A VERTICAL USER INTERFACE TO HORIZONTAL MICROCODE VIA A RETARGETABLE MICROASSEMBLER

David Abramson
John Rosenberg

Department of Computer Science
Monash University
Clayton 3168

ABSTRACT

Microcode was first introduced in 1951 as a technique for constructing the control unit of a processor [1]. Since that time, two basic forms of microcode have emerged namely, vertical microcode and horizontal microcode. This paper examines these techniques (and some variations of them) and discusses an approach which achieves the advantages of both. A new implementation of this scheme is described. The implementation aids hardware documentation and is flexible to hardware changes.

1. INTRODUCTION

The topic of this paper arose in the context of the MONADS project at Monash University [2]. The current phase of the project involves the design of a small microprogrammed processor to support software engineering principles [3,4]. The processor supports 32 bit data, 60 bit addresses and has about 100 internal registers. Consequently, it requires quite a complex microcode control word. During the design of the microcode structure some problems were encountered with conventional microcode techniques. This prompted the investigation of an alternative technique based on both vertical microcode and horizontal microcode, which resulted in the design and implementation of a retargetable microcode assembler. This paper describes the approach adopted and illustrates it with some examples.

The design of the new technique was driven by four main goals. First, the machine must provide an acceptable level of performance. The MONADS architecture differs from conventional machines, especially in the memory address area, and this results in even simple machine instructions requiring many basic operations. In order to ensure the overall efficiency we required as much parallelism as possible, as well as a short clock cycle. Second, we wanted the microcode to be easy to write, particularly as part of the operating system will be written in microcode. Third, because the requirements of the hardware are evolving as the project progresses, we required a scheme with as much flexibility as possible. This evolution may require certain previously sequential operations to be performed in parallel. Fourth, it is likely that the hardware may be re-engineered using VLSI technology, and this may result in changes in the microcode structure. It is desirable that the existing microcode be able to be used for the re-engineered machine.

By examining conventional schemes we will show that it is not possible to achieve all of these goals with current structures.

2. CONVENTIONAL MICROCODING TECHNIQUES

In this section we evaluate the two basic microcode structures, vertical and horizontal, as well as some variations of these. Of course most machines are neither purely vertical or purely horizontal, rather they are a combination of the two schemes. However it is possible to classify broadly machines into one or the other category and thus it is instructive to discuss the relative advantages of each.

2.1. Vertical Microcode

In a control unit which uses a vertical structure the microcode word is divided into a small number of encoded fields, each controlling some part of the processor. In a true vertical processor only one operation can be performed in any clock cycle, however, in practice a limited amount of parallelism is supported. The control word is usually tailored to a particular computer architecture or instruction set. An example of this scheme is shown in Figure 1.

The main advantages of vertical microcode are that it saves space in the control store (because many operations are encoded in binary) and also that it is, in general, easier to write than horizontal microcode. With the increasing size of control memories the space advantage is becoming less important. However, the increasing cost of human time means that it is more important to be able to write microcode which is correct, readable and easy to maintain. The vertical structure makes code easier to write because there is only a limited number of fields, which allows the programmer to easily comprehend the effects of one micro instruction.

There is also a number of disadvantages. Whilst it may be easy to decide which operations should be grouped together for some fields, (such as which registers to enable onto a bus), it is not always clear for all fields. The correct choice can only be determined once all of the microcode has been written. In a production environment it may be possible to write the microcode for the basic instruction set before the machine is built in order to check whether the encoding is appropriate. Unfortunately this is not always convenient in a research environment where the microcode is likely to be modified and extended at a later stage.

Another problem associated with vertical microcode is that many of the fields must be decoded after they have been fetched in order to produce the control signals for the processor. These decoders are not only time consuming, but also take up valuable circuit board space.

In spite of these problems a vertical structure has been used for many processors (e.g. HP2100 [5]). However, these difficulties are inconsistent with our design goals and thus a conventional vertical control word was not appropriate for our processor.

2.2. Horizontal Microcode

Horizontal microcode splits the control word into many fields, each controlling a small number of operations. Consequently, the control word is much wider than in the vertical scheme. An example of this scheme is shown in Figure 2.

The main advantage of horizontal microcode is that any number of operations can be performed at the same time. (In fact many of these combinations

may not even be sensible, such as enabling two registers onto a bus at the same time.) Also, there is no need for decoders as the control word bits can be used directly to control the processor. The absence of decoders saves board space, and also makes the instruction execution faster.

The problems with horizontal microcode are that it is difficult to write, and that it takes up more control memory than the vertical scheme. For reasons already stated the latter objection is not usually considered a problem. The large number of fields makes it difficult for a programmer to comprehend the effects of any one micro instruction. Also, several apparently unrelated fields must be specified in order to achieve a single logical operation. To limit the number of fields which must be specified, a key word approach is usually adopted with unspecified fields taking default values. A macro facility may also be provided.

Many modern machines are using horizontal microcode rather than vertical because of the falling cost of high speed memory and the considerable gains in instruction execution efficiency (e.g. the VAX 11/780 [6]). However, because of the difficultly in writing code, we were hesitant to use the horizontal microcode technique for our processor.

2.3. Nano-coding

Nano-coding is a technique which provides a horizontal control word but reduces the amount of control memory required. This is achieved by using two control memories, the first containing index values into the the second. The second memory then contains horizontal micro instructions, however, these are always addressed by reading the first memory. If a control word is to be repeated then only one copy resides in the second memory, and the index value is repeated in the first. Because the number of different horizontal microcode words required is typically quite small, the first memory can be much narrower than the second, saving overall control memory space. An example of this scheme is illustrated in Figure 3.

The technique has the advantage that it provides as much parallelism as necessary, without using a large amount of memory. However, the microcode programmer must still write microcode in the horizontal format and suffer the problems discussed earlier. Also, the scheme is slower than a pure horizontal method as the first memory must be read before the second can produce the microinstruction.

Nano-coding, although not widely used, has been adopted in systems where control memory space was restricted (e.g. M68000 [7]). As with the pure horizontal scheme we had grave concerns regarding writing large amounts of microcode.

2.4. High Level Microcode

A great deal of research has been made into techniques for compiling high-level programs (either in existing languages or special microprogramming languages) directly into microcode . (A good survey may be found in [8].) In this way each line of source code may produce a number of words of microcode. The advantages of this are clear; the code is much easier to write, debug and maintain. The major disadvantage is that the programmer no longer has a clear view of the underlying architecture, and thus does not have as much control over the efficiency of the final program. Whilst this argument has been largely dismissed in the case of assembler programming versus conventional high level programs on the grounds of programmer efficiency, it is harder to dismiss in the case of microcode.

It is the authors' view that microcode should be seen at a technique for implementing control hardware. Since the entire processor speed depends on the efficiency of the microcode, it is important that the programmer has as much control over the final code as possible. Thus, we rejected the high level language approach for our processor.

2.5. Summary

All of these coding techniques were rejected as they could not support the goals stated in section 1. Vertical microcode provides the ease of writing, but unduly restricts the inherent parallelism of the processor. It also slows the microcode down and complicates the hardware design by requiring decoders. Horizontal microcode provides a desirable hardware environment, but is awkward to read and write. Nano-coding still suffers the same deficiencies as horizontal microcode. High level microcode may restrict the programmer from making efficient use of the underlying micro machine.

3. A VERTICAL USER INTERFACE TO HORIZONTAL MICROCODE

We have described in detail the advantages and disadvantages of vertical and horizontal microcode. The scheme we now describe allows the machine to be built with a horizontal structure, but allows the user to write microcode using an encoded vertical format. The microcode assembler software [9] maps the vertical format of the source code onto the horizontal machine structure. In such a scheme the microassembler accepts source code in a user specified vertical format. The output of the assembler is binary code in a user specified horizontal format.

This scheme has the following advantages. The machine is basically horizontal in structure. Thus, there is no loss of parallelism, and also the hardware is faster and easier to design. The user, however, sees a vertical format, and thus need only be concerned with a small number of fields. It is therefore easier to read and write the code and more difficult to produce illegal or nonsensical combinations of fields. Also, if a basic operation requires several horizontal fields to be set, these can be defined in a specification file, thus eliminating a potential source of error.

A major difference between this scheme and a conventional vertical structure is that it is always possible to allow previously sequential operations to occur in parallel without altering the hardware. All that is required is a new vertical field value which sets more than one field of the horizontal microword. In fact any combination which could have been coded using the horizontal format can be coded using the vertical format and an appropriate mapping in the specification file. This is particularly important in a research environment, where microcode may be altered and rewritten after the processor has been built.

The scheme can be implemented by two main techniques. The first approach (adopted on the VAX 11/780 [6]) uses a key word type of micro assembler for the horizontal machine combined with a macro facility. It is then possible to develop a set of macros which provides the vertical interface. The second technique relies on a retargetable microassembler which accepts a specification file defining the mapping between the vertical and horizontal machines. Examples of such an assembler are AMDASM [10] and M29 [11].

All of the implementations of both techniques examined by the authors allowed a clear specification of the vertical machine, however, the description of the underlying horizontal machine tended to be scattered throughout the vertical machine specification. Thus, if modifications are made to the

underlying architecture all of the vertical fields related to the altered horizontal fields must be located and corrected. This can be a non-trivial task. Consequently, we decided to develop a new micro assembler which clearly distinguished between the vertical user interface and the underlying horizontal machine.

4. THE MICROASSEMBLER

The microassembler is controlled by a specification file, which contains the necessary information to parse input lines, validate operands, map vertical fields onto horizontal fields, and produce code. The specification file may include comments (signified by a semi-colon) to make it more intelligible and maintainable. A clear distinction is made between fields held in the source code (called source fields) and fields of the microword (called object fields).

4.1. Source fields

Each field on a source line is called a source field, and may contain either a defined field value, a constant or a label. Each source field must be declared using the following code:

```
source field <identifier>
    position [is] <number>
    size [is] <number> [characters]
    legal values
          ( "<string>" [in <setname>] { "<string>" [in <setname>] }
          | label
          | value )
```

In this code, the identifier defines the name of the field. The position indicates the relative position of the field in the source line. The size gives the size of the field in characters. Either a list of legal field values may be specified, or the keyword label or value. Legal values may be grouped into sets for error detection purposes. If a label or value is selected the field must contain either a legal number or a label name. The following example shown the declaration of a source field called ENX which has three legal field values.

```
source field ENX        ;enable the X bus registers
        position is 1
        size is 3  characters
        legal values are

        ""
        "NOP"
        "MDR"           ;the memory data register
```

An example of a constant field and a label field are shown below:

```
source field CONSTANT   ;constant field
        position is 8
        size is 8  characters
        legal values are value
```

```
source field TARGET     ;target address
        position is 9
        size is 10  characters
        legal values are label
```

These source field declarations are used to parse the source line, and to validate the contents of each field. They are not used in code generation.

4.2. Object Fields

Each field of the horizontal microword is called an object field. Control over code generation is achieved with object fields, using the following declaration:

```
object field <identifier> is microword [<number>:<number>]
    [ size is <number> [bits] ]
    value [is]
          { <number> if <sourcefieldname> = "<string>" }
          label in <sourcefieldname>
          value in <sourcefieldname>
    [ otherwise <number> ]
```

During code generation each object field specification is scanned. If one of the if statements matches with the field value contained in the named source field, then the appropriate number is chosen. This number is then placed in the bits of the microword specified by the range [<number>:<number>]. If none of the if statements match, then the otherwise number is placed in the microword.

If the object specification is for a label or value, then either the value or the label address (from the symbol table) is placed in the microword. An example of an object declaration which sets the values for a six bit object field called RAMX, is shown below:

```
object field RAMX is microword[5:0]
        size is 6 bits
        value is

        5  if ENX = "MDR"
        5  if WRR = "MDR"

        otherwise 0
```

In this example the value 5 is placed in bits 0 through 5 of the microword if either the source fields ENX or WRR have the value 'MDR'. If neither has this value then 0 is placed in the microword. An example of fields which use a label value and a constant value are shown below:

```
object field CONST32 is microword[91:60]
        size is 32 bits
        value is value in CONSTANT

object field TARGET is microword[75:60]
        size is 16 bits
        value is label in TARGET
```

4.3. Illegal Combinations

Some combinations of source fields may still be illegal. One common situation is when two or more source fields produce code for the same bits in the microword (e.g. in the case of constants and label targets which share bits in the microword). The microassembler automatically detects conflicting values for these fields and produces appropriate error messages. Other illegal combinations (e.g. attempting to read and store back into a latch device in a single cycle) can be declared using the following code:

illegal if <source_field_name> [(= | #) ("<literal>" | <setname>)]

{ and <source_field_name> [(= | #) ("<literal>" | <setname>)] }

where '#' signifies inequality.

These declarations are used when the source line is being validated with the source field specifications. An example is shown below:

```
illegal if ENX = "MDR" and
           WRR = "MDR"      ; MDR is a latch device
```

This would cause an error message to be generated if a source line has 'MDR' in both the ENX and WRR fields.

4.4. Other Specifications

The specification file also contains the details of how the microword is to be split into memory devices (such as Read Only memories), together with the dimensions of the memories.

5. AN EXAMPLE

We now consider a small example to demonstrate use of the assembler. The example in Figure 1 shows a vertical control word with two fields. One field determines which of the processor registers is enabled onto a bus, and the other determines which register is modified. Various different modifications are possible namely, writing into a register, clearing a register or incrementing a register. Because of the vertical structure it is not possible to copy the Program Counter (PC) to the Memory Address Register (MAR) whilst incrementing the PC.

From the view of the programmer, each line of microcode only consists of two fields. The ENABLE field has the following values:

MDR, MAR, IR, PC

The MODIFY field has the following values:

MDR, MAR, IR, PC, INCPC, CLRIR, INCMAR, NOP

Coding is thus quite easy.

Figure 2 shows the same machine implemented using a horizontal control word. Each signal in the vertical machine uses a separate bit of the control word. The source level of this horizontal machine has 11 fields, each of which can either be set or clear. The following fields are required:

ENMDR, ENMAR, ENIR, ENPC, STMDR, STMAR, STIR, STPC, INCPC, CLRIR, INCMAR

Coding for the horizontal machine requires many more fields to be set or cleared. It is also possible to create combinations of fields which do not

make sense, such as ENMDR and ENMAR, both of which enable registers onto the same bus. At the same time it is possible to achieve useful combinations which are not attainable with the vertical machine, such as copying the Program Counter to the Memory Address Register whilst incrementing the Program Counter.

A vertical interface to this horizontal machine can be defined for the new micro assembler. The following specification file is required:

```
source field ENABLE
        position is 1
        size is 5 characters
        legal values are

                "NOP"
                "MDR"
                "MAR"
                "IR"
                "PC"

source field MODIFY
        position is 2
        size is 15 characters
        legal values are

                "NOP"
                "MDR"
                "MAR"
                "IR"
                "PC"
                "INCP"
                "CLRIR"
                "INCMAR"

object field ENMDR is microword[0:0]
        size is 1 bit
        value is  1 if ENABLE = "MDR"
        otherwise 0

object field ENMAR is microword[1:1]
        size is 1 bit
        value is  1 if ENABLE = "MAR"
        otherwise 0

object field ENIR is microword[2:2]
        size is 1 bit
        value is  1 if ENABLE = "IR"
        otherwise 0

object field ENPC is microword[3:3]
        size is 1 bit
        value is  1 if ENABLE = "PC"
        otherwise 0

object field STMDR is microword[4:4]
        size is 1 bit
        value is  1 if MODIFY = "MDR"
        otherwise 0
```

```
object field STMAR is microword[5:5]
        size is 1 bit
        value is  1 if MODIFY = "MAR"
        otherwise 0

object field STIR is microword[6:6]
        size is 1 bit
        value is  1 if MODIFY = "IR"
        otherwise 0

object field STPC is microword[7:7]
        size is 1 bit
        value is  1 if MODIFY = "PC"
        otherwise 0

object field INCPC is microword[8:8]
        size is 1 bit
        value is  1 if MODIFY = "INCPC"
        otherwise 0

object field CLRIR is microword[9:9]
        size is 1 bit
        value is  1 if MODIFY = "CLRIR"
        otherwise 0

object field INCMAR is microword[10:10]
        size is 1 bit
        value is  1 if MODIFY = "INCMAR"
        otherwise 0
```

Using this specification, microcode with only two fields per line can be assembled for the horizontal machine shown in Figure 2. We could now define extra field values to gain parallelism. For example, the specification file can be changed to allow the PC to be enabled and then incremented. This requires the following alterations:

```
source field ENABLE
        position is 1
        size is 5 characters
        legal values are

                "NOP"
                "MDR"
                "MAR"
                "IR"
                "PC"
                "PC&INC"

object field ENPC is microword[3:3]
        size is 1 bit
        value is  1 if ENABLE = "PC"
                  1 if ENABLE = "PC&INC"
        otherwise 0
```

```
object field INCPC is microword[8:8]
        size is 1 bit
        value is  1 if MODIFY = "INCPC"
                  1 if ENABLE = "PC&INC"
        otherwise 0
```

This would not have been possible without hardware modifications had a true vertical machine been built.

The flexibility of the specification technique can further be demonstrated by another example. Consider the case in which we wish to change the sense of the hardware signal INCPC. All of the implementation information relevant to the object field INCPC is contained in the one definition. Thus, the only modification necessary to the specification file is shown below:

```
object field INCPC is microword[8:8]
        size is 1 bit
        value is  0 if MODIFY = "INCPC"
                  0 if ENABLE = "PC&INC"
        otherwise 1
```

This change may be contrasted to the changes necessary in a macro scheme or a traditional retargetable micro assembler (such as AMDASM and M29) in which the information pertaining to INCPC would be contained in the specification of both of the fields MODIFY and ENABLE.

6. CONCLUSION

The assembler has been written and tested and a specification file for the MONADS processor has been developed. The MONADS micro machine has a horizontal structure with approximately 50 object fields. The vertical view of this has only 12 source fields, and a line of source code fits neatly onto a terminal screen line. This conveniently allows a standard screen editor to be used for entering and updating the microcode source. Most of the microcode for the basic instruction set of the processor has been written.

During the development of the microcode it has been necessary to modify the machine specification file. This was necessary when the microcode programmer discovered that certain sections of the microcode could be more efficiently coded if several operations could be performed in parallel. Other changes have included alterations to the encoding of some object fields. In all cases these modifications only required small changes to the specification file. It was only possible for the programmer to suggest these changes because he had knowledge of the underlying structure of the MONADS machine. This was obtained from the specification file which, because of its English-like syntax, forms a useful part of the documentation of the hardware.

REFERENCES

[1] Wilkes, M.V. "The Best Way to Design an Automatic Calculating Machine", Report of the Manchester University Inaugural Computer Conference, Manchester, 1951.

[2] Keedy, J.L. "Support for Software Engineering in the MONADS Computer Architecture", Ph.D. Thesis, Monash University, August, 1982.

[3] Rosenberg, J. "MONADS-PC Instruction Set", MONADS-PC Technical Report Number 1, Department of Computer Science, Monash University, March, 1984.

[4] Abramson, D.A. "MONADS-PC Micro Architecture Manual", MONADS-PC Technical Report Number 2, Department of Computer Science, Monash University, March, 1984.

[5] Hewlett-Packard, "Microprogramming Guide for Hewlett-Packard Model 2100 Computer", Hewlett-Packard, November, 1971.

[6] Digital Equipment Corporation, "VAX 11/780 Microprogramming Tools User's Guide", Digital Equipment Corporation, June, 1979.

[7] Stritter, S. and Tredennick, N. "Microprogrammed Implementation of a Single Chip Microprocessor", Proceedings of the 11th Annual Microprogramming Workshop", pp. 8-16, November, 1978.

[8] Sint, M. "A Survey of High Level Microprogramming Languages", Proceedings of the 13th Annual Microprogramming Workshop", pp. 141-153, November, 1980.

[9] Tuke, M. "MONADS-PC Microassembler Specification", MONADS-PC Technical Report Number 4, Department of Computer Science, Monash University, Mat, 1984.

[10] Advanced Micro Devices "The Am2900 Family Data Book", Advanced Micro Devices Inc., 1978.

[11] Eager, M.J. "M29 - An Advanced Retargetable Microcode Assembler", Proceedings of the 16th Annual Microprogramming Workshop", pp. 92-100, October, 1983.

FIGURES

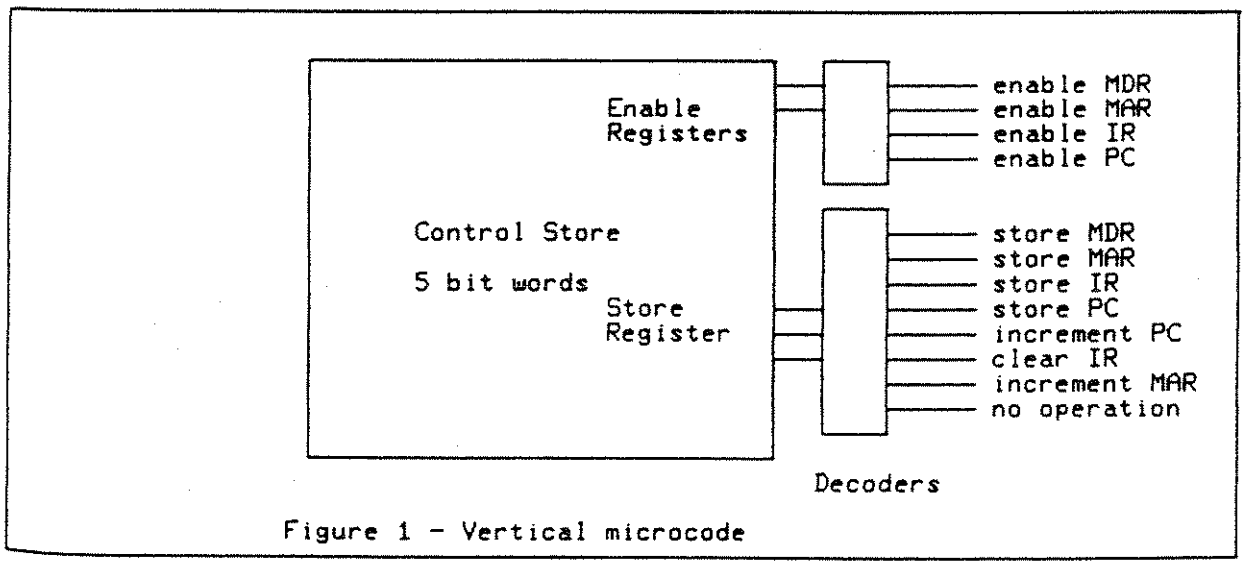

Figure 1 - Vertical microcode

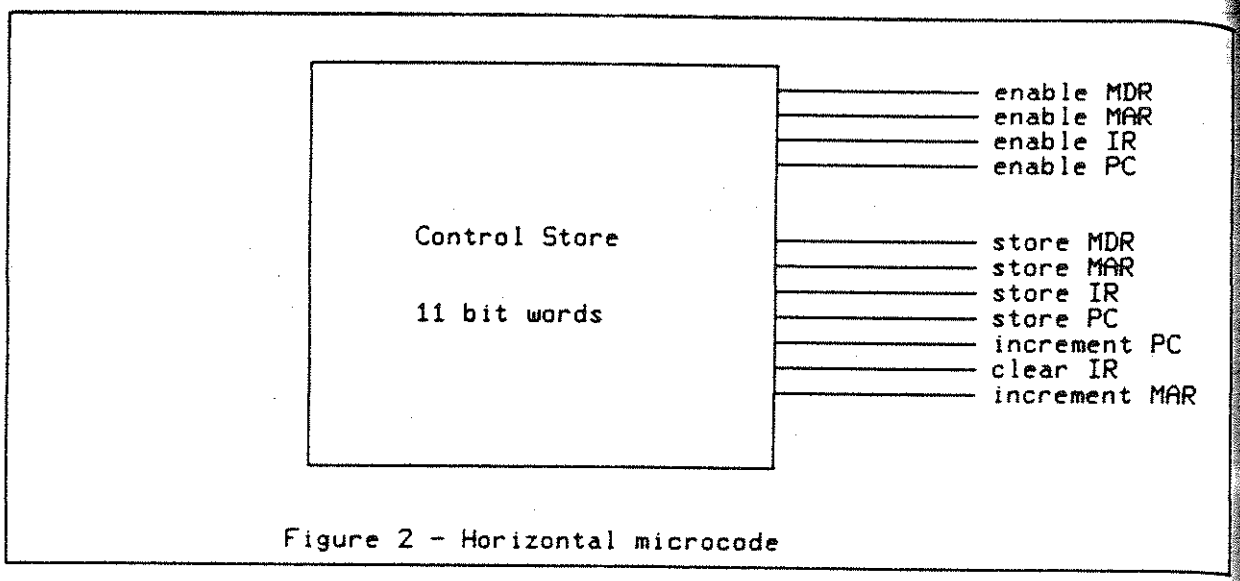

Figure 2 - Horizontal microcode

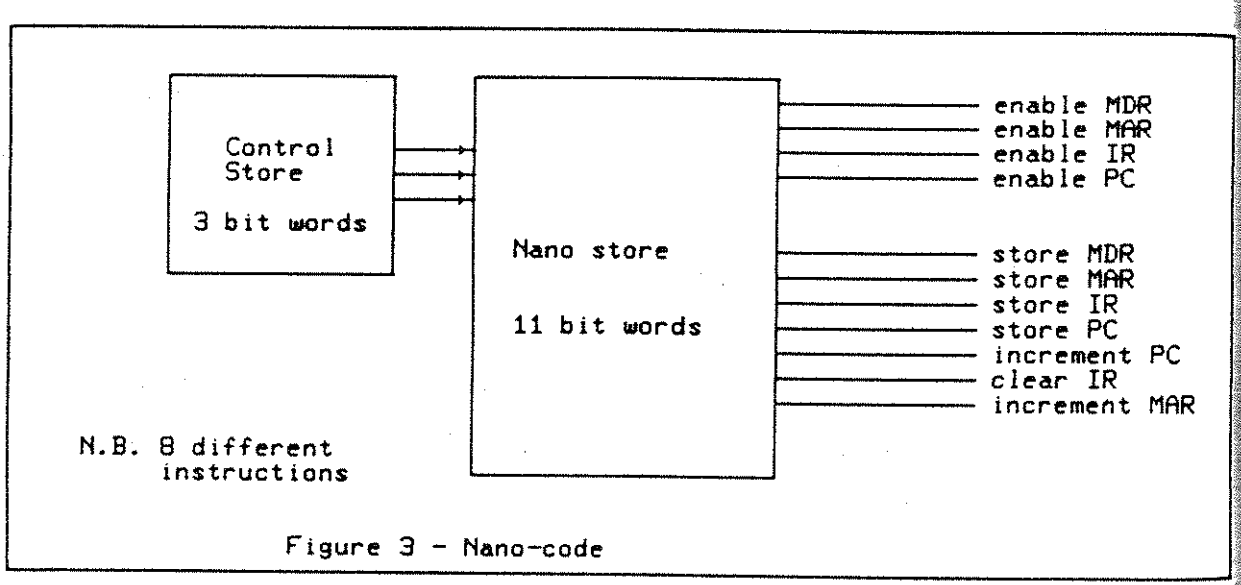

Figure 3 - Nano-code

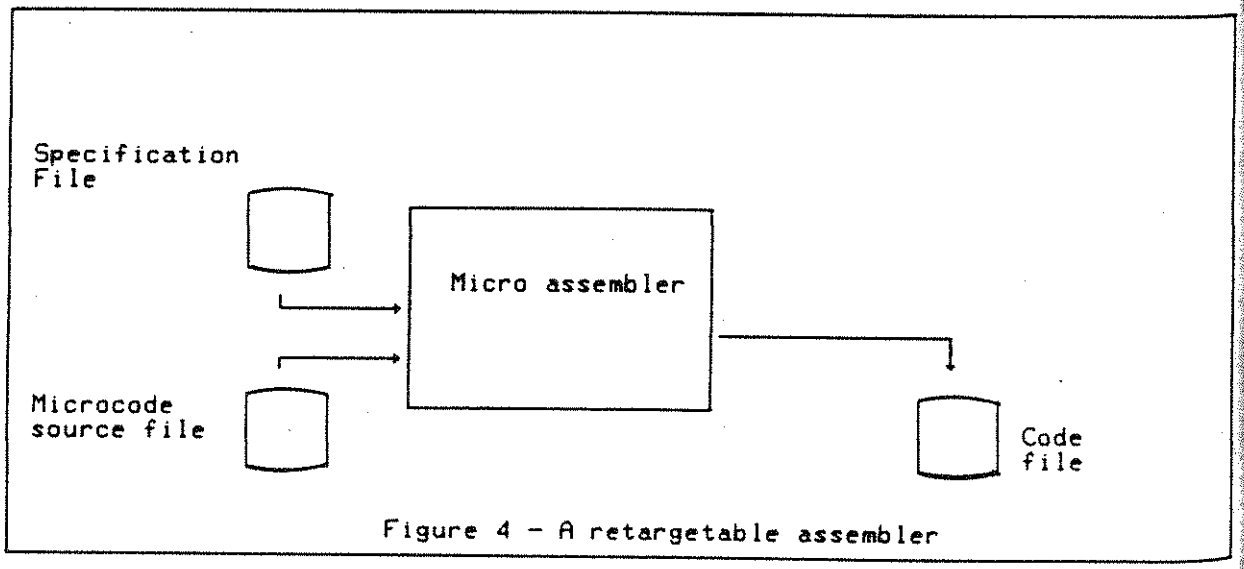

Figure 4 - A retargetable assembler

AUSTRALIAN COMPUTER SCIENCE COMMUNICATIONS

Vol. 7, Number 1

February 1985

PROCEEDINGS OF THE EIGHTH AUSTRALIAN COMPUTER SCIENCE CONFERENCE

Department of Computer Science,
Monash University,
Melbourne, Victoria.

Department of Computer Science,
University of Melbourne,
Melbourne, Victoria.

ACSC-8