The MU5 Computer System

Derrick Morris
Roland N. Ibbett

*Department of Computer Science,
University of Manchester*

M

First published 1979 by
THE MACMILLAN PRESS LTD
London and Basingstoke
Associated companies in Delhi Dublin
Hong Kong Johannesburg Lagos Melbourne
New York Singapore and Tokyo

Printed in Great Britain by Bell and Bain Ltd.,
Glasgow

British Library Cataloguing in Publication Data

Morris, Derrick
The MU5 computer system.—(Macmillan computer science series).
1. MU5 (Computer system)
I. Title II. Ibbett, Roland N
001.6'4044 QA76.5

ISBN 0-333-25749-9
ISBN 0-333-25750-2 Pbk

The MU5 Computer System

This book is dedicated to all those
who contributed to the MU5 Project

Contents

1 Introduction

MU5 is the fifth computer system to be designed and built at the University of Manchester. The development of the systems leading up to MU5 is described by Lavington [1]. This book is concerned with the design, implementation and performance of MU5. It covers both hardware and software as these have been designed as an integrated system by a closely knit group of 'Engineers' and 'Programmers'. No attempt is made to assign individual credit.

A precise starting date for the project is difficult to pinpoint. Many of the ideas it embodies grew out of the previous Atlas Project. The records show that talks with ICT (later to become ICL) aimed at obtaining their assistance and support began in 1966. An application for a research grant was submitted to the Science Research Council in mid-1967, and a sum of £630 446 spread over 5 years became available in January 1968. In 1968 an outline proposal for the system was presented at the IFIP 68 conference [2]. The feasibility of constructing a big computer system for the amount of the grant relied upon the availability of production facilities, at works cost price, at the nearby ICT West Gorton Works. Even so, the finance was a limiting factor, and it was accepted that the hardware produced would only be a small version of the potentially large system that was to be designed.

The level of staffing may be of some interest. In 1968 a group of 20 people was involved in the design, made up as follows

11	Department of Computer Science staff
5	Seconded ICT staff
4	SRC Supported staff

The peak level of staffing was in 1971 when the numbers, including research students, rose to 60. This fell during the commissioning period to 40. In the evaluation stage, from 1973, only 25 people were involved.

Motivation for the project was twofold. First there was the desire to continue the tradition of designing and building advanced systems, pioneering ideas which could be exploited by the computer industry. In addition there was a requirement for a system to support the research school of the Department of Computer Science. Significant expansion of this research school was planned beginning with the first year of the Computer Science graduates in 1968. Experience had shown that research into hardware/system software could not be carried out on a computing service machine. It is excluded by both the nature of the work and by the excessive computing requirements of the simulation studies, and the automation of hardware and software design which dominate the research.

The design objectives are best covered by the following quotations from the grant application to the Science Research Council dated May 1967. It was felt that a computer should be provided 'off the shelf' to initiate the project.

> 'The computer required is an ICT 1905E specially fitted with a 750 ns store ... The 1905E will be transformed into a multi (initially 2) computer system by the addition of a completely new high-performance computer with a target throughput of 20 times that of Atlas ... It will be constructed by ICT (their agreement has been obtained) and will be charged at works cost price ... The 1905E, with the proposed modifications in view, will provide a vehicle which permits an immediate start on software developments aimed at the full system programs of the multi-computer system. The system programs will be written in a modular way to facilitate changes and extensions when these are required as the hardware develops.'

Thus the emphasis was on a multi-computer system containing at least one new high-performance machine having a target throughput 20 times that of Atlas.

'This factor will be achieved as follows

(1) Integrated circuits and interconnection techniques will give a basic computing speed of seven times Atlas.

(2) A 250 ns core store will be used, this is eight times the speed of the Atlas store.

(3) The design will include

Fast operand registers
Register to register arithmetic
Multiple arithmetic units

Items (1) to (3) will give a factor of about ten, indeed the time for the inner loop of a scalar product is expected to be 1 μs as compared with 12 μs on Atlas.

(4) An instruction set will be provided which will permit the generation of more efficient object code by the compilers. Particular attention will be given to the techniques for computing the addresses of array elements. Array bound checking will be provided as a hardware feature.

(5) The efficiency of the Atlas supervisor is approximately 60%. The provision of special hardware and the information obtained from a detailed study of the Atlas system over the past two years will permit this efficiency to be significantly increased.

Items (4) and (5) will give at least a further factor of two.'

Clearly, performance was to be measured in terms of system throughput rather than raw machine speed. Significant factors were to be sought from optimising the hardware to meet the software requirements and an available production technology was to be used. Indeed the chosen technology was that to be used in the construction of ICT 1906As. However, it was anticipated that associative storage would play a significant role in the system design [3] and that suitable integrated circuit elements would be developed for this purpose.

On the hardware side this book is mainly concerned with the design and implementation of the MU5 processor. However, the design was intended for a range of machines and the actual processor built is one example, which is towards the top of the range, with a scientific bias. The range was intended to go from machines of about PDP-11 cost to a multi-computer system incorporating several MU5s with differing biases at the top of the range. Thus the MU5 built has an 'Exchange' to which reference is made in several places. This is the hardware unit which connects the various computers of the total system. The software description takes into account both the range and the multi-computer aspects.

Although the design team had set themselves the task of designing a range of machines which could be marketed, it had no formal commitment to the computer industry. The ICT involvement was through the secondment of individual members of ICT to the University Team. Nevertheless, it could hardly be fortuitous that the design of the ICL 2900 is so similar to MU5 that in 1969 the possibility of MU5 being marketed as an early member of the 2900 range was seriously considered. After a three-month 'convergence' exercise in early 1970, when the designs were drawn even closer together, the idea was abandoned because of ICL's fear that the cost of maintaining compatibility would outweigh any advantage of early availability. During this period some changes were made to the detailed design of MU5 in the name of compromise, not all of which have been beneficial. Although there has been no attempt to maintain compatibility since that time the MU5 operating system and compilers can be transferred to 2900 with ease. The converse is not true.

Software plans for the project were geared as much to the MU5 multi-computer system and the range concept as to the MU5 processor.

> 'The initial operating system will be for a single computer system but it will be extended to accommodate additional computers whose structures and order codes are different from those of the 1905E. It will be modular and easily changed in order to accommodate future hardware and software developments. The detailed design of the operating system has not been completed. However, it will have the following features
>
> (1) Some form of file storage and on-line access
>
> (2) Job queueing and scheduling for base load jobs
>
> (3) Priority routes through the system for urgent jobs
>
> (4) The basic supervisor will be kept to a minimum and most of the operating system facilities will run as non-privileged programs.'

Compilers were to be produced using ideas developed from the Atlas Compiler Compiler. The emphasis was to be on efficiency, compactness and machine independence.

These initial objectives remained as the project developed and the reader will judge the extent to which they have been achieved.

2 The Architecture of the MU5 Processor

The design of the MU5 processor was approached through its order code, this being the natural interface between software requirements and hardware organisation. Full interplay between the two aspects was considered vital throughout the design. Efficient processing of high-level language programs was the prime target. In 'number crunching' applications, this meant a fast execution rate for the high-level language programs. However, the system envisaged would be interactive, and to combat the system overheads this entails, it was considered important to produce small compilers and compiled programs. Thus, an order code was sought which satisfied the following conditions

(1) Generation of efficient code by compilers must be easy

(2) Programs must be compact

(3) The instruction set must allow a pipeline organisation of the CPU leading to a fast execution rate

(4) Information on the nature of operands (scalar or array element, for example) should be available to allow optimal buffering of operands.

In this chapter the order code of MU5 is examined from the point of view of its use and implementation. However, a large part of the order code of such a highly structured system is concerned with address generation, and before discussing this it is appropriate to establish the policy relating to address validation, the mechanism which protects one user from another.

2.1 INTERPRETATION OF ADDRESSES

The most far reaching decision in the design of an order code is whether the addresses it generates are real or virtual. If real addresses are generated they will be used directly to access the store. Therefore the address must have been previously validated, as it was being computed, say. The alternative offered by the IBM system, of tagging store blocks to indicate ownership, was not considered flexible enough for a multi-access system in which the core allocation would be constantly changing. In effect, the real address based systems considered require that all address words contain an origin and a limit, and hence relate to bounded contiguous sections of store. Also the CPU must know which words in the store are address words. It then checks that each operand address is calculated from an address word, and that it falls within the specified limits. Since all address words are known to the system, out-of-use information can be moved out of main store until next required, provided the address words involved are appropriately marked and updated. A classic example of this type of machine is the Basic Language Machine [4], although it has never progressed beyond the prototype stage. Also the Burroughs machines since the 5000 series have had a similar type of controlled address formation, and currently the 'capability machines' promote a similar idea. Alternatively, if the order code generates virtual addresses, then special hardware is needed between the CPU and the store to validate the address and translate it into a real address. Sometimes the address will relate to information not in the main store, and the hardware will detect this and initiate its transfer, usually with software assistance. This special hardware may be a single datum and limit as for example in ICL's 1900, or multiple datum and limit as for example in the PDP-11, or a paging system as in Atlas.

The real address based systems have several attractions. Perhaps foremost from the performance point of view is the fact that the address generated by the CPU can be presented directly to the store, thus avoiding the time delay inherent in paging systems. Also the units of information delimited by address words, which would be the units the system might automatically move from one level of store to another, would be complete logical entities (procedures or arrays, for example). It can be argued that this is more efficient than moving fixed-size pages which represent arbitrary fragments of a program and its workspace [5]. The other side of this argument is that the problems of allocating and retrieving store in variable sized areas lead to some store not being utilised, for example because the empty areas may be too small. This has to be offset against the paging problem in which, even when all pages are in use, some will be partially occupied by unwanted information. It is by no means clear where the balance lies.

Two additional considerations led to the choice of virtual addressing for MU5. First it was felt that the most significant task of the operating system was store management, the dominant part of which is concerned with the automatic movement of information between levels of store. Such movement requires that the real addresses of the information moved be changed. If these real addresses are allowed to scatter through each program's private store, this task becomes complex. For example, the address words that require changing because of movement of information between levels of store are themselves subject to moving. Also, the same address might appear in several places. It was felt to be a cleaner solution to hold all information relating to the way a program maps into real store in a separate data structure outside the program and entirely under operating system control.

The second consideration was that a program should not be constrained in the way it might build a data structure within its own workspace by the mechanism for address validation. Close examination of, for example, the system proposed by Iliffe [4] will reveal the awkward constraint that arrays must be homogeneous.

Once the decision to base the system on virtual addressing had been taken, it was not difficult to reject the single datum and limit approach. Although such a system leads to an extremely simple organisation within the operating system, the entire program must be placed in a contiguous area of store each time the CPU is assigned to it. In contrast, one of the main attractions of Atlas had been the large virtual address space available to every user job, which could be used sparsely without significant penalty. For example, the compilers and operating system used the top half of the virtual store, user code was compiled into the bottom quarter, and the next quarter was used for the stack work space. Other smaller entities such as input and output buffers were fitted into the gaps in between. From this informal partitioning of the store on Atlas grew the idea of formalising the division into a segmented virtual store, which is also exploited in the Multics system [6].

In MU5 the final decision was to use a large virtual address, and to subdivide it into a segment number and a displacement within the segment. It was anticipated that large systems would be paged, but that small ones might employ multiple datum and limit registers (one per segment).

2.2 THE ORDER CODE

2.2.1 Choice of Instruction Format

The first step in choosing an instruction format is to decide how many operand addresses an instruction will have. Obviously this is influenced by the size of an operand address. If the instruction contains only register addresses, so that main store is addressed indirectly through registers, several addresses can be accommodated. If full store addresses are to be used, then one is usually the limit, although some machines, for example the PDP-11, have variable sized instructions and allow up to two full store addresses to occur in the long instructions.

It was decided from the start of the MU5 design that in order to comply with condition (1) above, there would be an address form corresponding to each form of operand permitted by high-level languages. Furthermore it was felt that to have more than one such operand per instruction would conflict with conditions (2) and (3). Only one facet of high-level language programs caused concern on account of this decision. This was the known high rate of usage of simple instructions such as

I := I + 1

Clearly, three instructions would be required to implement this in a one address code. However, the high execution rate expected of these simple orders and the possibility of them overlapping with adjacent orders was thought to compensate. For other reasons the possibility of using addressable fast registers for frequently used operands or addresses was rejected in favour of hardware optimisation using associative memory. First there was the desire to simplify the software by eliminating the need for optimising compilers. Equally important though was the desire to have fast procedure entry and exit, unfettered by the need to dump and restore registers. Thus through general design considerations the choice of format was restricted to the zero address (stacking machine) type or some form of one address code.

From a compiler point of view the stacking machine is attractive. The simple algorithm for translating from Algol to Reverse Polish (and hence to stacking machine code) which forms the basis of the 'Burroughs Compilogram' is a convincing demonstration of this. Its simplicity stems from the fact that operands carry directly over to Reverse Polish without any relative change of position and a simple push down stack is all that is required to sort the operators into correct sequence. Consider for example

(A + B) * ((C + D) / (E + F))

which in Reverse Polish becomes

AB + CD + EF + / *

There were two arguments which steered the MU5 design away from the stacking machine form. The first is related to efficiency of hand-coding, which is something of a paradox since MU5 is a high-level language machine. However, observations on Atlas indicated that while high-level language programs were running, the CPU typically spent half its time executing in a small set of library procedures concerned with I/O handling, mathematical functions, etc. This basic library would be hand-coded. Thus from the performance point of view, this small amount of hand-coded software was just as important as all the compiler generated code. Unfortunately most of the hand-coded sequences worked out worse in stacking machine code than in single address code. This was because the main calculation, the address calculations and the control counting, tended to interfere with each other on the stack. The problems are illustrated by the following example of a simple move sequence, although either machine could have a single function for this purpose.

```
Single Address Code              Stacking Machine Code

LOAD MODIFIER                    STACK MODIFIER
X: ACC = SOURCE[MODIFIER]        X: DUPLICATE
ACC => DEST[MODIFIER]            DUPLICATE
INC AND TEST MODIFIER            STACK SOURCE[TOP OF STACK]
IF NOT END BRANCH X              SWOP
                                 STORE DEST[TOP OF STACK]
                                 STACK 1
                                 SUBTRACT
                                 IF NOT END BRANCH X
```

The point being made is that a single stack is under pressure when it has to support all the functions involved in counting, address calculation and main calculation. In any given context, detailed changes to the specification of instructions would ease the problem, but only at the expense of it recurring in a different context. A machine with several stacks would have worked better, for example

a control stack
an index stack
an address stack
the main stack

This sort of arrangement would also fit the pipeline requirement better since the stacks could be distributed along the pipeline.

The second argument against the stacking machine would apply equally to a multi-stack organisation. Consider the example

```
A := B + C
```

For the two types of instruction format under consideration it would be coded as follows

```
ACC = B          STACK B
ACC + C          STACK C
                 ADD
ACC => A         STORE A
```

If the operands normally come from main store the execution times of each of the above sequences would be about the same, since they will be controlled by the access times for A, B and C. However, if an operand buffering scheme is utilised, giving a high hit-rate (say > 90%) for operands such as A, B and C, the access time to the stack becomes important. On MU5 the stack and the operand buffers would be the same speed, and the above example would have caused six stack accesses in addition to the three operand accesses. Some, but not all, of the accesses could have been overlapped.

The instruction format eventually chosen for MU5 represented a merger of single address and stacking machine concepts. All the arithmetic and logical functions take one operand from an accumulator and the other operand is specified in the instruction address. Thus a sequence such as

```
ACC = B
ACC + C
ACC => A
```

typifies the style of simple calculations. However, there is a stack, and a variant of the load order (*=) causes the accumulator to be stacked before being re-loaded. Also a special address form exists (STACK) which unstacks the last stacked quantity. Thus, the above example could be written in MU5 code in a form approximating to Reverse Polish, as follows

```
ACC = B
ACC *= C
ACC + STACK
ACC => A
```

A more realistic use of the stack is in conjunction with parenthesised subexpressions. For example, the expression

```
(A + B) * ((C + D) / (E + F))
```