PUC

Series: Monografias em Ciência da Computação
Nº 5/79

MICROPROGRAMMING
PART II: DEVELOPMENTS IN MICROPROGRAMMING LANGUAGES

by

Ayuia N. Akonteh

Departamento de Informática

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO
RUA MARQUÊS DE SÃO VICENTE, 225 – CEP-22453
RIO DE JANEIRO – BRASIL

PUC/RJ - DEPARTAMENTO DE INFORMÁTICA

Series: Monografias em Ciência da Computação
Nº 5/79
Series Editor: Daniel A. Menascé

March, 1979

MICROPROGRAMMING
PART II: DEVELOPMENTS IN MICROPROGRAMMING LANGUAGES*

by

Ayola N. Akonteh

* This work has been sponsored in part by FINEP.

INDICE

Abstract:

This study, part II of a series on microprogramming, reviews some of the difficulties in the development of Higher Level Microprogramming Languages (HLML) and how some of these difficulties have been resolved. It concludes that the difficulties of HLML design are partly due to lack of participation of language designers in the hardware design.

Key Words:

Compiler Optimization, Delimiters, Microprogramming Languages, Octal mask, semantics, syntax.

Resume:

Esta Parte II de uma serie sobre microprogramação tem como objetivo uma breve recordação das dificuldades encontradas em desenvolvimento das linguagens de microprogramação de alto nivel (HLML) e para mostrar como algumas dessas dificuldades já foram resolvidas. O estudo conclui que algumas das dificuldades em projetos de (HLML) são atribuidos a falta de participação inicial na parte de desenhistas de (HLML) em projetos de hardware.

Palavras Chave:

Otimização de compilador, delimitadores, linguagem de micropro - gramação, máscara octal, semanticas, sintaxe.

MICROPROGRAMMING

PART II: DEVELOPMENTS IN MICROPROGRAMMING LANGUAGES

2.0 - Introduction

The original interest in microprogramming stems from three important uses: interpretation, emulation and control in digital systems. The horizon has now expanded to include compiler optimization, alternate levels of program implementation, signal processing, etc; together with a flexibility that permits alternate function definition and emulation of varying architectures.

The greatest difficulty in microprogramming remains efficient codification due to lack of adequate higher level languages for same. This study briefly surveys some of the difficulties in microprogramming language developments and some of the progress made in this direction.

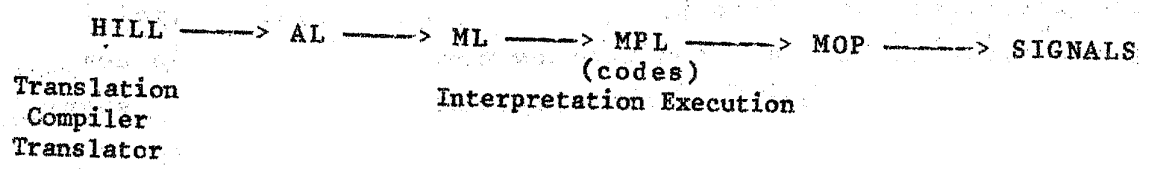
2.1 - Microprogramming Language Development

The greatest impediment to microprogramming language development lies in a thorough understanding of both the software and hardware architectures of various systems to design a common language. Other difficulties exist with some of them and are traceable to the following necessities.

- Few microprogrammers are participants in the design of systems they microprogram.
- Major systems design considerations are economic with little consideration for syntax, semantics, portability and other problems that go with firmware development.
- Microprogramming originally limited to interpretation, is today used in compiler optimization, control in digital systems, signal processing, etc.
- There now exist a need for easy and alternate function

definition in digital systems. Most microprogrammable machine (Nannodata QM-1) can have their instruction set altered by new definitions.

In general, basic trends in microprogramming language development has followed the same trends as found in higher level languages, the first of which was the ML (register transfer language) developed for the LX-1 at MIT (Hornbuckle, 1970)



Some of the later developments can be traced to Eckhouse (1971), as well as Mallet and Lewis (1971) with some references made by Husson (1972). Some of the Computer Hardware Design Languages (CHDL - (Chu-1972)) donot seem to offer quite the same facility as traditional high level languages .

2.2 - Some Microprogramming Languages

Microprogrammable computers are characterized by varying architectures (horizontal, vertical, etc) which make it necessary to classify microprogramming languages according to their encoding. There exist at least three types of microprogramming languages, besides the register transfer languages; viz.

a) Assembly language;
b) flowchart;
c) higher level language.

2.2.1 - Assembly Language Type

This form is used mostly in vertically organized machines such as the IBM 360/25.

The assembly language type of microprogramming languages offers limited functional capabilities in parallelism or branching. The only advantage is that it may be coded in higher level notation such as found in the IBM 360/25.

For example, $U_1 = U_1 + HO$

indicate 1 byte addition of register contents U_1 and HO in the IBM 360/25 (Husson, 1970). In the assembly language notation of the same machine, the following indicate the ORing of constants:

```
V1 = V1  $  K04
         OR(+)
```

2.2.2 - Flowchart type

The flowchart type of microprogramming language apart from offering an ease to read and possibilities of algebraic or Mnenonic representation is characterised by graphical information similar to logic flow diagrams. Each line of information represents a distinct function. It has the advantage in that it could accomodate several control schemes with all sequencing functions explicity stated.

2.2.3 - The Higher Level type

The development of higher level microprogramming languages though difficult offers some advantages, among which are the following.

i) Specific problem orientation rather than machine oriented;
ii) Reduction in software development time because of familiarity with expression;
iii) Program readability hence easy proof of correctness;
iv) Possible program portability or adaptation;

Exist also disadvantages, some of which are as follows:

i) Slow execution;
ii) Memory space for object program;
iii) Diagnosis of error depend or Interpreter and programmer's knowledge of target machine.
iv) Implantation may be difficult.

A normal test to determine if a given microprogramming language is high level would include.

i) Defined hierachy;
ii) Defined syntax;
iii) Defined semantic;

Most known high-level microprogramming languages, however, lack one or two of the above characteristics and may even lack some of the following capabalities.

i) Declarations;
ii) Procedural statements embedded in blocks;
iii) Timing and concurrency of operations;
iv) Portability;

2.3 - Examples of Microprogramming Languages

Several examples of microprogramming languages exist with limited generality, efficiency and ease of coding. The following survey indicates some of these languages.

2.3.1- The General Purpose Microprogramming Language (GPM)

The GPM was developed at the USC's Information Science Institute for the MLP-900 microprocessor linked to a PDP-10 by an input/output bus. It provides users access to a writable control memory (4K) microprocessor as a service in the ARPANET multiprogrammed environment.

The goal of GPM is to provide a higher level language with wide coding options. GPM Statements fall into four (4) categories:

i) syntactic bloc structure;
ii) hardware generalization;
iii) Multi-instruction Statements;
iv) Expressions;

The syntactic block structure takes the form

```
BEGIN
  Declarations
    Body
      END.
```

Hardware provides a jump on less than zero with a higher level language providing other jumps relative to zero. An example of this is in the GOTO destination.

```
GOTO 100;  Absolute jump
GOTO +10;  relative jump
```

Hardware may also perform assignments, for example

CE0 ←——— CE1 (77)

Which signifies transfer of contents of register 1 to 0 for all bits in an octal mask.

There are several other interesting features of the GPM (see Oestreicher) including multi-instruction statements, expressions, etc. Despite its interesting attributes, it remains a specialised language for the MLP-900.

2.3.2 - MIDDLE (Microprogram Design and Description Language).

MIDDLE (Dembinski and Budkowski, 1978) allows the description of executing hardware, has a full mathematical semantic description and allows for microprogram verification. A program in MIDDLE normally consist of series of declarations followed by entry labels and by a sequence of labelled statements separated by semicolons (Debimski et al, 1978). The following is an example of the syntactic characteristic of the syntactic characteristic of MIDDLE.

<Program>::=<declarations>start<entry>;<statement list>

Declarations in MIDDLE may be of constants, functions. on variables and may take the form,

declaration :: = dcl<constant part>
<function part>
<variable part led>

A variable declared to be of a certain type (binary, integer, real, array, etc) may also have a hardware specification (sequential) combinational). In general therefore, MIDDLE meets with a lot of the higher level microprogramming language characteristics suggested by Dasgupta (1978).

2.3.3 - The Datasaab FCPU Microprogramming Languages (ML).

The (ML) though with higher level language attributes is specially developed for the Saab FCPU a vertically organized (encoded) machine. The (ML) structure is simple with minimized use of delimiters except = -, ( ):/* and an 'End - of - card'.

The language can specify both decimal and hexadecimal constants and comments beginning with the delimiter (*) and ending with 'End-of-card'. There is no provision for continuation.

The general ML program may be represented as follows:

```
<DECLARE>
      <Global Declarations>
            BEGIN
               <Statements>
                  DECLARE
                       <Local Declarations>
                            BEGIN
                               <Statements>
                                       END
                                       END
```

Note that the inner block of this program form can be repeated n times, where $0 \leq n < \infty$. Other interesting features of ML include vectors and subroutines, etc.

CONCLUSIONS

This study though presenting only a passing view of microprogramming languages does accentuate some of the common problems encountered in integrating hardware and software concepts to develope a consistent, codable microprogramming language. Participation of hardware designers in the development of language facilities will probably ease some of the barriers to create a language for alternate level program implementation.

Bibliography

1. Akonteh, A.N. Theoretical evolution of the microprogramming concept.
Monograph in Computer Science #2/79 PUC-RJ

2. Akonteh, A.N. Microprogramming Principles and Developments
Monograph in Computer Science, #4/79 PUC-RJ

3. Akonteh, A.N. Trends in Emulation Technology
Monograph in Computer Science #6/79 PUC-RJ

4. Dasgupta, S. Towards a Microprogramming Language .
11th Annual Microprogramming Workshop Pacific Grove, California, 1978

5. P. Debinski e Al. An introduction to the verification of a Microprogramming Language "Middle".
11th Annual Microprogramming Workshop Pacific Grove, California, 1978

6. Husson, S.S. Microprogramming : Principles and Practices
Prentice-Hall, Englewood, N.J; 1970