A COMPARATIVE STUDY OF HIGH LEVEL MICROPROGRAMMING LANGUAGES

by

Eugene Schreiner

B. S., Fort Hays Kansas State College, 1967

A MASTER'S REPORT

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1973

Approved by

Myron A. Calhoun
Major Professor

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I wish to acknowledge the assistance given me by Dr. Myron Calhoun, my major professor, Dr. Paul Fisher, and Dr. Virgil Wallentine in the preparation of this report. I especially want to thank Dr. Wallentine for suggesting the topic and the basic format and Dr. Calhoun for proof-reading the report.

Also, I wish to thank Sharon, my wife, for typing my report and many other student's papers so that I could attend graduate school.

1.0 INTRODUCTION

1.1 Basic Concepts of Microprogramming

The concept of microprogramming is normally attributed to Wilkes (25). He envisioned the control portion of a computer as a number of register transfers (micro-operations) which, when performed in the correct sequence, resulted in the execution of a machine instruction. A collection of these micro-operations which requires one basic time unit to execute is called a microinstruction. A sequence of microinstructions is called a microprogram. In recent years, microprograms, stored in high-speed, nondestructive read-only storage (ROS), have been used to control the sequences of the machine instructions of a number of digital computers.

The reader who may be unfamiliar with microprogramming may refer to Appendix A which contains the description of a simple hypothetical microprogrammed computer (microprocessor) or to one of the following references (9, 17, 20, 25).

To date, the main reason for the use of microprogram control has been in order to make a complete line of computers with different internal structures and performance ranges accept the same instruction set. An example of this is the IBM S/360 line of computers. Recently several microprogrammable computers with writable control storage have been implemented--MPL-900 (formally IC-9000), Microdata 3200, Hewlett-Packard 2100S, and B1700. A microprogrammable computer differs from a microprogrammed computer by the support of a facility which allows the user to change the microprogram. These microprogrammable machines have the advantage of allowing the user to tailor the microprogram to match his specific application. The advantages and disadvantages have been discussed in a number of papers (9,pp.29-30; 17, pp. 117-125; 23, pp. 12-21).

As more microprogrammable computers become available, as the trend appears to be, it will become the user's responsibility to take full advantage of this new tool.

1.2 The Need for a High Level Microprogramming Language

Today if a microprogrammer wishes to alter the control storage of a microprogrammable computer, he has to write the new microprogram in an assembly-like language. A similar situation once prevailed in the programming of computers. The programmer had to write all programs in detailed assembly language. When higher level programming languages were developed, the programmer's job was simplified considerably. The many advantages of higher level programming languages should also be available to the microprogrammer. Therefore, the development of a high level microprogramming language seems to be an admirable goal.

A high level language would also reduce the burden manufacturers have in producing the microprograms for their computers. Microprograms for manufacturer's computers are often more difficult to write than the ones a user would write for a microprogrammable computer. This increased difficulty arises from the use of long, complicated, special purpose microinstruction formats. At the present time, a flow-chart-like language is often used which is automatically assembled into microcode (24, p.236).

The argument against the development of a high level language for microprogramming usually stresses the point that microprogramming is not like programming, in that in microprogramming efficient microcode is more important than ease of programming. This is true; microcode that is used over and over cannot be inefficient code. But I feel the microprogrammer should have a high level language to use if he wishes. A situation where such a language would be useful to a microprogrammer is when he is writing microsequences that will not be used a high percent of the time, but will be swapped in and out of control storage as needed. The need for a standard notation to describe microprogramming would be satisfied by a high level microprogramming language.

The speed of computers has increased tremendously in the past few years and will probably continue to increase. It is costing more to write and debug programs than to execute them. Ease of programming has at times become a more important issue than the speed at which the program is executed. This has caused a great increase in both special purpose and general purpose high level programming languages. This increased computer speed makes a high level micro-programming language attractive. If optimization techniques are used at compile time, the sometimes less efficient microcode produced for these routines may be outweighed by the savings in the time to write the microcode.

There have been a number of attempts to develop a high level microprogramming language. These have had varying degrees of success. This report will discuss various proposed languages and the attempts to implement them as microprogramming languages.

1.3 Purpose of this Report

The purposes of this report are as follows:

1. To indicate the characteristics that are required for a suitable high level microprogramming language.

2. To present a survey of the current literature in the area of defining and implementing such a microprogramming language.

3. To judge each of these languages on how well they satisfy these characteristics.

4. To determine if there is a suitable high level microprogramming language currently available.

2.0 REQUIREMENTS FOR A HIGH LEVEL MICROPROGRAMMING LANGUAGE

The following (2.1-2.8) is a list of the basic requirements for a high level language for microprogramming. The list is a composite of the characteristics and requirements given by a number of authors: Husson (17), Chu (3), Hill and Peterson (16), Eckhouse (10), Crockett (7), and Tsuchiya (23). The languages that are compared in this report were selected with these requirements in mind. Only languages which are designed to simulate microprogrammed computers and/or produce microcode are discussed. Therefore, this report does not include a number of currently implemented high level hardware design languages. Although these hardware design languages do satisfy many of the requirements listed below, they do not, in my opinion, appear to be adaptable to microprogramming.

The languages compared in this report are:

CDL--Computer Design Language,

CASD--Computer-Aided System Design,

MPL--Microprogramming Language,

SIMPL--A Single Identity Microprogramming Language,

APL--A Programming Language, and

AHPL--A Hardware Programming Language.

A general description and example programs written in each language is given in Appendix B. A chart comparing the features of these six languages is given in Appendix C.

2.1 Natural Programming Style

A microprogramming language will satisfy this requirement if it has syntax like a common high level programming language. In many cases the syntax can be simpler than the syntax of languages like PL/1, FORTRAN, and ALGOL, because it need only be designed for the special purpose of microprogramming. If the language is easily understood the microprogrammer will only need the source listing to communicate the details of his microprogram to others. The more natural the language, the easier users from various disciplines will master it.

In my opinion the PL/1 and ALGOL-like languages of CDL, CASD, and MPL satisfy this requirement, while APL and AHPL do not. A glance at the example programs in Appendix B should verify this opinion. Two reasons I feel APL doesn't satisfy this requirement are: (1) APL's vocabulary is quite large and descriptions are difficult to read initially; and (2) the notation contains Greek letters and other unfamiliar symbols which make the language difficult and cumbersome for the average programmer. AHPL solves these problems, to some extent, by eliminating many APL operators that are not pertinent to computer descriptions.

SIMPL's notation is not as easy to read as CDL, CASD, or MPL because unique variable names must be used to describe changes in the data in a hardware component. I would say the ease of comprehension of SIMPL's notation lies somewhere between that of the PL/1 and ALGOL-like languages and APL.

2.2 Descriptive

The language must be descriptive so both structural properties and microprogram control of the systems can be described. The microprogrammer must be able to declare every computer component from the main memory to lights on the control panel. He must be able to describe the timing points which will effect the execution of the microprogram. He must have means to easily describe the action implied by every micro-operation. Both the declaration and execution statements must be free of all ambiguous notation. Unspecified notation may cause a compiler to take a default action which could lead to a catastrophe, such as generating a temporary register that does not exist in this computer's hardware. The language must have sufficient detail so gating of bit, word, and bit-array level operations can be specified. At the same time the description of irrelevant components and operations should not be required. Lastly, the language must have means available to allow the description of the microinstruction format and control fields.

In summary, the language must describe the computers internal structure, microinstructions, and micro-operations in an unambiguous and precise way.

Using CDL a microprogrammer can describe the internal structure of a computer using _register_, _subregister_, _cas-register_, _memory_, _clock_, _switch_, _light_, and _terminal_ declaration statements. The micro-operations are unambiguously described using labeled execution statements, where the label allows the programmer to explicitly indicate the clock phase and other conditions which must be present to evoke the execution of a micro-operation. The description of the microinstruction control fields is accomplished using _decoder_, _terminal_, and labeled execution statements.

CASD and MPL describe a computer using PL/1-like statements. Declaration statements are used to describe the internal structures of the computer. Micro-operations are described using PL/1-like execution statements. To my knowledge, CASD and MPL lack the facilities to describe the microinstruction control fields.

A microprogrammer using SIMPL would describe the internal structure of the computer and its microinstruction in a separate MODEL program. The MODEL program is written once and is then used as part of the input to the SIMPL compiler for each microprogram written for this computer. An ALGOL-like execution statement is used to indicate micro-operations. (See Appendix B for example programs)

Using APL a microprogrammer can describe micro-operations in a concise, precise, and unambiguous manner. APL does not have adequate means to declare internal computer structure and microinstruction control fields. APL has been critized by Eckhouse (10, p. 8), Darringer (8, p. 9), and Gentry (13, p. 4) as lacking the facility to declare and thereby accurately describe the properties of computer components.

AHPL has register, bus logic, and bus load declaration statements. It does not have the means to declare other internal components. Micro-operations are described using a subset of APL notation. There are only four APL logic and arithmetic functions allowed in AHPL. These are "and," "or,""not," and "exclusive-or." Since AHPL was not designed for microprogramming, it lacks the means to describe microinstruction formats.

2.3 Procedural

The microprogrammer must be able to specify the sequence in which a set of operations is to be performed. Unlike high level programming languages that may have complex control structures such as recursion, a microprogramming language requires only simple structures. Some method to write subroutines and perform single or multiple branches is all that is required.

Facilities to satisfy this requirement are provided by: procedure and computed go to statements in CASD, MPL, and SIMPL; special operator and labeled execution statement in CDL; combination logic subroutines and branch statements in AHPL. APL's function satisfies the requirement for subroutines and the programmer can write his own multiple branch statements, see (1, p. 107) or Example APL Program line 7 of FETCH in Appendix B.

2.4 Timing and Concurrent Operations

Some means to indicate timing signals must be available to allow the micro-operations to be executed at a particular time. The language will need some kind of timing signals so the programmer can have micro-operations performed in parallel or sequentially. One of the advantages of horizontal microinstruction formats is to allow parallel data transfer. The language must have a way of indicating these parallel data transfers if efficient microcode is to be generated by the compiler. Facilities to indicate synchronous and asynchronous micro-operation execution are also required.

The CDL's labeled execution statements can be used to indicate concurrent operations that are synchronous or asynchronously performed. CDL has a clock declaration statement that can be used to describe multiple phase clock cycles.

SIMPL has no explicit timing control statements. Implicit timing is accomplished through the use of the "single identity" type program. The compiler recognizes concurrent operations and parallel data paths. Asynchronous operations are programmed using status indicators that are checked by the "if STATUS then . . ." statement.

CASD and MPL also use implicit timing. The compiler recognizes mutually exclusive operations that may be executed concurrently. Each labeled statement begins a block of possible concurrent micro-operations. CASD has the additional features of "DO CONCURRENTLY" and "WAIT" statements that can be used to indicate asynchronous operations.

AHPL uses the letters "S" and "A" at the beginning of a statement to indicate synchronous and asynchronous operations, respectively. Also, in synchronous operations, if more than one statement is written on a line, the statements are executed concurrently. Through the use of English-word operators like "WAIT," "DELAY," "DIVERGE," and "CONVERGE," complicated parallel synchronous operations within asynchronous operations can be specified.

APL has no explicit means for a programmer to indicate timing. Bingham (2) said "(when they used APL to generate microcode) the machine had two overlapping phases so timing was not difficult." A programmer would have to write his own timing functions if he wished to use APL for microcode generation.

2.5 Hierarchical Descriptions

The microprogrammer must be able to describe the microsequence exactly or conceptually. Components defined in terms of primitive operations (OR, AND, NOT) should be named and referred to symbolically in higher levels of the hierarchy. For example this would mean that every time addition is required the microprogrammer does not have to describe the add sequence in terms of AND-OR logic. What he does is write an addition routine and then uses it each time addition is performed.

CDL allows the programmer exact or conceptual microprogram descriptions through the use of operator and terminal statements. The example CDL program in Appendix B demonstrates the use of these statements to describe the AND-OR logic of the ADD function.

CASD, MPL, and SIMPL use PROCEDURE blocks, which may be contained in other PROCEDURE blocks, to build the complete program. The example programs in Appendix B use a PROCEDURE block to describe the addition of the contents of two registers.

APL uses FUNCTIONS to describe the microprogram at different hierarchical levels. The addition of two binary numbers is described by the functions ADDI and ADD1 in the example APL program in Appendix B.

AHPL uses the combined logic subroutine to achieve hierarchical descriptions of microprograms. An example combined logic subroutine that adds two numbers is given in the AHPL program in Appendix B.

2.6 Machine Independence

Machine independence actually means two things when applied to a high level microprogramming language. First, the language must be independent of the particular machines on which it is implemented. Second, the language must be independent of the particular microprocessor it is used to microprogram.

2.6.1. To be independent of the machines on which it is implemented the language must produce identical "object" microcode for a program when the program is executed on the different machines. Assume translators for some high level microprogramming language were available for the IBM S/360 and Honeywell H4200 computers. A program run on each of these machines would produce identical outputs if the language was machine independent.

All the languages discussed in this paper are claimed to be machine independent. APL and CDL are since they have been implemented for a number of different computer systems. Although AHPL has only been implemented on the CDC 6400 there is no evidence that it is machine dependent. Since MPL, CASD, and SIMPL are proposed as high level languages, this implies they are machine independent.

2.6.2. To be machine independent in this second sense, the language must provide facilities to define a machine in which the microprograms are to be implemented. On a particular day a programmer may use the language to write a program for the Interdata 85 and the next day use the same language to write a program for the B1700.

Briefly this means the language must be descriptive. The language must allow the programmer to give a complete description of any microprogram-controlled computer. This requirement was discussed in section 2.2. As mentioned in that section, APL is the only language which lacks means to completely describe a computer's internal structure.

In section 2.2 it was mentioned that the micro-instruction format must be described. There are many different microinstruction formats, and there is no way for a translator to produce microcode for a particular one unless a complete description of it is given. This is discussed more in the following paragraphs.

There are two basic types of microinstruction formats. One, called the horizontal microinstruction format, consists of a number of subfields. A subfield may be from one to several bits long. Different combinations of bits in a subfield cause different control signals to be issued which in turn controls the flow of data.

The second type of microinstruction format is called a vertical microinstruction format. This format is similar to a machine instruction format. It usually has an op-code and some operands, which often means it is much shorter and less complicated than horizontal microinstructions. The vertical microinstruction is generally considered easier to program, but often programs written using it require more time to execute than programs written using the horizontal microinstruction (23, pp. 112-116).

The only languages that have a means to describe these formats are CDL and SIMPL. In my opinion it would be impossible to have a machine independent microprogramming language that did not have a facility to allow descriptions of different microinstruction formats. Eckhouse (10) omitted a means to describe alternate microinstruction format in MPL. This is MPL's main deficiency. Eckhouse presents MPL programs that are supposed to produce microcode for the Interdata 3 and the IBM 2050 processor with little mention of how MPL adjusts for the completely different microinstruction formats. I assume he wrote a different translator for each microprocessor. This means MPL requires a different translator for each microprocessor it will be used to program.

Tsuchiya (23) admits that SIMPL is not completely independent of the computer for which the microcode is to be produced. The sequence of micro-operations that is specified by a particular SIMPL statement may be redefined as necessary. A microprogrammer may modify or redefine the semantics of microinstruction routines in different compilers. This flexibility may be a big advantage as more complicated and different microprogrammable computers become available.

In summary, no language discussed in this report can produce microcode for different microprocessors without some modification of its translator. Since SIMPL has a facility to describe the microinstruction less modification is required.

2.7 Simulator Available

The language should have a simulator available to provide the microprogrammer the facility to test and debug his microprogram before the source code is translated into microcode. A simulator will make it feasible for the microprogrammer to try alternate routines, microinstruction formats, and test microprogram efficiency. The simulator will increase the probability of the microprogram producing optimum microcode.

CDL has a number of simulators available. There are CDL simulators for IBM's 7094 and S/360; Univac 1108; and CDC 6600 (5, p. XIX).

Microprograms written in APL can be simulated on any of a number of APL time sharing systems. Raynaud, *et al.* (19) describes the simulation of microsequences for a computer system. The APL example program in Appendix B was simulated using APL/360.

A partial prototype simulator for MPL was developed as part of Eckhouse's research (10). The other high level microprogramming languages have not had a simulator implemented.

2.8 Translator Available

The language must have a translator that will produce the machine-dependent microcode, which can then be loaded into the control storage of the microprogrammed machine. Since efficiency is of prime importance in microprograms, the translation processor should include optimizing techniques. A translator is, of course, the most important requirement of a high level microprogramming language. Without a translator the high level language will be no more than a microprogram notation that can be used for documentation or possibly as in the case of CDL and APL to simulate the microprogram.

Of the six languages discussed in this paper only MPL and APL have been used to produce microcode. MPL was used to produce a microprogram for the INTERDATA 3 as a test case (10). The final output of the translator was a notation very similar to the INTERDATA 3 micro-assembly language that is used for microprogramming. This MPL translator used optimizing techniques. Eckhouse states that a complete compiler (translator for MPL) has not been implemented.

APL has been used to prepare, translate, and checkout microcode (1). Bingham (2) said their APL programs produced 200 lines of microcode while rejecting 800 lines. The APL programs for this application required three man-weeks to prepare. This translator was for a particular machine and could not be used for general microprogramming; in addition Bingham's APL translator did not optimize the microcode.

3.0 COMPARISON OF THE HIGH LEVEL MICROPROGRAMMING LANGUAGES

The six languages surveyed in this report satisfy the requirements for a suitable high level microprogramming language in varying degrees. We can eliminate some of the languages as possible high level microprogramming languages.

CASD is only a proposed language and there are no plans to implement it. AHPL was not designed as a microprogramming language and lacks means to describe the microinstruction. CDL is an adequate language for simulating microprograms but requires the programmer to explicitly indicate the control bits. He actually has to write the microcode before he can simulate it. For this reason it does not appear feasible to take a language such as CDL and build a translator for it.

MPL satisfies most of the requirements for a high level microprogramming language. It does not have a completely implemented simulator or translator. A simulator is not as critical a requirement as a translator. If and when a complete translator for MPL is implemented the language will still lack the requirement of being machine independent, meaning a different translator will have to be written for each microprocessor.

SIMPL appears to be the most realistic and usable high level microprogramming language. The proposed SIMPL and MODEL languages possess all the facilities needed to write microprograms for both horizontal and vertical microprogrammed computers. The property of allowing the microprogrammer to modify the semantics of a microsequence of the translator appear to make the SIMPL language the best high level microprogramming language. Unfortunately, as mentioned before, SIMPL is only a proposed language and no attempt has been made to develop a translator for it.

APL is the only language that has been used to produce usable microcode. The microprogrammer wishing to use APL must write his own APL routines to translate the APL description into microcode. Computer scientists have critized APL as being hard to read and understand by the novice programmer. This is no real problem because a microprogrammer is not a novice programmer. "(He) must be thoroughly familiar with the design philosophy, the architectural characteristics, and the technological capability that govern the design of (a microprogrammable computer)" (17, p. 16). I feel a specially designed microprogramming language with notations similar to AHPL would be a much better microprogramming language than any of the currently implemented versions of APL.

If I had to pick a currently available language for writing microprograms, it would have to be APL even though it lacks the aforementioned facilities.

What is needed is the implementation of a language like MPL which could be used to microprogram a few computers. If this language proved acceptable, then the implementation of a much more versatile language like SIMPL could be undertaken.

If SIMPL produces efficient microcode as its author claims it can, the use of high level microprogramming languages would become as popular with microprogrammers as high level programming languages are with programmers.

4.0 CONCLUSION

This report has covered the basic characteristics a suitable high level microprogramming language must possess. The language should be natural, descriptive, procedural, and machine independent. It should also have hierarchical structures, timing and concurrent operations, a simulator for the microsequences, and a translator.

Of the six languages compared APL is the only implemented language that has been used to produce microcode (1). The implementation of a translator for MPL would provide the microprogrammer with a language, but he would be limited to microprogramming for a particular machine. The implementation of a language like SIMPL would be a much more difficult task. If it can be shown that a language similar to SIMPL can produce microcode as efficient as is produced by conventional microprogramming methods, there would be more support and interest in implementing such a language.

In my opinion the need for a high level microprogramming language has not been great enough for manufacturers to support the development of such a language. As more microprogrammable computers become available the need may increase to a point where a manufacturer will develop a high level microprogramming language to support their computer. At the present the development of such a language appears to be a good research topic for students in computer science, but not a money-making proposition for manufacturers.

APPENDICES