KFKI-1982-18

F. VAJDA

RECENT DEVELOPMENTS IN THE APPLICATIONS OF MICROPROGRAMMING

Hungarian Academy of Sciences

CENTRAL RESEARCH INSTITUTE FOR PHYSICS

BUDAPEST

KFKI-1982-18

RECENT DEVELOPMENTS IN THE APPLICATIONS OF MICROPROGRAMMING

F. Vajda

Central Research Institute for Physics
H-1525 Budapest 114, P.O.B.49, Hungary

To be presented on the 13. WASCO-TAGUNGS /March 31-April 3, 1982/ Budapest, Hungary

HU ISSN 0368 5330
ISBN 963 371 901 1

ABSTRACT

The paper provides a brief mosaic-like summary of the basic issues of microprogramming and the microprogrammed systems. It surveys the present status of the application areas of microprogramming.

АННОТАЦИЯ

В статье дается короткий обзор основных публикаций и сообщений в связи с микропрограммированием и микропрограммируемыми системами. Затрагивается также вопрос настоящего состояния в области применения микропрограммирования.

KIVONAT

A cikk mozaikszerüen röviden összefoglalja a mikroprogramozás és mikroprogramozott rendszerek alapvető problémáit és a mikroprogramozás alkalmazási területeinek jelenlegi helyzetét.

1. INTRODUCTION

30 years have gone by since 1951 when Professor M.V.Wilkes coined the term microprogramming [1] and used it to describe a new systematic method for designing the control unit of a digital computer. In its first decade *computer scientists* discovered microprogramming and published their findings. Then it was rediscovered by *computer specialists* and applied in computer system design and implementation. This time it is being learned by *a lot of people* who should use and program bit-sliced microprocessors. Recently, microprogramming has also been discovered as an important mean of *microelectronic* integrated circuit implementation.

The main *application areas* of microprogramming can be grupped as it follows:

1. *Computer manufacturers* who implement more and more primitives and functions for improving realiability (by microdiagnostics) privacy and data security (by data protection) and propriety protection etc. by microprogramming. In the same time they support user (dynamic) microprogramming of their systems.
2. *"Traditional" users* of user-microprogrammable computer (with WCS and support software) who recently have found the possibilities of microprogramming.
3. *Research works* closely connected to microprogramming, e.g. firmware engineering, dynamic, high-level language oriented computer architectures, tuning and restructuring of computer architecture etc.
4. Designers in the former areas of the *traditional digital design* who apply bit-sliced microprocessor families and firmware supported mainly by the component manufacturers.
5. Applications for *component design and implementation* by the computer (e.g. DEC:T-11) and component (e.g. Motorola: 68000) manufacturers for high-end microprocessor and support chip realization.

The history of microprogramming is naturally included in the history of computer technique and application. As an example let me quote the development of the *implementation of floating-point instructions:*

- First they were performed by subroutines in the frame of a *floating point package* (software)
- Later they were (partly) implemented in *circuitry* (hardware)
- Afterwards the *microprogrammed* versions of floating point instruction sets have appeared (firmware)
- In the up-to-date computers an independent *processor* (floating point processor) executes these instructions (firmware and parallel processing)
- Hereafter standard VLSI *floating-point chips* will perform these functions (microelectronics)

2. LEVELS OF A COMPUTER SYSTEM AND THE MICROPROGRAMMED CONTROL

A set of conceptual levels for describing, understanding, analyzing, designing and using computer systems was defined [2]. There are at least five *levels of system description* that can be used to describe a computer:

- *Device level* (p-type and n-type semiconductor, dielectric material and metal formed in various ways)
- *Circuit level* (transistors, resistors, capacities etc. forming inverters, multivibrators etc.)
- *Logic (switching circuit) level* (combinational and sequential logic functions)
- *Register transfer level* (registers and other functional units e.g. multiplexers, ALUs, counters etc. and the functional transfer between them)
- *Processor-memory-switch (PMS) level* (CPUs, memories, I/O etc.)

In contrast to this *structural view* another more *functional multilevel structure* can also be defined. According to this view, a computer system consists of *layers of interpreters*, such like the layers of an onion, i.e.:

- Logic (hardware) level
- Microprogram (firmware)level
- Machine instruction (architecture) level
- Operating system level } (software)
- Programming language level
- Job control level

In this view *firmware* [3] is used for establishing a *programmable level* between the machine architecture (instruction set) and the real hardware. More recently, firmware also implements certain functions of the real hardware level for *performance* reasons and for added *flexibility*.

Some layers are bypassed when more ideal primitives exist at deeper (i.e. firmware or hardware) layers (vertical migration) [4].

Considering microprogramming as a means of computer *control implementation* we should reconsider the design tradeoffs on computer performance. These tradeoffs can be groupped into three categories:

- circuits technology,
- data path topology and
- control unit implementation.

Attention here is focused on the CPU. Processor families can be categorized as low, medium and high performance implementations. *Technology push* has been strongest in the areas of logic and memory. Relevant parameters of digital *integrated circuits* are: functions, density, cost, speed, number of pins, realibility and speed-power product.

The main trend in the current *applications of the logic components:*

1. In *low and very low end CPUs*: single chip custom design (with built-in microprogammed control)

2. In *medium performance CPUs*: semicustom design (gate arrays and ULAs: Uncommitted Logic Arrays)

3. In *high end computers:* very fast (e.g. ECL) logic.

In semiconductor *memory chips* the three main parameters are: increasing bit density, increasing speed and decreasing price.

These factors have made microprogrammed control an increasingly attractive and *cost-effective choice* for the computer and lately for the microcomputer designers.

The heart of the *data paths* of a CPU is the arithmetic logic unit (ALU) trough which all data circulates and where most of the processing actually takes place. Significant differences exit among the data paths of the different range of machines. One major differences is in the *width* of the data paths. The second feature is the *speed* of the data paths which is a direct result of implementation i.e. the circuit technology employed. The *structure and number* of data paths are the main parameters affecting performance. Set of data paths allows highly parallel operation. A CPU with horizontally microprogrammed control unit can perform more in a given microcycle and thus needs less microcycles to complete an instruction. The structure of the microinstructions for taking best advantages of instruction features can also decrease the effective microcycle count.

Considering the impact of special semiconductor components on microprogrammed control unit design, programmable logic arrays (PLAs) have an increasing role in implementations of both the functional and the control parts.

3. FIRMWARE ENGINEERING

With the growth in microprogramming came the requirement for better microprogram production techniques. The major reasons for this need are the following ones:

- It is very *expensive* to develop microprogram without a set of proper tools.
- Erros in applications are *expensive to repare*.
- New application areas (migration, microdiagnostics, high-level language support etc.) have led to *large microprograms*.
- The availability of microprogrammable processors (with writable control stores, WCSs) is grown *user microprogramming* in traditional sense.

- Microprogrammed control has been in wide-spread use by the availability of bit-sliced microprocessor families *outside the territory of the traditional computer techniques.*

Approaches to firmware engineering [5,6] generally draw from experience and knowledge in

- *microprogramming* (microprogramming techniques, microprogramming languages, simulators, debuggers, test system, etc.)
- *hardware engineering* (design techniques, hardware description languages, simulators, test systems and methods etc.)
- *software engineering* (design, specifications, verifications, test and programming techniques, specification, design and implementation languages, verification and test methods, etc.)
- *system design methodologies* (methods, disciplines, concepts, principles)

Since much firmware and its development is considered to be *proprietory*, many firmware engineering techniques have not been published in the open literature.

Considering the *phases of microprogram development* in the stages of system life cycle we could find six main periods [5].

3.1 Design and specification

Techniques of software engineering [7] (e.g. top-down design, stepwise refinement, structured walk-throught etc.) are used but none meet all the criteria of formality, constructability, comprehensibility, minimality, wide range of applicability and extensibility.

3.2 Construction techniques

The widely accepted design tools for aiding the actual writing of microprograms are the *microassemblers* and there are no widely accepted higher level microprogramming languages.

Lately there are meta-assembler available, a design to be usable for a number of microprogrammable machines or bit-sliced microprocessors. Target definions can be written at the assembler directive level for these machines. Other features include macro definition capability, specification of default values, self documenting etc.

There is an increasing number of research papers dealing with definition of higher level microprogram languages and with the design of *code generators and optimizers* [8] for the system which means *microcode compaction* or microcode improvement. Most part of the code improvement methods are basically *local* techniques [9] but recently more general *global* compaction techniques [10] are also presented.

The basic problem with HLL is how to include machine *dependent* information to machine *independent* language. Several machine dependent HLL have been designed [11], but programs written in these languages are *not transportable.*

3.3 Verification

The work in the area is based for the most part on work done on the verification of software programs [12]. There are two major classes of verification methods. In the first one assertions are made about the *effect of the execution* of a number of microinstructions while in the second class the verifier attempts to find a *simpler microprogram* that is equivalent to the microprogram to be proved [5].

3.4 Testing

In order to test microprograms effectively, the firmware tester should be able to do the following:

- *Display* the state of the machine resources (i.e. all registers, control and primary store etc.)
- *Modify* some part of the processor state (registers, control and primary store) easily.
- Define *breakpoints* to halt execution at preselected points for examination of the processor state.
- Generate a *record* of execution to allow the examination of the experiment off-line by the record of the execution paths.

3.5 Debugging

Debugging is done by employing either *hard* [13] or *soft* techniques [14]. The former ones are utilizing the *host machine* (resident debugger) while the latter ones are using a *simulator*. There are a couple of advanteges of the resident debuggers. There is no problem - using the actual hardware - with the accurate representation of the target machine for the simulator and they can run on a stand-alone configuration.

3.6 Maintenance

The two main categories are: repair and updating. Despite of the fact that firmware maintenance could be *very expensive* somehow it is a neglected field. It can be supported by a better *documented* microprogram and *field tools* for retesting and/or reverification the repaired/new version.

4. APPLICATIONS OF MICROPROGRAMMING

Interpretation of machine language instruction continues to be the most common application of microprogramming. Here we discuss a few application and research areas.

4.1 Emulation

As we use the term emulation [13] refers to using firmware (often assisted by a hardware and software), in one machine to execute programs originally written for another. We call the machine which is realized by the emulator a *target machine* and the machine which supports microprograms a *host machine*.

Work concerning emulation can be groupped as it follows[15]:

- Emulation of *traditional mini- or microcomputers* (mainly research or student projects for learning microprogramming fundamentals).

- For *software compatibility* purposes a new computer family could emulate the former one (e.g. IBM System/360 for the IBM 1400 series or IBM 7090, the DEC VAX system for the PDP-11 family etc.)
- *Emulation oriented* machines, with other words universal emulators (e.g.Q machine).
- Emulation of *new architectures* for research purposes.

4.2 Executing Higher Level Language Programs

The implementation of machines which support high-level languages via a mixture of software-firmware techniques is a well-established technology.

There are different *approaches to support* a high level language on a machine [16]:

- *Traditional approach* i.e. *one* machine language *for all* high level languages (no optional program representation)
- *One* machine language *for each* high level language (e.g. Burroughs B1700S language for supporting Cobol, RPG and Fortran).
- *Extended instruction set* for supporting a language (e.g. DEC CIS: Commercial Instruction Set for supporting Cobol or VAX-11 for supporting Fortran).
- *Direct execution* of the high level language (e.g. Western Digital Pascal Microengine).
- *Multiple* machine language for a high level language which actually involves the development of several machine languages to find the best support environment [17].
- Compilation to an *intermediate level* and the direct execution of the code of this level (e.g. Pascal p-code).

4.3. Operating system enhancement

In order to improve operating system features by implementing primitives in microcode is one of the main application area of the method of vertical migration [4].

For example to support the OS/VSZ MVS (Multiple Virtual Storage) operating system of IBM, a microcode feature called CPS:MVS is provided that enhances the operating system in the area of lock management, integrity, tracing, interrupt handling and real storage management. By the technique of microcode assist the operating system overhead could be reduced by more than 70% in some cases [19].

4.4 Microprogrammed monitoring

Monitors are well-known tools for observing certain aspects of computer systems. A firmware or microprogrammed monitor is a tool whose sensor and selector part at least are implemented in the firmware level of the system to be measured.

There are different objects (and objectives to be pursued) that can be observed by these tools [20].

- To determine the kind of *instruction* used or the *memory area* which is *referenced* by instructions (performance improvement)
- To check the *result* of a microprogram (debugging support)
- *Interface* of the hardware level (microdiagnostics)
- To monitor *software functions* (for supporting vertical migration decisions)

Recently microdiagnostics is very popular and widely used by computer manufacturers for improving *realibility and maintenance* of computer systems.

4.5 User-microprogrammable computers

There are an increasing number of user-microprogrammable computer available on the market. (e.g. DEC PDP-11/60, HP 21MX, HP 1000 etc.)

For user (dynamic) microprogramming, they should have the following features:

- A *Writable Control Store* (WCS) with extra address space in the control store
- Means to *load, dump and check* the WCS
- Program *access* to microcontrol (bus or I/O port, special instructions, trap, microbreak etc.)
- *Support* software (microassembler, micromonotor etc.) for user microprogramming.

For simplifying microprogramming they usually apply simle (vertical) microinstruction sets which is a real bottle-neck for writing efficient (fast) microprograms.

Typical *applications:*

- Instruction *set extensions* with special functions
- Microcoding *application kernels* (10% of the code is normally executed 90% of the time)
- Implementing *specific algorithms* (e.g. Fast Fourier Transform)
- Realizing an *application environment* (e.g. terminal control for improving stack handling capabilities, dynamic allocation of communication buffers, speeding up interrupt processing etc.)

4.6 Control Devices and Special-Purpose Applications

In modern computer systems, *control* devices are frequently implemented by microprogramming (e.g. data channels, disc control unit, graphical display control etc.).

Microprogrammed versions can greatly improve the features of *processing* systems (e.g. image processing).

Glossary of basic terms

Architecture
The set of computer's features which are visible to the programmer.

Field (control field)
A collection of bits in the microinstruction that controls a primitive machine activity.

Emulation
The use of microprogramming techniques for the interpretive execution of one machine by another.

Encoded control
A method of microword organization in which the value of the fields must be decoded to generate control signals.

Firmware

A term used to describe the microprogramming level between hardware and software in the implementation of computer systems (usually by code in ROM).

Horizontal architecture

It is not a very precise term to categorize machines whose microword capability has some of the following attributes:

- capability of specifying multiple simultaneous operations
- not highly encoded
- relatively wide
- specifies the address of its successor

Host machine

A microprogrammable computer upon which an emulator for a target machine is implemented.

Macroinstruction (Machine instruction)

A machine language or macro level instruction.

Microcode

One or more microinstructions of writing a microprogram.

Microcycle

The smallest unit of time available for the execution of a single microinstruction.

Microinstruction

One instruction of a microprogram.

Microoperation

Specification of a primitive machine operation or an operation specified by one (or more) field(s) of a microinstruction.

Microprogram

A program composed of microinstructions.

Microprogrammable

Pertaining to the capability to control the actions of the micro-level machine via microprogramming.

Microword

A word in the control storage.

Organization

A level below architecture, organization is concerned with, how the facilities available to the programmer are provided.

Realization (Implementation)

In the hierarchy of architecture, organization and structure, realization describes the lowest level - the chips and connections which implement a machine structure.

Structure

All the functional building block and their relationship (connections, links, intercommunications).

Target machine

The computer whose architecture is implemented by an emulator running on a host machine.

Vertical architecture

A terms used to describe machines whose microwords have some of the following attributes:

- specify a single operation
- Relatively narrow
- Highly encoded
- Microinstruction-counter sequencing

WCS (Writable Control Store)

Control Store implemented with RAM so that the user can dynamically alter its contents.

REFERENCES

[1] M.V. Wilkes: The best way to design an automatic calculating machine. Manchester Univ. Computer Inaugural Conf., Manchester, July 1951. pp. 16-18 (Reprinted by the Microprocessing and Microprogramming. Vol. 8. 1982. pp. 142-144)

[2] C.G. Bell et al.: Computer Engineering. Digital Press, 1980.

[3] A. Opler: Forth-generation software, Datamation, January 1967. pp. 22-24 (Reprint by the Microprocessing and Microprogramming, Vol. 8. 1981. pp. 146-148)

[4] L. Richter: Forschung and Praxis bei vertikaler Migration (13. Wasco-Tagung in Budapest, 1982.)

[5] G. Chroust and J.R. Mühlbacher (eds.): Firmware, microprogramming and restructurable hardware, North Holland Publ.Co. 1980.

[6] S. Davidson, B.D. Shriver: An overview of firmware engineering, Computer, Vol. 11. 1978. pp. 22-33

[7] B.W. Boehm: Software engineering, IEEE Trans. on Computers, Vol. C-25, 1976. pp. 1226-1242

[8] M. Tokoro et al.: Optimization of microprograms, IEEE Trans. on Computers, Vol. C-30, 1981. pp. 491-504

[9] D. Landskov et al.: Local microcode compaction techniques, Computing Surveys, Vol. 12. 1980. pp. 261-294

[10] J.A. Fisher: Trace scheduling: A technique for global microcode compaction- IEEE Trans. on Computers, Vol. C-30. 1981. pp. 478-490

[11] S. Dasgupta: Some aspects of high-level microprogramming, Computing Surveys, Vol. 12. 1980. pp. 296-326

[12] T. Gilb: Software metrics. Wintrop Publ. Inc. Cambridge, Mass., 1977.

[13] F. Vajda et al.: EMU: A bit-sliced microprocessor-based emulator, in Microprocessor System: Software, firmware and hardware (eds.: Sami, Thompson and Mezzalire) North Holland Publ. Co. 1980. pp. 53-58

[14] C. Vickery: Software aids for microprogram development, 7th Annual Workshop on Microprogramming, pp. 208-211

[15] T.G. Ranscher and P.M. Adams: Microprogramming: A tutorial and survey of recent developments, IEEE Trans. on Computers, Vol. C-29. 1980. pp. 2-20

[16] M.J. Flynn: Directions and issues in architecture and language, Computer, October, 1980. pp. 5-22

[17] Y. Chu and M. Abrams: Programming languages and direct--execution computer architecture, Computer, July 1981. pp. 22-32

[18] G. Frieder: The Fortran project, SIGmicro Newsletter, Vol. 8. 1977. pp. 47-50

[19] H. Cordero and J.B. Chambers: Second group of IBM 4341 machines outdoes the first, Electronics, April 8. 1981. pp. 149-152

[20] W. Grätsch and H. Kästner: Firmware monitoring - History and perspective. Microprocessing and Microprogramming, Vol. 8. 1981. pp. 237-246

Kiadja a Központi Fizikai Kutató Intézet
Felelős kiadó: Szalay Miklós
Szakmai lektor: Ambrózy György
Nyelvi lektor: Csákány Antal
Gépelte: Beron Péterné
Példányszám: 315 Törzsszám: 82-169
Készült a KFKI sokszorositó üzemében
Felelős vezető: Nagy Károly
Budapest, 1982. március hó