The Growth of Interest in Microprogramming: A Literature Survey

M. V. WILKES

University Mathematical Laboratory, Cambridge, England

The literature is surveyed beginning with the first paper published in 1951. At that time microprogramming was proposed primarily as a means for designing the control unit of an otherwise conventional digital computer, although the possible use of a read/write control memory was noted. The survey reveals the way in which interest has successively developed in the following aspects of the subject: stored logic, the application of microprogramming to the design of a range of computers, emulation, microprogramming in support of software, and read/write control memories. The bibliography includes 55 papers.

Key words and phrases: microprogramming, microprogram, literature survey, control unit, control memory, read-only control memory, stored logic, emulation, firmware

CR categories: 1.3, 4.9, 6.1

I believe that I was responsible for the introduction of the term microprogramming with its present meaning [1]. My object was to provide a systematic alternative to the usual somewhat ad hoc procedure used for designing the control system of a digital computer. The execution of an instruction involves a sequence of transfers of information from one register in the processor to another; some of these transfers take place directly and some through an adder or other logical circuit. I likened the execution of these individual steps in a machine instruction to the execution of the individual instructions in a program. Hence the term *microprogramming*. Each step is called for by a microinstruction and the complete set of microinstructions constitutes the microprogram. The analogy is made more complete by the fact that some of the microinstructions are conditional.

Figure 1 is taken from [1]. The microprogram is held in a read-only memory here shown as consisting of two diode matrices, matrix A and matrix B. These are to be regarded as corresponding to two fields in the microinstruction. The outputs from matrix A are connected to gates in the arithmetic unit and elsewhere in the computer. The access circuits of the memory consist of a decoding tree with an associated address register. A timing pulse enters the decoding tree and the resulting output from matrix A brings about the appropriate microoperation. The output from matrix B is fed, via a delay circuit, to the address register and thus controls the selection of the next microinstruction. One of the wires from matrix A is shown as branching before it enters matrix B. The direction taken by the pulse at this branch depends on the setting of the sign flip-flop of the accumulator; the choice of the next microinstruction to be executed thus depends on whether that number is positive or negative. In a similar way the sequence of control can be made to depend on the setting of other flip-flops in the processor or

This paper was originally prepared for the ACM Workshop on Microprogramming held at the Mitre Corporation in October 1968.

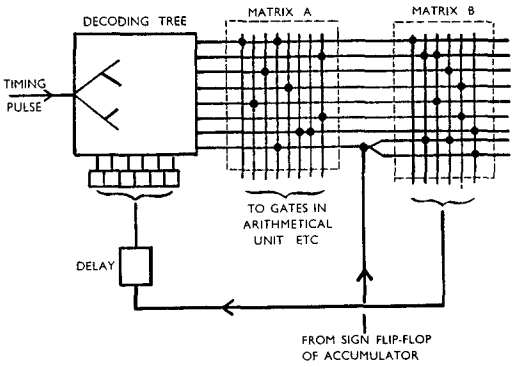

FIG 1

elsewhere. If desired, the branch can be between the decoding tree and matrix A; in this case the microoperation itself, as well as the sequence of control, can be made to depend on the setting of the flip-flop in question.

The Whirlwind computer at MIT made use of a diode matrix in its sequencing circuits, although the analogy with programming was absent; this use of the diode matrix may, however, be regarded as an antecedent to the scheme shown in Figure 1.

The term microprogramming has also been used, notably at the Lincoln Laboratory, to describe a system in which individual bits in an instruction control directly certain gates in the processor. Work by van der Poel [11, 22] falls into this category. Such schemes give the programmer a larger repertoire of instructions than he would normally have. There is not very much in common, however, between this type of programming and the type with which I am primarily concerned in this paper. Comments on the two types of microprogramming were made by Beckman, et al [15].

The ideas outlined in [1] were elaborated in other papers by myself and my colleagues [2, 3, 7, 8] I was personally at pains to specify from the outset that I regarded microprogramming as a method of designing the control unit of an otherwise conventional digital computer, that is, one with a fixed instruction set. Besides being less "ad hoc" than conventional methods, a design based on microprogramming would enable decisions about details of the instruction set to be postponed until a late stage in the construction of the computer, and to be less influenced by exigencies of implementation. It would even be possible to change, or add to, the instruction set after the machine was completed.

The use of a read-write memory instead of a read-only memory to hold the microprogram so that the programmer could set up his own microprogram was mentioned as an intriguing possibility, but I doubted whether a computer designed in that way was really needed. Such computers have, however, been built and much interest is now being shown in them.

Reference [7] describes the control system of a computer built at the University of Cambridge and known as EDSAC 2. The read-only memory was composed of a matrix of ferrite cores through which wires were threaded. The same microprogram matrix controlled all operations of the computer, including fixed-point and floating-point arithmetic operations, the action of the peripheral devices, including magnetic tape, and the internal sequencing of the core memory

Billing and Hopmann published a paper in 1955 in which they discussed general principles of microprogramming and their practical application [4]. Glantz [5] and Mercer [6], who published in the two following years, were mainly interested in microprograms that could be readily altered by the programmer. In 1958 Blankenbaker [9, 12] was interested in a very simple, almost skeleton, digital computer intended to be of theoretical rather than practical interest. In 1958 Dinneen [10] described a computer with a diode read-only memory for the microprogram.

In 1960 Kampe [13] described a microprogrammed computer based, as he put it, on "Wilkes's model in its purest form." He was enthusiastic about the ease and reliability of design using the microprogramming method, although given a simple instruction set, a computer with a conventional control might be somewhat cheaper to build once it had been designed. In 1961

Pinkerton [16] (see also [35]) described an early commercial microprogrammed computer.

The years 1961 to 1964 were peak years as far as international interest in microprogramming was concerned. Not only were there further papers from the United States and one, just mentioned, from the United Kingdom, but papers originating in Italy, Japan, Russia, Australia, and France appeared. There were two Italian papers. Gerace [27] described the CEP computer constructed at Pisa which was based on the use of a ferrite core read-only memory; Grasselli [23] was concerned with the problem of how to construct a stored logic (see below) computer without using a very fast read-write memory. The Japanese paper (Hagiwara, et al [26]) described the use of an original form of read-only memory for holding the microprogram. This consisted of a diode matrix with a diode at each intersection instead of only at selected intersections. Each diode was connected in series with a photo-transistor and could be switched in or out of circuit by illuminating or darkening the transistor. Light was allowed to fall, through a perforated card, on only those transistors that were required to be conducting. Thus the microprogram in use could be replaced by another one by changing the perforated card. The Russian paper (Emelyanov-Yaroslavsky, et al [28]) was a recapitulation of the principles of microprogramming and its possibilities. From Australia came a description of CIRRUS, a microprogrammed computer with quite an elaborate instruction set including floating-point operations and interrupts (Allen, et al [29], see also Allen [14]). The French paper (Harrand [24]) was on the evolution of microprogramming concepts.

During the same period there was a remarkable burst of interest in the United States in *stored logic* computers, in which the designers attempted, within the severe limitations imposed by the technology then available, to give the programmer some control over the choice of the microprogram. The February 1964 issue of *Datamation* contained an introductory article on this subject [30] and no fewer than four articles describing computers with stored logic facilities of various kinds [31–34]. Information about two of these had already appeared in 1961 [17, 18, 25]. It cannot be said that these computers had the success that their designers hoped for; one reason for this may have been that they were seeking to obtain efficiency at the assembly-language level, whereas users were becoming more interested in the use of higher-level languages. However, the interest in stored logic was not confined to designers of small computers. Several papers were presented at the 1961 ACM National Conference on microprogramming aspects of the IBM 7950 [19–21]. This system consisted of a Stretch computer together with a high speed file-processing complex; the latter consisted of three computers relatively independent of the Stretch computer but coupled to it. Various uses were made throughout the system of the stored logic principle; for example, the programmer could set up complex file processing operations which were then called for by a single machine instruction. Essentially the idea was to make use of stored logic in support of a conventional instruction set.

McGee and Petersen [40] drew attention to the advantage of using an elementary microprogrammed computer, or *controller*, as an interface between computers and peripheral devices. They discussed in detail how a microprogram for controlling a film scanner could be written. A much later contribution by Rose [49] discusses a graphical interface system.

By 1964 there began to appear the first signs of the modern interest in microprogramming as a means of designing a range of computers of differing power with compatible instruction sets. These developments reflected the improved performance of capacitive and transformer-type read-only memories that the use of transistors (instead of vacuum tubes) had made possible In the IBM System 360 series [36] all but the largest computer then announced (model 70) had microprogramming based on a read-only memory. In the

following year, a paper appeared describing the RCA Spectra 70 series of computers [37] and of these one of the intermediate models (model 45) was microprogrammed.

1965 and 1966 saw papers on an entirely new subject, namely the emulation of one computer by another [42–44]. Tucker [38] defined an emulator as a package that includes special hardware and a complementary set of software routines. An emulator runs five or even ten times as fast as a purely software simulator. Tucker goes on to discuss the design of emulators for large systems. It is only in very unusual circumstances that it is practicable to write a microprogram that implements directly on the object machine the instruction set of the subject machine; this is because of differences in word length, processor structure, and so on. Tucker recommends that in order to design an emulator one should first study a simulator and see in what areas it spends most of its time. This analysis will generally lead to the identification, as candidates for microprogramming, of a group of special instructions which are related not to specific instructions of the subject machine but rather to problems common to many such instructions. The most important of these special instructions is likely to be one that performs a similar function to the main loop in an interpreter and sends control to an appropriate software simulator for each instruction interpreted. Another will probably be an instruction that performs a conditional test in the way that it is performed on the subject machine. It may also be worthwhile adding special instructions to deal with such instructions of the subject machine as are difficult to simulate. If this procedure is carried to the extreme, the software simulation disappears altogether and we have a *full hardware* feature. Full hardware features are economically practicable only for small machines (McCormack, et al [41]).

Sometimes the design of an emulator can be much simplified if a small change or addition is made to the register interconnection logic of the object machine; an example, cited by Tucker, is the addition of a small amount of logic to the IBM System 360/65 processor in order to facilitate the emulation of overflow detection on IBM 7090 shifts. Such additions (if made) can enable the efficiency of the emulator as a whole to be improved to a useful extent. Sometimes more substantial additions are worthwhile, such as hardware registers intended to correspond to particular registers on the subject machine. By careful design of an emulator it is even possible to handle correctly certain types of function that are time-dependent on the subject machine. McCormack, et al [41] gives an example of a case in which hardware additions to the object machine were necessary in order to enable it, when running under the emulator, to handle data at the rates required by certain peripheral devices. It is generally found that, in order to accommodate an emulator, it is necessary to provide a second section to the read-only memory approximately equal in size to the section that holds the microprogram for the basic instruction set. There is no doubt that emulators will be of great economic importance to the computer industry in the future, and the fact that they can be provided relatively easily on a microprogrammed computer is an argument in favor of microprogramming as a design method.

In 1967 a complete discussion by Tucker of the microprogramming techniques adopted in the design of the IBM System 360 was published [48]. The reasons microprogramming appeals to designers at the present time may be summarized as follows:

1. It provides economical means whereby the smaller machines of a series can have large instruction sets compatible with those on the larger ones.

2. Maintenance aids can be provided; for example, the read-only memory can have a parity bit, and special diagnostic microroutines can be provided for the use of the maintenance engineers.

3. Emulation is possible.

4. Flexibility exists to provide new features in the future.

Since many computers now have microprogramming capacity over and above that required for the basic instruction set, means exist to experiment with microprograms designed to support software in various areas. Opler [45] has suggested the term *firmware* for such microprograms, and he suggests that firmware may take its place along with software and hardware as one of the main commodities of the computer field.

Several papers have already appeared on microprogrammed support for the compiling or interpretation of higher level programming languages [39, 46, 47], and this subject is likely to become very important. Some of the additional or special features that have been provided are designed to complement the basic instruction set by providing additional instructions whose lack the compiler-writer particularly feels. An example is the provision in one of the versions of CPS [46] (which is an interpreter rather than a compiler) of instructions for floating-point decimal arithmetic; the basic instruction-set provides only for floating-point binary. More significant, however, is the provision of instructions for searching lists, manipulating stacks, and evaluating Polish strings. Some of these instructions are quite elaborate and run through many machine cycles. They terminate either when the job is complete, when a count has run out, or when an exceptional situation which can only be dealt with by software is encountered.

Hawryszkiewycz [50] has written on the use of microprogramming support for problem-oriented languages and has experimented with a set of special instructions for the simulation of an analog computer. He reports a three-to-one speeding up of the simulation as a result of providing this support.

In a paper assessing the status of microprogramming in the light of developments in integrated circuits, Flynn and MacLaren [51] point out that the so-called stored logic computers failed to achieve in any general sense the great promise of enabling the programmer to alter the "structure" of the computer. This was because read-write memories of adequate speed and capacity were not available, and the designers had to resort to various expedients that did not really achieve their objectives. Large scale integration should change this situation by making suitable read-write memories available at no very great cost. Flynn and MacLaren propose that such a memory should be used not only to hold the microprogram but also for scratch pad purposes, so that the processor would not need any specialized arithmetic registers. The processor would in fact be a stored (micro)program computer in its own right. They go on to discuss the effects that this would have on the design of assemblers, compilers, and software in general. An important development in this general direction has very recently been described by Rakoczi [55].

Already, computers with read-write control memories to hold the microprogram are beginning to appear. The future of such systems raises issues which it is hard to determine at the present time. It once seemed that they would fill no established need and would lead to major problems in the areas of compatibility and debugging. The situation is perhaps different, however, now that the value of microprogramming support for software has been demonstrated, at least in some cases. In the future, the need for such support may be felt in so many areas—compilation, interpretation, emulation, simulation, operating systems—that very large read-only memories will be necessary to hold the microprograms. If this happens, it will, perhaps, be more economical to provide a relatively small read-write memory into which microprograms can be transferred from core storage when they are needed. If read-write control memories become common, it will, I feel, be on these economic grounds, rather than from a desire on the part of the designers to please the user. The compatibility problem has taken on a different aspect now that we have become accustomed to the idea of a privileged mode in which only programs written by systems programmers can run. Loading of the control memory would, presumably, be possi-

ble only in this mode, and this would perhaps relieve the anxiety of those who feel that giving the ordinary users access to the control memory would lead to chaos.

ACKNOWLEDGMENT

I would like to thank Mrs. M. O. Mutch for the valuable contribution she made to the work of preparing the bibliography.

BIBLIOGRAPHY

1. WILKES, M. V. The best way to design an automatic calculating machine Manchester U Computer Inaugural Conf, 1951, p. 16.
2. WILKES, M. V., AND STRINGER, J. B Microprogramming and the design of the control circuits in an electronic digital computer. *Proc Camb Phil Soc 49* (1953), 230.
3. STRINGER, J Microprogramming and the choice of an order code. Automatic Digital Computation, Proc Symposium held Oct. 1953, NPL, London, p 71, H.M.S O., London, 1954
4 BILLING, H., AND HOPMANN, W. Mikroprogramm-Steuerwerk. *Electron Rundschau 9* (1955), p. 349.
5. GLANTZ, H. T. A note on microprogramming *J ACM 3,* 1 (Jan. 1956), 77–84.
6 MERCER, R. J Micro-Programming. *J. ACM 4,* 2 (Apr. 1957), 157–171
7 WILKES, M. V, RENWICK, W, AND WHEELER, D. J. The design of a control unit of an electronic digital computer. *Proc IEE 105* (1958), 121.
8. WILKES, M. V Microprogramming. Proc. Eastern Joint Comput. Conf., Dec. 1958, Spartan Books, New York, p. 18.
9. BLANKENBAKER, J V. Logically microprogrammed computers. *IRE Trans EC-7* (1958), 103.
10 DINNEEN, G. P., LEBOW, I. L., AND REED, I S. The logical design of CG24. Proc. Eastern Joint Comput. Conf., Dec. 1958, Spartan Books, New York, p 91
11. VAN DER POEL, W. L. Zebra, a simple binary computer. Information Processing, Proc. Int. Conf on Information Processing, UNESCO, 1959, Butterworths, London, p 361
12 BLANKENBAKER, J. V. Logically microprogrammed computers—examples of small computers Information Processing, Proc Int. Conf. on Information Processing, UNESCO, 1959, Butterworths, London, 1960.
13 KAMPE, T W The design of a general-purpose microprogram-controlled computer with elementary structure. *IRE Trans EC-9* (1960), 208.
14 ALLEN, M W. System design of CIRRUS Australian Comput. Conf., Sec. C5.2, 1960.
15 BECKMAN, F. S., BROOKS, F P., AND LAWLESS, W. J. Developments in the logical organization of computer arithmetic and control units. *Proc IRE 49* (1961), 53.
16 PINKERTON, J. M. M The evolution of design in a series of computers, LEO I–III. *Comput. J. 4* (1961), 42.
17. SEMARNE, H. M., AND PORTER, R E A stored logic computer. *Datamation 2,* 5 (1961), 33.
18. SEMARNE, H. M., AND MCGEE, W. C. Stored logic computing. Preprints ACM 16th Nat. Conf., 6C-4, 1961.
19. MEADE, R. M. A discussion of machine-interpreted macroinstructions. Proc. 16th ACM Nat. Conf., 6C-1, 1961.
20. CONROY, E D., AND MEADE, R. M. A microinstruction system. Preprints ACM 16th Nat Conf, 6C-2, 1961.
21. CONROY, E. D. Microprogramming. Preprints 16th ACM Nat. Conf., 6C-3, 1961
22 VAN DER POEL, W. L. Microprogramming and trickology. In *Digital Information Processing,* W. Hoffmann (Ed.), Interscience, New York, 1962, p. 269.
23. GRASSELLI, A. The design of program-modifiable microprogrammed control units *IRE Trans EC-11* (1962), p. 336.
24. HARRAND, Y. Evolution des concepts de microprogrammation. Proc. 3rd AFCALTI Cong. on Computing and Information Processing, Toulouse, France, 1963, p. 187.
25. BOUTWELL, E. O., JR, AND HOSKINSON, E. A. The logical organization of the PB440 microprogrammable computer. AFIPS 1963 Fall Joint Comput Conf., Vol 24, Spartan Books. New York, p. 201.
26. HAGIWARA, H., AMO, K, MATSUSHITA, S., AND YAMAUCHI, H. The KT pilot computer—A microprogrammed computer with a phototransistor fixed memory. Proc. IFIP Cong. 62, North-Holland, Amsterdam, 1963, p. 684
27. GERACE, G B. Microprogrammed control for computing systems *IEEE Trans EC-12* (1963), 733
28. EMELYANOV-YAROSLAVSKY, L. B., AND TIMOFEEV, A. A Microprogram control for digital computers. Proc. 1962 IFIP Cong., North-Holland. Amsterdam, p. 567
29 ALLEN, M. W., PEARCEY, T, PENNY, J. P, ROSE, G A, AND SANDERSON, J G CIRRUS, an economical multiprogram computer with microprogram control. *IEEE Trans. EC-12* (1963), 663.
30. AMDAHL, L. D Microprogramming and stored logic. *Datamation 10,* 2 (1964), 24.
31. MCGEE, W. C. The TRW-133 computer. *Datamation 10,* 2 (1964), 27.
32. HILL, R H. Stored logic programming and applications *Datamation 10,* 2 (1964), 36.
33 BECK, L., AND KEELER, F The C-8401 data processor. *Datamation 10,* 2 (1964), 33.
34 BOUTWELL, E. O., JR The PB440 computer. *Datamation 10,* 2 (1964), 30
35 —— LEO 326 and LEO 360. *Data Proc. Mag.* (Mar.–Apr. 1964), 2.
36. STEVENS, W. Y. The structure of SYSTEM 360, Pt. II System Implementations *IBM Syst J 3* (1964), 136

37 BEARD, A. D. Spectra-70, basic design and philosophy of operation. WESCON/65, Pt. 4, 1965, p. 12 1.

38. TUCKER, S G. Emulation of large systems *Comm ACM 8,* 12 (Dec 1965), 753–761.

39. MELBOURNE, A. J., AND PUGMIRE, J. M A small computer for the direct processing of FORTRAN statements *Comput. J* 8 (1965), 24.

40 MCGEE, W. C, AND PETERSEN, H. E. Microprogram control for the experimental sciences Proc. AFIPS 1965 Fall Joint Comput. Conf., Vol. 27, p 77.

41 MCCORMACK, M A, SCHANSMAN, T T., AND WOMACK, K K. 1401 compatibility feature on the IBM system/360 model 30 *Comm. ACM 8,* 12 (Dec. 1965), 773–776.

42 BENJAMIN, R. I The Spectra 70/45 emulator for the RCA 301 *Comm. ACM 8,* 12 (Dec. 1965), 748–752

43 GREEN, J. Microprogramming, emulators and programming languages *Comm ACM 9,* 3 (Mar 1966), 230–232

44 CAMPBELL, C. R, AND NEILSON, D A Microprogramming the Spectra 70/35. *Datamation 12,* 9 (1966), 64.

45. OPLER, A. Fourth generation software *Datamation 13,* 1 (1967), 22

46. BLEIWEISS, L S., ET AL. Conversational programming system (CPS), IBM Contributed Program Library 360D 03.4.016, 1967.

47. WEBER, H A microprogrammed implementation of EULER on IBM System/360 Model 30. *Comm ACM 10,* 9 (Sept 1967), 549–558

48 TUCKER, S G Microprogram control for System/360. *IBM Syst. J. 6* (1967), 222.

49. ROSE, G. A. Intergraphic A microprogrammed graphical-interface computer. *IEEE Trans EC-16* (1967), 773.

50 HAWRYSZKIEWYCZ, IGOR T Microprogrammed control in problem-oriented languages *IEEE Trans EC-16* (1967), 652.

51 FLYNN, M J, AND MACLAREN, M D. Microprogramming revisited Proc ACM 22nd Nat. Conf, 1967, p. 457.

52. HUSSON, S S Microprogramming manual for the IBM/360 mod 50. Tech Rep. TR00 1479-1, IBM Corp, 1967

53 DREYER, L Principles of a two-level memory computer *Comput Autom. 17* (1968), 40.

54 LAWSON, H W. Programming-language-oriented instruction streams. *IEEE Trans. C-17* (1968), 476

55 RAKOCZI, L. L. The computer-within-a-computer, a fourth generation concept. IEEE *Computer Group News 3,* 2 (1969), 14