ISSN 0254-2757

QI QUÆSTIONES INFORMATICÆ

Volume 5 • Number 2 October 1987

An official publication of the Computer Society of South Africa and of the South African Institute of Computer Scientists

'n Amptelike tydskrif van die Rekenaarvereeneging van Suid-Afrika en van die Suid-Afrikaanse Instituut van Rekenaarwetenskaplikes

QUÆSTIONES INFORMATICÆ

An official publication of the Computer Society of South Africa and of the South African Institute of Computer Scientists

'n Amptelike tydskrif van die Rekenaarvereniging van Suid-Africa en van die Suid-Afrikaanse Instituut van Rekenaarwetenskaplikes

Subscriptions

Annual subscription are as follows:

	SA	US	UK
Individuals	R10	\$7	£5
Institutions	R15	\$14	£10

Computer Society of South Africa
Box 1714 Halfway House

Quæstiones Informaticæ is prepared by the Computer Science Department of the University of the Witwatersrand and printed by Printed Matter, for the Computer Society of South Africa and the South African Institute of Computer Scientists.

HANDS-ON MICROPROGRAMMING FOR COMPUTER SCIENCE STUDENTS

Peter G Clayton
Department of Computer Science
Rhodes University

ABSTRACT

This paper advocates hands-on training on microprogrammable hardware for Computer Science students, and suggests an appropriate microprogram development approach for this purpose. The particular tool discussed is table driven, and allows for the downloading of microinstructions onto a hardware prototyping board. The system has been successfully implemented on an IBM PC microcomputer.
KEY WORDS Microprogram Education

1. INTRODUCTION

The more fundamental levels of Computer Science overlap naturally with Electronic Engineering, and are included in most recommendations on Computer Science curricula[1,3]. Microprogramming is one of the few specialist options at this level which, if given more than superficial coverage, gives the Computer Science student some contact with hardware design.

It is desirable from an educational point of view that students be able to verify their microprogram designs. Walker[5] has argued that the success of teaching in this subject area is heavily dependent on the nature of the teaching aids available to students for the execution of microprogramming exercises. Another key aspect in the successful application of a practical course is the degree of availability of equipment to the student.

2. SIMULATORS AND HARDWARE-BASED INSTRUCTION TOOLS

There are considerably fewer obstacles to the learning process using interactive simulation techniques than using hardware-based teaching aids. Most Computer Science students are unacquainted with hardware debugging techniques. This situation is aggravated by the primitive nature of the man/machine interface of hardware development kits, and the fact that the successful implementation of a hardware design is time consuming. Microprogram development hardware is additionally handicapped as an instructional aid by its high purchase price and therefore limited availability to large classes.

Simulators can overcome these encumbrances from a teaching point of view, and have been the popular means[4] for allowing large numbers of students to actually experience microprogramming, without incurring a large hardware expense or producing a contention problem. Simulators give the benefit of source statement editing, compilation and interpretation in a form familiar to computer users. Above all, trace statements can be included in a hardware simulation without affecting its behaviour (since it cannot be regarded as running in real time), and can be the basis for setting up a powerful debugging facility. This facility should allow the user to control the rate of execution of the microprogram, and to interrupt execution to alter or monitor the contents of registers, flipflops and memory locations.

From an educational perspective, simulators are restricted in two respects. Firstly, the hardware configuration which they simulate is inflexible, and is often a highly simplified processor. They can naturally be written to include a selection of hardware configurations, but the user remains restricted by the author's imagination. Secondly, the Computer Science student does not see any actual microprogrammable hardware.

At Rhodes University, an attempt has been made to exploit the benefits of both forms of implementation by making use of a simulated controller sequencer to control a hardware processor. The disadvantage of this approach is that the processor is never able to run in real time. A user friendly system with good monitoring and debugging facilities is the compensatory tutorial advantage. The student is given the opportunity of some hands-on hardware experience while being reasonably sheltered from the problems of timing and hardware debugging. A suggested exercise is to present the student with a working processor and require him to make minor modifications to it. Students are encouraged to test their microprograms with the aid of a

quæstiones informaticæ, 5, 2, pp 63-67, issn 0254-2757

simulation of the processor before attempting to control the hardware.

It was felt that an improved development system for instructional purposes should provide experience not only in microprogram design, but also in microinstruction design. Consequently, a definition level below the microprogram specification stage was included in the Rhodes University model, enabling the user to design the microinstruction format. This facility allows flexibility in the configuration of the processor hardware.

3. A BRIEF OVERVIEW OF THE RHODES MICROPROGRAM SYNTHESISE

The system consists of two definition programs and a simulated controller sequencer, related as in Figure 1. It has been written for use with any processor which can be controlled by the chosen controller sequencer. The current design is based on the AMD 2910[2] architecture.

The controller sequencer has both a hardware and a software interface. This allows users of the system to test experimental processor designs either by writing a simulation of their design, or by building a prototype of the hardware.

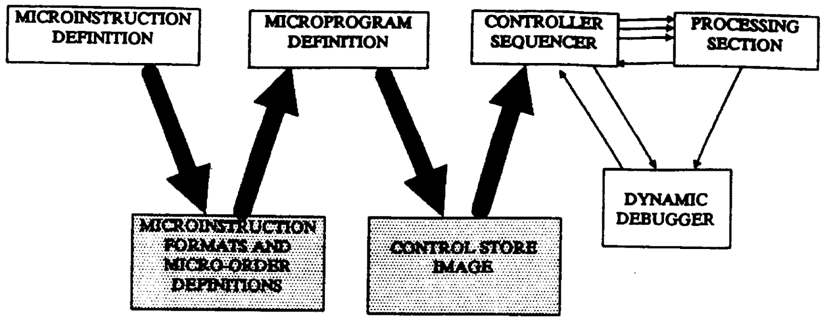

figure 1

The Microprogram Synthesiser

3.1 Microinstruction Definition

This part of the system allows the user flexibility in deciding on the format of his microinstructions, and the degree to which they are encoded. The particular choice of controller sequencer device places obvious constraints on the freedom of design in this area. At the same time, the user is able to specify a symbolic language for the micro-assembler.

The microinstruction definition program allows the user to define the formats of all possible micro-orders (fields within the microinstruction), in a flexible notation which caters for fixed value fields, variable fields and "don't care" fields, which may be overlayed with other micro-orders at a later stage. Specific micro-order bit patterns may then be defined which conform to these formats. This two-phase arrangement spares the user a significant amount of typing and reduces the number of possible errors. Each micro-order definition must include a name, or "mnemonic", by which the operation will be known. The set of mnemonics provide the basis for a user defined symbolic language which is used in the next stage of the system to write the microprogram.

As an example of the notation, consider some of the ALU functions available on the AMD 2901[2] bit-slice chip, shown in figure 2.

	I_5	I_4	I_3	
Pins:	**27**	**28**	**26**	**operation**
	L	L	L	ADD
	L	H	H	logical OR
	H	L	L	logical AND

figure 2

2901 ALU Functions

Having chosen a microinstruction width of 56 bits, a designer might decide to use bits 3 to 5 as a micro-order field to control the ALU function. A summary of the type of details which the user is required to furnish in order to record this micro-order follows:

(a) Define the format:

    Format number : 1
    Description : ALU function
    Format : 3x 3a 50x (i.e. xxxaaaxxxx ... x)

The letter "a" denotes the positions of the the active bits for this format, which need not necessarily be adjacent. The letter "x" is used for "don't care" fields.

(b) Define the micro-orders which use the format description of (a) above.

To define the ADD operation:

    Micro-order uses format number : 1
    Mnemonic to represent value : ADD
    Value of active field : 000

The other ALU operations are defined in the same way:

    Use format number : 1
    Mnemonic : OR
    Value : 011
    Use format number : 1
    Mnemonic : AND
    Value : 100
    .
    .

(Both definitions are required unless definition (a) has already been entered to describe a previous micro-order of the same format, in which case only (b) is necessary).

In this way, a table is constructed in which each entry records a micro-order mnemonic, its associated binary value, and its precise position in the microinstruction. Several micro-orders are combined in later stages to construct a microinstruction. Built-in checks guard against such obvious errors as conflicting or overlapping micro-order fields.

For example, if two more micro-order fields were defined as

(a) Format number : 2
    Description : ALU input
    Format : 3a 53x
    Format number : 3
    Description : A latch select
    Format : 6x 4a 46x

(b) Use format number : 2
    Mnemonic : DA
    Value : 101
    Use format number : 3
    Mnemonic : Areg
    Value : n (Areg will take a numeric parameter)

it would be possible to specify that a microinstruction should add the data from the D-bus and the A-latch, and that the A-latch should select register 12, using the notation:

    ADD & DA & Areg #1100

Figure 3 shows the binary pattern which would be constructed.

Thus it can be seen that the microinstruction definition program essentially provides the information for a table-driven micro-assembler for a particular hardware configuration.

3.2 Microprogram Definition

An interactive micro-assembler/editor allows the user to encode a microprogram in a symbolic language and translate this representation of the microprogram into an absolute representation for loading into control storage. This part of the system is driven by the tables set up in the microinstruction definition stage, and enables the user to associate conventional machine language mnemonics and symbolic labels with microprogram segments. Naturally, the design of the computer should include an instruction decoder to map macro-instruction (conventional machine level) opcodes to the addresses of the corresponding sequences of microinstructions.

1. ADD fills in the field defined by format 1 with the values associated with this mnemonic.

0	1	2	3	4	5	6	7	8	9	10	11	55
			0	**0**	**0**							
x	x	x	a	a	a	x	x	x	x	x	x	x

2. DA uses format 2.

0	1	2	3	4	5	6	7	8	9	10	11	55
1	**0**	**1**	0	0	0							
a	a	a	x	x	x	x	x	x	x	x	x	x

3. Areg #1100 fills in the field defined by format 3 with the value of the associated parameter of Areg.

0	1	2	3	4	5	6	7	8	9	10	11	55
1	0	1	0	0	0	**1**	**1**	**0**	**0**			
x	x	x	x	x	x	a	a	a	a	x	x	x

figure 3

The Table-Driven Micro-Assembler's Action

For example, the JMP macro-instruction might be implemented at the microprogram level on an AMD 2901 system by a microcode segment, comprising three microinstructions, which fetches the instruction operand and loads it into the program counter. The type of information supplied by the user is shown in figure 4. The notation used in this part of the microprogram synthesiser allows for the use of symbolic names in place of absolute addresses (often unknown at the time of writing the microprogram), and for the inclusion of comments.

Macro-instruction Mnemonic : JMP
Hexadecimal opcode : 60
Microcode segment

```
Areg #0000 & Breg #0000 & RAMA & ZB & ADD & CO=1 &I-U & YB-AR &CY ; get jump address
DR-DB & BREG #0000 & DZ & ADD & RAMF & I-U & CY   ; put jump address into program count
NOP & JUMPADDR #fetch & NOINCR & UNCOND & KF      ; go to instruction fetch cycle
```

figure 4

Microcode Sequence for an Assembler Level Jump

4. GENERAL REMARKS

The present implementation has been developed in Pascal on the MSDOS™ operating system, and is characterised by a user friendly man/machine interface. The sample processor, constructed on a plug-in prototyping board for the IBM PC using AMD 2901[2] bit-slice chips, can optionally be simulated.

Microprogramming courses have been run at Rhodes University in the fourth year of the Computer Science Curriculum both with and without the practical component described in this paper. Although difficult to measure quantitatively (the presence of a practical facility has partially altered the course approach), the introduction of the practical component has coincided with a visible increase in the general level of enthusiasm for and comprehension of the subject. The system has undergone several levels of refinement, with many suggested improvements coming from the students themselves.

ACKNOWLEDGEMENT

The author gratefully acknowledges the contribution made by Michael Ward in his implementation of the Rhodes Microprogram Development System.

REFERENCES

1. ACM Curriculum Committee on Computer Science, [1979], Recommendations for the Undergraduate Program in Computer Science, *CACM*, **22**, 3, 147-165.
2. Advanced Micro Devices, [1979], The 2900 Family Data Book with Related Support Chips, AMD.
3. Capper L., [1986], A Philosophy for the Teaching of Computer Science and Information Technology, *Computer Journal*, **29**, 1, 83-89.
4. Parker J.R. and Becker K., [1984], A Microprogramming Simulator for Instructional Use, *ACM SIGCSE BULLETIN*, **16**, 1, (Proceedings of the Fifteenth SIGCSE Technical Symposium on Computer Science Education).
5. Walker A.J., [1981], Teaching Bit-Slice Microprocessor Applications using Simulation Techniques, *Microprocessors and Microsystems*, **5**, 9.

SECOND
ANNUAL SUMMER SCHOOL AND COLLOQUIUM IN INFORMATICS

The second ASSCI, sponsored by the South African Institute of Computer Scientists and the Department of Computer Science of the University of Natal, Pietermaritzburg, will take place

from the **12^{th}** to the **15^{th} January 1988**

The summer school will take place on the first two days and will involve learning to program in LISP or in PROLOG with hands-on experience or studying an advanced functional programming language. Lectures in these areas will be drawn from:

Prof. N Phillips	*University Natal, Pietermaritzburg*
Prof. S Postma	*University Natal, Pietermaritzburg*
Dr. K Danhof	*South Illinois University*
Mr. P Wentworth	*University Port Elizabeth*
Mr. C Ammann	*University Natal, Durban*

The colloquium on the last two days will include discussions on logic programming and functional programming and in particular discussions on

Functional programming environments and applications, and parallelism and distributed processing in functional and logic programming.

The keynote address will be given on Thursday 14^{th} January by Dr Kenneth J Danhof of Southern Illinios University at Carbonale. His topic will be

Parallel Prolog and Database Applications

Delegates may register separately for the summer school and/or the colloquium if they wish. All sessions will be held on the Main Campus of the University of Natal Pietermaritzburg. Accommodation will be available in a University residence (bed and breakfast only). Any dependents wishing to stay in residence and not attend the ASSCI will be welcome. Teas will be provided and lunches and suppers are easily obtainable locally. Transport to/from the Pietermaritzburg airport and/or Pietermaritzburg railway station will be provided free of charge.

For further information contact the organizer:

Mr Chris Scogings
Department of Computer Science
University of Natal
P O Box 375, Pietermaritzburg 3200
Telephone : 0331-63320 ext 642/646

Fees	Summer School and Colloquium	R240 (SAICS members R120)	12-15 January
	Summer School	R200 (SAICS members R100)	12-13 January
	Colloquium	R 50 (SAICS members R 25)	14-15 January
	Accommodation	R 20 per night per person	

NOTES FOR CONTRIBUTORS

The purpose of the journal will be to publish original papers in any field of computing. Papers submitted may be research articles, review artilces and exploratory articles of general interest to readers of the journal. The preferred languages of the journal will be the congress languages of IFIP although papers in other languages will not be precluded.

Manuscripts should be submitted in triplicate to:

Prof. G. Wiechers
INFOPLAN
Private Bag 3002
Monument Park 0105
South Africa

Form of manuscript

Manuscripts should be in double-space typing on one side only of sheets of A4 size with wide margins. Manuscripts produced using the Apple Macintosh will be welcomed. Authors should write concisely.

The first page should include the article title (which should be brief), the author's name and affiliation and address. Each paper must be accompanied by an abstract less than 200 words which will be printed at the beginning of the paper, together with an appropriate key word list and a list of relevant Computing Review categories.

Tables and figures

Tables and figures should not be included in the text, although tables and figures should be referred to in the printed text. Tables should be typed on separate sheets and should be numbered consecutively and titled.

Figures should also be supplied on separate sheets, and each should be clearly identified on the back in pencil and the authors name and figure number. Original line drawings (not photocopies) should be submitted and should include all the relevant details. Drawings etc., should be submitted and should include all relevant details. Photographs as illustrations should be avoided if possible. If this cannot be avoided, glossy bromide prints are required.

Symbols

Mathematical and other symbols may be either handwritten or typewritten. Greek letters and unusual symbols should be identified in the margin. Distinction should be made between capital and lower case letters; between the letter O and zero; between the letter I, the number one and prime; between K and kappa.

References

References should be listed at the end of the manuscript in alphabetic order of the author's name, and cited in the text in square brackets. Journal references should be arranged thus:

1. Ashcroft E. and Manna Z., The Translation of 'GOTO' Programs to 'WHILE' programs., *Proceedings of IFIP Congress 71,* North-Holland, Amsterdam, 250-255, 1972.
2. Bohm C. and Jacopini G., Flow Diagrams, Turing Machines and Languages with only Two Formation Rules., *Comm. ACM,* **9**, 366-371, 1966.
3. Ginsburg S., Mathematical Theory of Context-free Languages, McGraw Hill, NewYork, 1966.

Proofs

Proofs will be sent to the author to ensure that the papers have been correctly typeset and *not* for the addition of new material or major amendment to the texts. Excessive alterations may be disallowed. Corrected proofs must be returned to the production manager within three days to minimize the risk of the author's contribution having to be held over to a later issue.

Only orginal papers will be accepted, and copyright in published papers will be vested in the publisher.

Letters

A section of "Letters to the Editor" (each limited to about 500 words) will provide a forum for discussion of recent problems.