JOHANNES KEPLER
UNIVERSITÄT LINZ

TECHNISCH-NATURWISSENSCHAFTLICHE FAKULTÄT
INSTITUT FÜR INFORMATIK

FIRMWARE AND MICROPROGRAMMING IN HIGH-END COMPUTERS

G. Chroust[+]

SYSPRO 18/81 Nov. 81

INFORMATIK-BERICHTE

A-4040 LINZ/AUHOF, ALTENBERGERSTRASSE 69
ÖSTERREICH/AUSTRIA

FIRMWARE AND MICROPROGRAMMING IN HIGH-END COMPUTERS

G. Chroust[+]

SYSPRO 13/81 Nov. 81

Research for this paper has been partially supported by the Austrian Fonds zur Förderung der wissenschaftlichen Forschung under project 3489.

This paper has been accepted for publication in SEAS-Tagung, Nizza 1981. As a courtesy to the publisher its distribution is strictly limited.

[+]Univ.Doz.Dr.G. Chroust
IBM Labor
Cobdengasse 2
1010 Wien

F I R M W A R E   AND   M I C R O P R O G R A M M I N G

IN HIGH-END COMPUTERS

G. Chroust
IBM Laboratory Vienna, 1010 Vienna, Austria and
Joh. Kepler University Linz, 4040 Linz, Austria

1.0 THE CONTROL UNIT OF A COMPUTER

1.1 BASIC STRUCTURE OF A COMPUTER SYSTEM

The lowest level in a computer which is visible to the general user, has traditionally been called the '_machine language level_'. Usually all programs written in a higher-level language (e.g. PL/I) are _translated_ into instructions of this machine language. The machine language consists of very simple instructions (e.g. load a register, add two numbers); more complex functions have to be expressed by these instructions. Machine instructions themselves are directly _executed_.

Fig. 1 shows the model of a very simple execution unit. The program and its data reside in Main Memory. One machine instruction after the the other is fetched from Main Memory, specifying the details of the execution: which data are to be fetched, combined and stored. Physically this execution is done by sending control signals to the proper control points (numbers in Fig. 1). In order to perform an addition, for example, the address of the first operand is brought into the Storage-Address-Register (SAR, control point 1). Activating control point 2 causes the corresponding value to be brought into the Storage-Data-Register (SDR). Subsequently this value and the value of the accumulator are routed into the Adder (control points 6 and 7). After addition (control point 8) the result is gated back into the accumulator (control point 10). _How_ the control signals are generated by the Control Unit is not shown in Fig. 1.

The task of the _control unit_ is to generate a meaningful sequence of control signals which are to be sent to the control points of the execution unit.

Historically two methods have been used to implement the Control Unit:

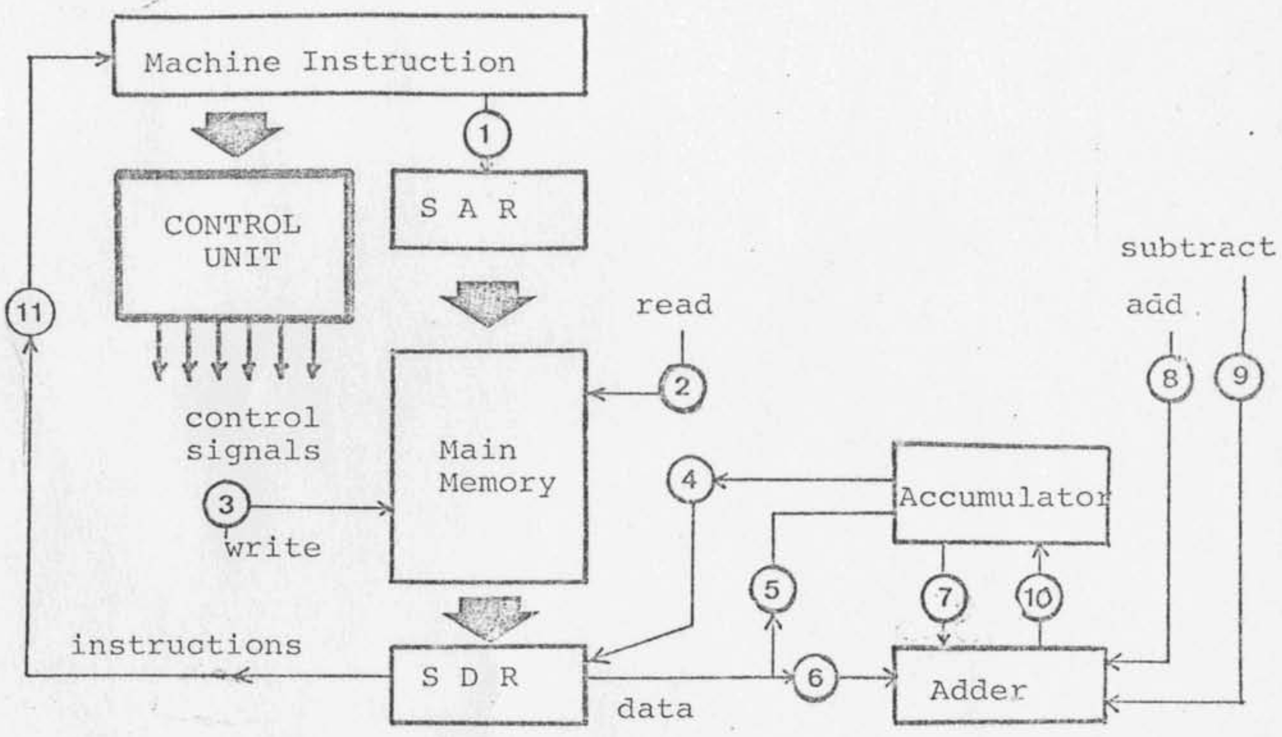

Fig. 1: Internals of a very simple computer

- Hardwired Logic:
  This is the initial, classical method. Starting from an agreed-upon set of machine instructions a switching network was constructed using the method of switching algebra. The network is constructed from AND, OR, NOT gates and delays, and is 100% taylored to the initially specified instruction set.

  The complexity of the network grows more than linearly with the size of the instruction set. A network of this kind is difficult to develop, its design is error prone, it is almost impossible to verify, it is hard to maintain and difficult to be adapted to new requirements.

- Systematic Design (Microprogramming)
  In 1951 Prof. Maurice Wilkes, Great Britain, (/Wilkes-a/) proposed a different method for designing the control unit (Fig. 2): He later commented (/Wilkes-b/) on it:
  "My objective 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; ... 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."

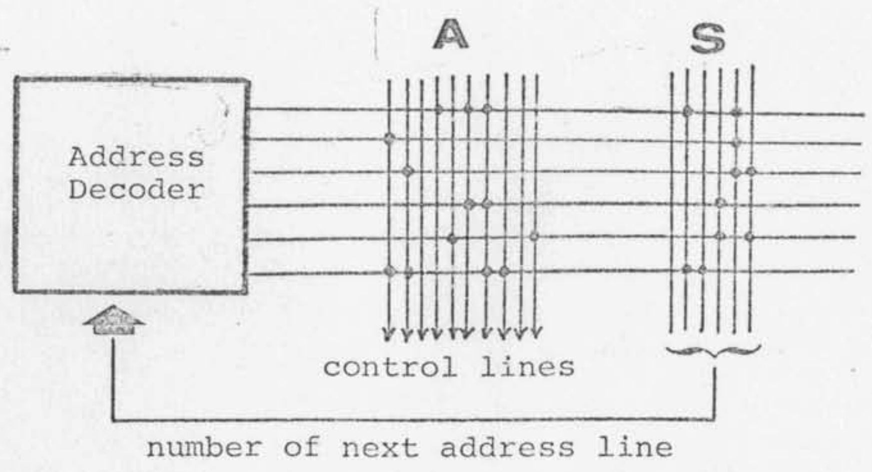

Fig. 2: Microprogramming Scheme by M. Wilkes (1951)

In his proposal the control signals are generated via matrix A: Exactly one vertical line ('control line') corresponds to each control point. During each machine cycle exactly one horizontal line ('address line') is activated by the decoder. Those control lines which should receive a control signal during this cycle are connected to the address line (heavy dots in matrix A, diodes in the original proposal). Matrix S identifies the next address line to be activated.
Wilkes' proposal is a very systematic, tabular, method for implementing a control unit: each control line must appear in the scheme and every possible state transition is specified by a connection in matrix A. To each address line all those control lines are connected which have to be activated together.

The diode matrices were later replaced by a control storage (Fig. 3): each word of a control storage (microinstruction) corresponds to the contents of one address line. The individual bits of the microinstruction are associated with the individual control line (e.g. 0 or 1 for 'not active' or 'active', respectively).

In microprogrammed machines each machine instruction is interpreted by a sequence of microinstruction (a microprogram).

The importance of Wilkes' idea is to make the hardware independent of the machine instructions. It is 'only' necessary to write the correct microprograms and load them into the control storage. Ganzhorn (/Ganzhorn/) called this a

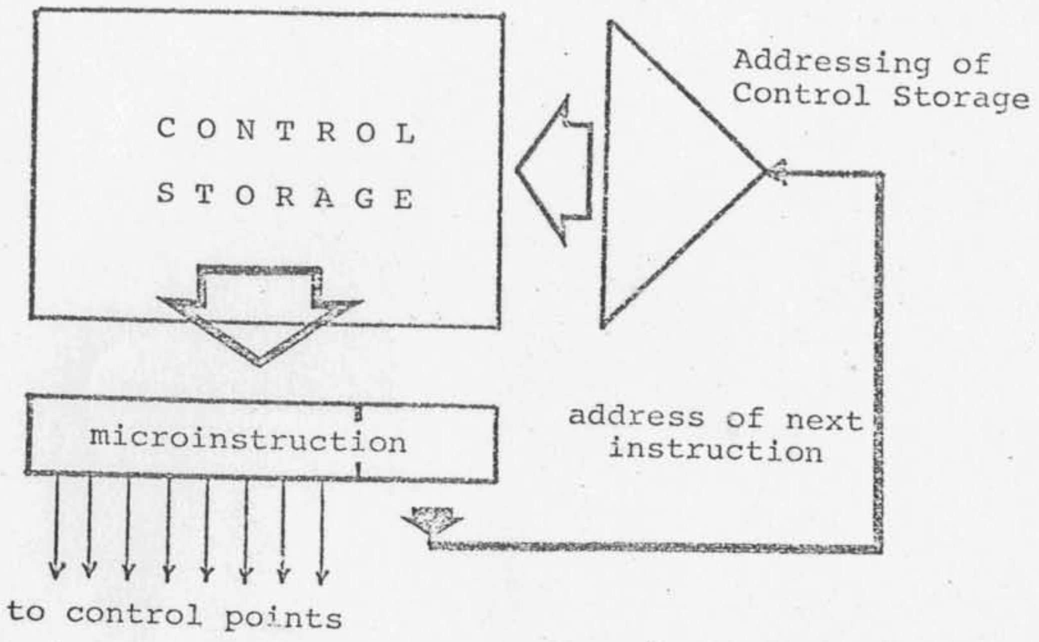

Fig. 3: Hardware Structure of a microprogrammed Machine

'stored function'. Hardware and machine instructions are separated by an indirect step: the microprogram.

2.0 THE GROWTH OF MICROPROGRAMMING

2.1 IBM SYSTEM/360

In 1951 Wilkes' idea did not rouse much interest. His proposal was too expensive, too slow and not cost effective enough. In 1960, however, technological progress in storage technology and economic needs revitalized the concept: At that time a considerable number of - incompatible - computer models were on the market (IBM 650, IBM 704, IBM 1401, ...). Each model had different price/performance characteristics and, more importantly, a completely different machine language. For these computers considerable software libraries had been developed, unfortunately almost exclusively in machine language (of the respective model!). If a user wanted to change to a different model, it meant almost complete reprogramming. In many cases, even the source decks did not exist any more (/Tucker-a/).

In this situation IBM decided to make once again an incompatible change, but - at the same time - to create a way of avoiding this problem in the future. The result was the IBM System /360. A _family_ of computers, from small to large, was to be offered. Each should have the same _machine_ architecture (/Amdahl/, /IBM-a/), i.e. the same machine instructions, the same registers, etc.. A program conforming to this architecture should run on all models of the computer family. The compatibility was restricted to the architecture. Underneath the machine architecture completely different hardware was to be used (from smaller and inexpensive up to very fast and expensive ones).

The only reasonable method for implementing a machine architecture of this type on a set of radically different hardware machines was to introduce the indirect step of microprogramming (/Husson/, /Tucker-a/).

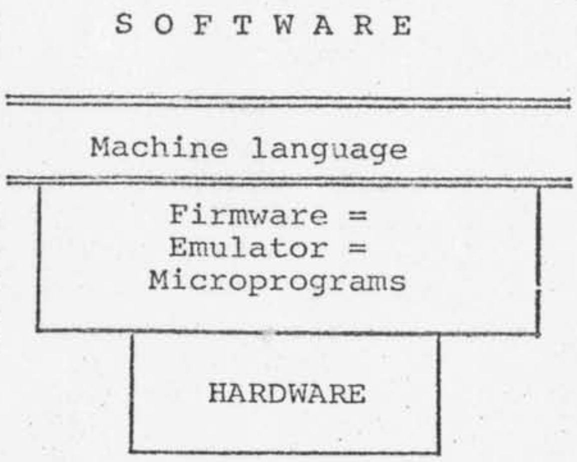

Fig. 4: Microprogramming (Firmware) Layer

| Microprogramming is used to define and to implement a machine architecture independently of the underlying hardware. The microprograms are said to emulate the machine architecture. |

2.2 FIRMWARE

In 1967 (15 years ago!) A. Opler (/Opler/) recognized the potential of microprogramming and coined the term Firmware: He said:
"I use this term to designate microprograms resident in the computer's control memory, which specializes the logical design for a special purpose, e.g. the emulation of another computer. I project a tremendous expansion of firmware - obviously at the expense of hardware but also at the expense of software."

3.0 THE SIGNIFICANCE OF MICROPROGRAMMING AND FIRMWARE

3.1 DEVELOPMENT PROCESS

The development of a new computer system requires hardware and software to be developed in parallel. System software is still mostly written in machine language. The interface between hardware and software is the machine language. For hardwired-logic machines this interface must be frozen rather early in the design cycle. Later changes are difficult and expensive. For microprogrammed machines, the hardware and the machine architecture are separated by the intermediate firmware layer (/Berndt/). Changes in the hardware and in the machine architecture can easily be accomodated in the firmware, even at a fairly late stage.

3.2 SMALL MACHINES

Firmware allows rather small hardware machines also to be equipped with a rich machine architecture, containing many machine instructions (cf. Fig. 5). Examples are the low-end models of the /370 family. The emulation of the architecture is done via large microprograms, thus trading hardware cost for speed.

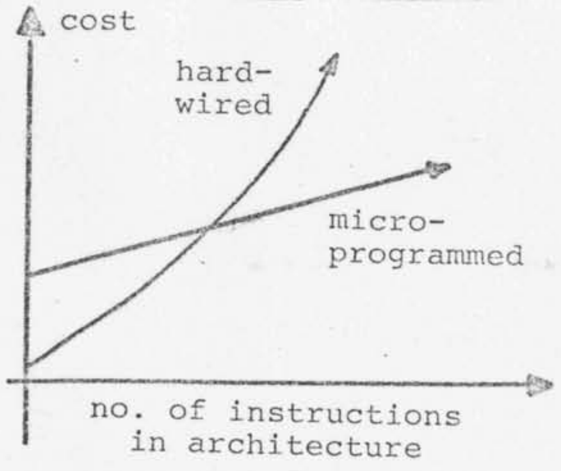

Fig. 5: Cost comparison

3.3 HARDWARE COSTS

With an increasing number of machine instructions to be implemented for a machine architecture a microprogrammed solution is more economical (/Husson/), cf. Fig. 5. Increasing the number of machine instructions only means a growing size of the control storage, whilst the remaining elements (cf. Fig. 3) stay the same.

3.4 ABSORPTION OF COMPLEXITY

Certain functions are easily implemented in either software or hardware, but not in both. By its dual nature firmware is able to absorb such complexities (/Reigel).

3.5 PROGRAMMING INSTEAD OF HARDWARE ENGINEERING

For hardwired machines the realisation of the machine architecture was the domain of hardware engineers, who worked with state tables, transition diagrams, and switching theory to design the Control Unit. Microprogramming is a programming technique (/Chroust-a/, /Davidson/, /Husson/). Hardware engineers have only to provide the very general hardware structure as shown in Fig. 3. Since both sides of the machine architecture interface are programmed, a much better matching can be achieved. In addition software tools can be applied to firmware development (/Davidson/).

3.6 SOLVING THE COMPATIBILITY PROBLEM

Machine architecture is based on the contents of the control storage, which can be reloaded with different microprograms: for example with the emulator for an machine architecture of an older computer (e.g. /360 Mod. 25 emulation on /370 Mod. 115). Thus programs of an 'old' architecture can be run on the new hardware.

3.7 SECURITY

Firmware is not accessible to the general user. This gives an additional security margin against inadvertent or malicious access or change when compared to functions implemented in software.

3.8 MAINTENANCE AND MICRODIAGNOSTICS

The advent of dynamically loadable control stores allows for the diagnostic routines to be loaded only when needed. They normally reside on some external medium. Since microprograms are nearer to the hardware they allow a much finer resolution of malfunctioning units (/Bartow/).

3.9 FLEXIBILITY OF IMPLEMENTATION LEVEL

The flexibility of firmware allows, even after delivery of the machine, to move functions to/from firmware. Moving functions from software into firmware attracts increased attention under the term 'Vertical Migration' (see below).

4.0 SOME TECHNICAL TERMS

4.1 CONTROL STORAGE TECHNOLOGY

Initially control storage was read-only (called 'ROM' or 'ROS'), in a variety of technologies (cf. /Husson/, /Painke/). Today control stores are built exclusively in semiconductor technology (/Sc-Am/). In most cases they are writeable control stores ('WCS'), which can be reloaded/changed during machine operations. This will have far-reaching consequences in that microprograms and thus machine architectures can be changed dynamically (/Tucker-b/, /Wilner/).

4.2 HORIZONTAL AND VERTICAL MICROINSTRUCTION FORMATS

Large machines have many internal units (adders, multipliers, shifters,...) which must be controlled by bits in the control words. Smaller and cheaper machines, on the other hand, have only a few units to be controlled. The width of the microinstruction is a major cost factor (cf. Fig. 3). Therefore, for small machines, one tries to reduce the width of the microword by encoding the information (/Hoff/): several control lines have to share to the same bit position, an extra field controls the distinction. Naturally this reduces the flexibility of the microcode and introduces a more complex decoder, thus trading speed for cost. Thus we see wide ('horizontal') microinstructions in high-end machines and short ('vertical') microinstruction in low-end machines.

4.3 NANOPROGRAMMING (PICOPROGRAMMING)

For various machines it is advantageous to introduce two levels of microprogramming. The higher level is vertically microprogrammed. Each microinstruction of this level is interpreted by the lower-level microprogram, usually called nanoprogram. The nanoprogram is invariably in horizontal format.

5.0 VERTICAL MIGRATION +)

The level in the hierarchy of a computer (/Berndt/, /Reigel/) at which a function is implemented, is governed by several influences: tradition, speed, frequency of use, implementation cost, maintainability/changeability, control storage size, legal requirements, etc.

In the last few years a strong trend can be observed to move function from software into firmware (/Chroust-a/, /Richter/, /Stankovic/, /Stockenberg/). In IBM terminology such migrated functions are called Assists. Candidates for Migration are

- heavily used functions of the operating system (/IBM-b/, /Olbert/),
- Support for higher-level languages, e.g. APL (/Hassit/),
- Monitoring of system performance, e.g. page faults, traces (/Chroust-c/, /Svobodova/).

+) A more detailed discussion can be found in G. Chroust: 'Firmware Support in High-end Computers' in this conference.

6.0 SUMMARY

Despite the fact that microprogramming (Firmware) eludes rigorous definition (/Chroust-b/), it has established itself as a key technology in implementing today's computers. In low-end computers it enables the realization of rich machine architectures on comparatively lean hardware. For high-end machines it offers a flexible way to increase and tune the system performance. For the complete range of models it permits the implementation of a uniform, compatible machine architecture providing a system family of computer models with widely differing cost/performance factors. Being an indirect step between software and hardware, microprogramming increases the hierarchical structure of a computer system and thus allows a more orderly and systematic design.

7.0 REFERENCES:

/Amdahl/ Amdahl G.M., Blaauw G.A., Brooks F.P. Jr.: Architecture of the IBM System/360.- IBM J. of Res. & Dev., vol. 8 (1964), No. 2, pp. 87-97;

/Bartow/ Bartow N., McGuire R.: System/360 Mod. 85 micro-diagnostics. SJCC 1970, Proc. AFIPS vol. 36 (1970), pp. 191-197.

/Berndt/ Berndt H.: Was ist Firmware? Elektron. Rechenanlagen 19 (1977), No. 2, pp. 77-80.

/Chroust-a/ Chroust G., Mühlbacher J. (eds.): Firmware, Microprogramming and Restructurable Hardware.- North Holland Publ. Comp. 1980

/Chroust-b/ Chroust G.: Microprogramming - An Interface Property.- EUROMICRO Newsletter vol. 2 (1976), No. 4, pp. 48-53.

/Chroust-c/ Chroust G., Kreuzer A., Stadler K.: A Microprogrammed Page Fault Monitor.- MICROPROCESSING and MICROPROGRAMMING, vol. 8 (1981) accepted for publication.

/Davidson/ Davidson S., Shriver B.D.: Firmware Engineering: An Extensive Update.- in: Chroust G., Mühlbacher J. (eds.): Firmware, Microprogramming and Restructurable Hardware, pp. 1-40; North Holland Publ. Comp. 1980.

/Ganzhorn/ Ganzhorn K.: Mikroelektronik in Computern.- IBM Nachr. vol. 24 (1975) no. 222, pp. 240-247.

/Hassit/ Hassit A., Lyon L.E.: An APL Emulator on System/370.- IBM System J. 1976, No. 4, pp. 358-378. Symp. 1979.- North Holland Publ. C. 1979, pp. 285-293.

/Hoff/ Hoff G.: Design of Microprogrammed Control for General Purpose Processors.- SIGMICRO Newsletter vol. 3 (1972), no. 2, pp. 57-64.

/Husson/ Husson S.S.: Microprogramming - Principles and Practices.- Prentice Hall, Englewood Cl, 1970.

/IBM-a/ IBM Corp.: IBM System/370: Principles of Operation.- Form No. GA22-7000.

/IBM-b/ Virtual-Machine Assist and Shadow-Table-Bypass Assist.- IBM Corp. Form No. GA22-7074, 1980.

/Olbert/ Olbert A.G.: Extended Control Program Support: VM/370.- SIGMICRO Newsletter, vol. 9 (1978) No. 3, pp. 8-25.

/Opler/ Opler A.: Fourth Generation Software.- Datamation, Jan. 1967, pp. 22-24.

/Painke/ Painke H.: Der Festwertspeicher in digitalen Rechenanlagen.- IBM Nachrichten 16 (1966), no. 176, p. 73-79.

/Reigel/ Reigel E.W.: At the Programming Language - Microprogramming Interface.- R.L. Wexelblat (ed.): Programming Languages - Microprogramming (ACM SIGPLAN/SIGMICRO Interface Meeting 1973), ACM, 1973.

/Richter/ Richter L.: Vertikale Migration - Anwendungen, Methoden und Erfahrungen.- Tagg. 'Hardware fuer Software', Konstanz, Okt. 1980, Verlag Teubner, pp. 9-28.

/Sc-Am/ Scientific American (ed.): Microelectronics - Special Issue.- Scientific American vol. 237 (1977), no. 3.

/Stankovics/ Stankovics J., Weidner T.: The Migration of Primitives (Summary of a Panel Discussion).- in: Chroust G., Mühlbacher J.R. (eds.): Firmware, Microprogramming and Restructurable Hardware.- North Holland Publ. Comp. 1980, pp. 213-215.

/Stockenberg/ Stockenberg J., van Dam A.: Vertical Migration for Performance Enhancements in Layered Hardware / Firmware / Software Systems.- Computer vol. 11 (1978), No. 5, pp. 35-50

/Svobodova/ Svobodova L.: Computer Performance Measurement and Evaluation Methods: Analysis and Applications.- Elsevier Publ. C. 1976

/Tucker-a/ Tucker S.: Emulation of Large Systems.- Comm. ACM vol. 8 (1965), no. 12, pp. 753-761.

/Tucker-b/ Tucker A.B., Flynn M.J.: Dynamic Microprogramming - Processor Organization and Programming.- Comm. ACM 14 (1971), no. 4, pp. 240-250

/Wilkes-a/ Wilkes M.V.: The best Way to design an Automatic Calculating Machine.- Manchester Univ. Computer Inaugural Conf, Manchester, July 1951, pp. 16-18.

/Wilkes-b/ Wilkes M.V.: The Growth of Interest in Microprogramming: A Literature Survey.- Comp. Surveys 1 (1969), No. 3, pp. 139-145.

/Wilner/ Wilner W.T.: Design of the B1700.- AFIPS(ed.): Proc. Fall Joint Compter Conference 1972, vol. 41, part 1, pp. 489-497.

100664710