run on Atlas, indicated that over a large range of programs, 80 per cent of all operand accesses were to named scalar variables, of which only a small number was in use at any one time. Thus a system which kept these operands in fast programmable registers would be able to achieve high performance, a technique commonly used in existing computers. However, this is exactly the sort of hardware feature which causes compiler complexity, and which the designers of MU5 sought to avoid.

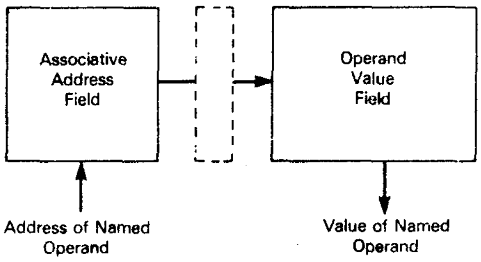


Figure 1.2 The MU5 Name Store

The alternative solution adopted in MU5 involves the identification within each instruction of the kind of operand involved, and the inclusion in the hardware of an associatively addressed buffer store for named variables, the Name Store. This store has to operate in conjunction with the main store of the processor, thus immediately introducing the need for a storage hierarchy in the system. Having observed this implication, further discussion of storage hierarchies will be left until Chapter 3; there are additional implications for the system architecture to be considered here, based on the timing of Name Store operations.

The Name Store consists of two parts, as shown in figure 1.2; the associative address field and the operand value field. During the execution of an instruction involving a named variable, the address of the variable is presented to the associative field of the Name Store, and if a match is found in one of its 32 associative registers, the value of the variable can be read from the corresponding register in the Name Store value field. The time required for association is 25 ns, and similarly for reading. Thus, in order for the index-arithmetic unit to operate at its maximum rate, the association time, reading time and addition time for successive instructions must all be overlapped (by introducing buffer registers such as that shown dotted in figure 1.2). In making provision for this overlap, however, another architectural feature has been introduced - the organisation of the processor as a pipeline. Further discussion of pipelines will be left until Chapter 4, but it should now be clear that there is considerable interaction between the various facets of the architecture of a given computer system.

2 Instructions and Addresses

An important characteristic of the architecture of a computer is the number of addresses contained in its instruction format. Arithmetic operations generally require two input operands and produce one result, so that a three-address instruction format would seem natural. However, there are arguments against this arrangement, and decisions about the actual number of addresses to be contained within one instruction are generally based on the intuitive feelings of the designer(s) in relation to economic considerations, the expected nature of the implementation, and the type of operand address and its size. An important distinction exists between register addresses and store addresses, for example; if the instruction for a particular computer contains only register addresses, so that its main store is addressed indirectly through some of these registers, then up to three addresses can be accommodated in one instruction. On the other hand, where full store addresses are used, multiple-address instructions are generally regarded as prohibitively expensive both in terms of machine complexity and in terms of the static and dynamic code requirements. Thus one store address per instruction is usually the limit (in which case arithmetic operations are performed between the content of the store location and the content of an implicit accumulator), although some computers have variable-sized instructions and allow up to two full store addresses in a long instruction format. In this chapter we shall introduce examples of computer systems which have three, two, one and zero-address instruction formats, and discuss the relationships which exist between each of these arrangements and the corresponding hardware organisation.

2.1 THREE-ADDRESS SYSTEMS - THE CDC 6600 AND 7600

The Control Data Corporation 6600 computer first appeared in the early 1960s, and was superseded in the late 1960s by the 7600 system (now renamed CYBER 70 Model 76). The latter is machine code compatible upward from the former, so that the basic instruction formats of the two systems are identical, but the 7600 is about four times faster than the 6600. The 6600 remains an important system for students of computer architecture, however, since it has been particularly well documented [7], and an understanding of its organisation and operation provides a proper background not only for an appreciation of the design of the 7600 system, but also that of the CRAY-1, which can be seen as a logical extension of the 6600/7600 concepts from scalar to vector operation. The design of all these machines will be discussed in more detail in Chapter 6.

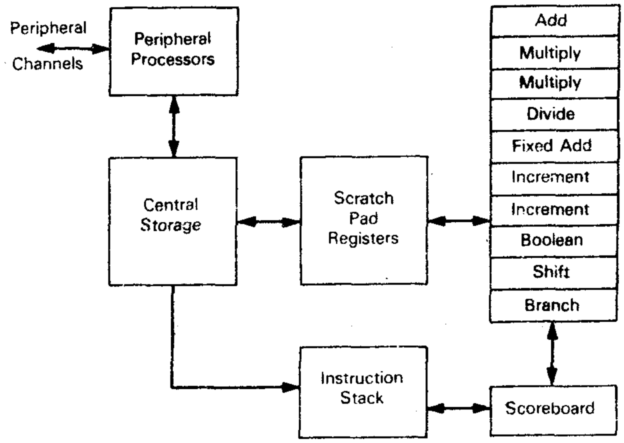
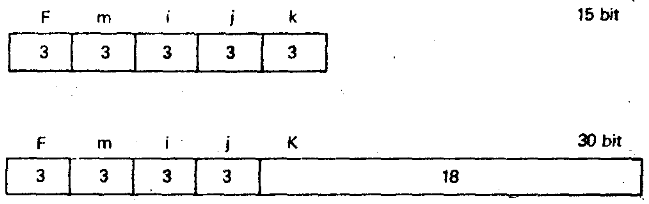


Figure 2.1 CDC 6600 Processor Organisation

The 6600 was designed to solve problems substantially beyond contemporary computer capability and, in order to achieve this end, a high degree of functional parallelism was introduced into the design of the central processor. An instruction set and processor organisation which could exploit this parallelism, while at the same time maintaining the illusion, at least, of strict serial execution of instructions, was therefore required. A three-address instruction format provides this possibility, since successive instructions can refer to totally independent input and result operands. This would be quite impossible with a one-address instruction format, for example, where one of the inputs for an arithmetic operation is normally taken from, and the result returned to, a single implicit accumulator. Dependencies between instructions can still occur in a three-address system, of course; for example, where one instruction requires as its input the result of an immediately preceding instruction, and the hardware must ensure that these are strictly maintained. This would be difficult if full store addresses were involved, but the use of three full store addresses would, in any case, have made 6600 instructions prohibitively long. There were, in addition, strong arguments in favour of having a 'scratch-pad' of fast registers in the 6600 which could match the operand processing rate of the functional units.



F denotes the major class of function
m denotes a mode within the functional unit
i identifies one of 8 registers within X, B or A
j identifies one of 8 registers within X, B or A
k identifies one of 8 registers within X B or A
K 18-bit immediate field used as a constant or Branch destination

Figure 2.2 CDC 6600 Instruction Formats

Thus the overall design of the 6600 processor is as shown in figure 2.1, and the instruction formats are as shown in figure 2.2. There are three groups of scratch-pad registers, eight 60-bit operand registers (X), eight 18-bit address registers (A), and eight 18-bit index registers (B). 15-bit computational instructions take operands from the two X registers identified by j and k, and return a result to the X register identified by i. Operands are transferred between X registers and Central Storage by instructions which load an address into registers A1-A5 (thus causing the corresponding operand to be read from that address in store and loaded into X1-X5), or into registers A6 or A7 (thus causing the content of X6 or X7 to be transferred to that address in store). The contents of an A register can also be indexed by the contents of a selected B register, and the B registers can also be used to hold fixed-point integers, floating-point exponents, shift counts, etc. The 30-bit instruction format (identified by particular combinations of F and m, as are the register groups) is used where a literal ('immediate') operand is to be used in an operation involving an A or B register, or in control transfer ('Branch') instructions, where the value in the K field is used as the jump-to, or destination, address.

The issuing of instructions to the functional units and the subsequent updating of result registers is controlled by the Scoreboard (section 6.1), which takes instructions in sequence from the Instruction Stack (section 5.2.1). This in turn receives instructions from Central Storage. The order code of the central processor does not include any peripheral handling instructions, since all peripheral operations are independently controlled by the ten Peripheral Processors, which organise data transfers between Central Storage and input/output and bulk storage devices attached to the peripheral channels. This relieves the central processor of the burden of dealing with input/output, a further contribution towards high performance.

2.2 TWO-ADDRESS SYSTEMS - THE IBM SYSTEM/360 AND /370

The design objectives for the IBM System/360 [1,9] (introduced in 1964) were very different from those for the CDC 6600. Here the intention was to produce a line of processors (initially six) with compatible order codes, but a performance range factor of 50. Furthermore, whereas the CDC 6600 was intended specifically for scientific and technological work, System/360 machines were required to be configurable for a variety of tasks covering both scientific and data processing applications. The former are dominated by floating-point operations, where fast access to intermediate values requires the provision of one or more registers close to the arithmetic unit(s) in the central processor. Data processing applications, on the other hand, frequently involve the movement (and fairly simple manipulation) of long strings of data, and direct two-address storage to storage operations are much more appropriate than operations involving three store addresses or operations involving the use of intermediate values held in registers. Furthermore, in machines at the low performance end of the range, logical registers are normally implemented within main storage, and the extra store accesses required for these registers would actually slow down this type of operation.

Store addressing is itself a major problem in the design of a compatible range of computers; to quote from IBM, in turn quoting from DEC [10,11], 'There is only one mistake...that is difficult to recover from - not providing enough address bits...'. In the design of the IBM System/360 large models were seen to require storage capacities in the millions of characters, whereas small models required short addresses in order to minimise code space requirements and instruction access time. This problem was overcome by the use of index registers to supply base addresses covering the entire 24-bit address space, and small displacement addresses within instructions. This can be seen in figure 2.3, which shows the five basic instruction formats: register to register operations (RR), register to indexed storage operations (RX), register to storage operations (RS), storage and immediate operand operations (SI), and storage to storage operations (SS). All operand addresses consist of a 4-bit B field