Initialization of Microprogrammed Machines

Helmut Painke
IBM Laboratories
Boeblingen, Germany

1. Introduction

Technoligical trends are key considerations in the design of computers. In the past years packaging density of circuits and memories has increased dramatically. Figure 1 reveals that circuit density improved by a factor of ten in the past five years, and there is ample evidence that this trend will continue in the next few years to come. As for memories, we observed a stagnation in the sixties. With the advent of monolithic memory the progress is even more evident than in the circuits area. A density improvement of a factor of ten in the past two years is accompanied by a whole bulk of indications that packaging density will progress at an even faster pace in the coming years. Consequently, cost is rapidly decreasing, especially with memories.

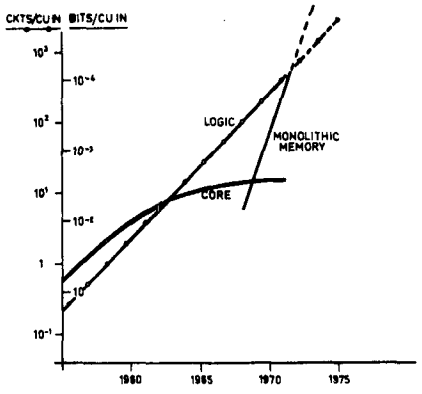

FIG.1: PACKAGING DENSITY TRENDS

There is another advantage memory has over circuits. The economies are obviously in favor of mass production of chips with one single partnumber compared to many chips with a variety of partnumbers. In the circuit area, however, partnumber proliferation seems inevitable. The tendency to raise the level of intelligence in peripheral areas of the system results in a close interplay between the mechanical properties and the electrical controls of those devices yielding decidedly individual logic structures. In the main processor area we have to deal with new architectures and expansions of old ones. The high investment in programs, however, enforces support of the old architectures as well. Consequently, the processor has to be adaptive to several architectures which suggests overlays.

All this implies that there is a strong motivation for the designer to define some universal self contained functions which can be implemented in highly integrated circuits, and then do the adaption to the various architectures and structures via microprogram in read/write memories which permit overlays.

2. Structures and Tradeoffs

Monolithic memories are volatile. Consequently, the microprograms have to be loaded from an external source so that the system gets its identity. In other words, the system must be initialized. Methods and techniques of this initialization are dependent on the formats and conventions used in the microprograms. So let us have a look at those first.

In designing a microprogrammed system the designer is faced with a wide spectrum of possible implementations. There is the Turing Machine with its few and simple instructions at one end and the hardwired machine at the extreme other end. His choice for the proper implementation point is mainly guided by two criteria:

1. microprogram space requirements versus circuit cost and
2. performance requirements.

Microprogram space is obviously dependent on the power and complexity of the microinstruction set, where the reciprocal relative power may be defined as the number of steps required

to perform a given function. If a microinstruction set is half as powerful as a given one twice as many instructions are needed to perform the same function. The word length of the microinstructions, however, can only be reduced by one bit if binary coding is applied. This means that with decreasing power of the microinstruction set the bit requirements increase at nearly the same rate (Figure 2).

As the microinstruction set gets simpler and simpler there are programming techniques like the use of subroutines that limit the memory requirements at the expense of degraded performance.

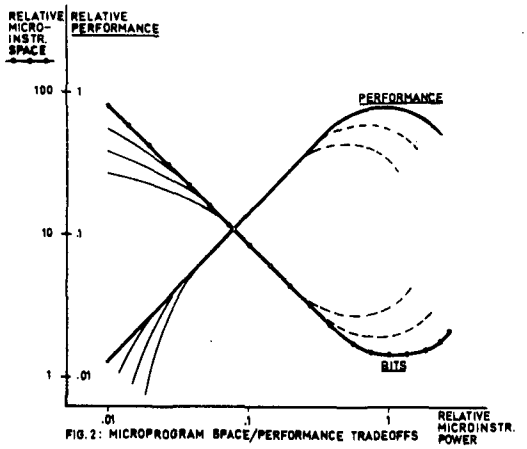

FIG. 2: MICROPROGRAM SPACE/PERFORMANCE TRADEOFFS

If the microinstruction set becomes very powerful compared to the function to be implemented - say more powerful than the macroinstruction set to be emulated - the bit requirements go up again.

This effect can be illustrated by trying to add two digits with a slide rule. The slide rule has the very powerful instructions multiply, divide, logarithmic and exponential functions, and by repetetive use of these functions a simple add may be performed. It is obvious that this takes more steps (= bit space) than if a simple add function would be available on the slide rule.

Each possible instruction set requires a certain number of circuits to decode and execute those instructions. This circuit count is a direct function of the power of the microinstruction set. The more complex a microinstruction set gets, the more specialized are its instructions which results in a larger number of different instructions for that set. Also, the more complex instructions need more circuits for decoding and execution. This means that with decreasing power the cost for the circuitry decreases while the cost for the microinstruction store increases. The crossoverpoint which signals the cost optimum varies with the total problem size and is technology dependent. The cost trends mentioned before, however, suggest that this optimum is constantly shifting to the far left of the curve in Figure 2.

Besides cost, performance is of prime importance. It is quite obvious that performance is practically directly proportional to instruction power. Half the instruction power results in twice the number of steps for a given function and, consequently, in nearly half the performance. Only a very small fraction of this speed loss can be recovered by the slightly faster cycle time associated with a simpler instruction set. Using subroutines in order to save bit space results in a further performance degradation. The same holds true for an overpowered instruction set.

The two curves in Figure 2 fan out as they approach the optimum (value "1"). The solid line marks the case where the microinstruction set not only has roughly the same power as the macroinstruction set to be implemented but also the same structure. There are numerous cases where instruction sets have identical power, but completely different structures. One example is a system with fixed word length versus one with variable word length. Optimizing the microinstruction set for one case results in a high inefficiency for the other case (dotted lines). Another example is instruction processing versus controlling of I/O functions. In a real system we have to deal with both cases. More than half the intelligence of a system is devoted to controlling I/O, and in the processor we have to support different architectures. There are two ways to overcome this inefficiency. One is to select an implementation point far enough left on the curve where the solid and dotted lines merge. This solution of course results in a sizable performance degradation which might not be tolerable in all cases. The other way is to implement the several areas in different microinstruction sets and optimize each one separately. This, of course, means that several processors are required to support the different microinstruction sets.

Summarizing, we can state that the general case for initialization is characterized by several addressees each having a separate format. Also, secondary storage in the different units like control registers usually introduce a new format if they are to be initialized. This

indicates that the initialization process can be rather complicated and normally requires elaborate controls. The problems associated with this fact will be addressed and solutions will be presented in paragraph 4.

3. Load Devices and Media

In general, any load device that is used for normal program load can also be used for microprogram load. There are, however, some requirements and conditions that are unique to microprogram loading and thus may affect the choice of the device or medium. These are:

- Microprograms are part of the systems hardware. Without them the system does not have its full identity. Consequently, the full cost of the loading device is part of the basic hardware cost unless it can be shared with other system functions.

- The microprogram implementation should be transparent to the system user. He should not be involved in initializing the system.

- The total size of the microprogram package is usually smaller than the macroprogram support for that system, typically by one order of magnitude.

- Microprograms and macroprograms are normally originated by different sources. In nearly all cases the microprograms are generated by the group responsible for the systems design, whereas the macroprograms are created in a different organization, often by separate software houses.

- Microprograms are usually loaded once a day. Even if overlays are used, e.g. for diagnostic reasons, access time and data rates are usually of secondary importance.

These peculiarities have some consequences with regard to the adequacy of the various devices for the initialization job. Punched cards offer the advantage of low additional systems cost since a card reader is usually available with each system. On the other hand user transparency is not achieved - the user has to shovel cards in order to help the system to get its identity. The user can even create hardware errors by mixing up the card decks. The comparatively low reliability is a further disadvantage.Similar considerations are valid for punched tapes.

Magnetic disks are the primary macroprogram stores of todays systems. Their high recording density, their excellent reliability supported by automated retry features, and their low bit cost can advantageously be used for microprogram loading as well. Usage is, however, restricted to disk oriented systems since using such an expensive device for initialization purposes only would be prohibitive. The transparency can easily be achieved by having the microprograms on the system resident pack which contains the basic support macroprograms. This, on the other hand creates problems in controlling the change level of programs from different sources on one pack.

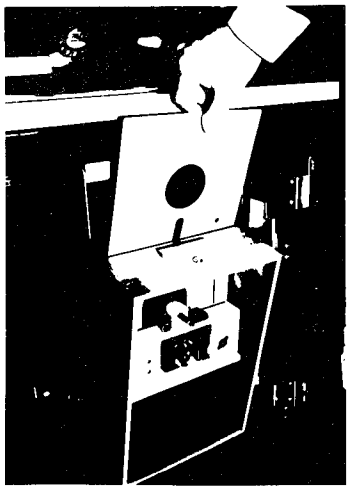

Fig. 3: Flexible Disk as Integrated Loading Device

All these problems can be avoided if a separate integrated loading device is used. Figure 3 shows a magnetic disk reader using flexible diskettes as an example. With a capacity of 80K bytes and a data rate of 4K bytes it meets all requirements of a wide spectrum of microprogrammed systems. Tape cassette readers could do the same job. They usually provide slightly higher capacity at slightly lower data rates. Those integrated devices meet the transparency requirement of fairly low device and media cost. Furthermore, they allow the desired decoupling between systems programs and microprograms. They seem to mark the implementation trend of the future.

4. Procedures and Problems

There are three phases in the process of initialization. Firstly, the system hardware has to be set up to a predefined initial status. This is usually done by a hardware reset line which resets all error latches, starts or sets up the internal system clock and initializes

certain control registers. In the next step the microprograms are loaded into the various units, and in the third step the system is set to the "ready" state by either setting up a start address for the microprograms or by starting them into the main waiting loop.

While the first and the third step are fairly trivial techniques the loading of the microprograms is easier said than done. Loading from a disk pack e.g. involves seeking and verifying of the right track, format recognition, reading and validity checking, reformatting, transferring of the data to various units, retry on errors etc. It is evident that if all those functions have to be available in a system that has not been initialized yet a substantial part of the systems control and practically the whole disk controller has to be implemented in hardware rather than in microprogram.

The simpler the device, the smaller is the hardware part required for this bootstrapping. An integrated loading device like the one shown in Figure 3 with its simple construction and its highly specialized functions helps minimizing the amount of hardware required for controlling the device. Even in this case, however, the initialization process with its various formats and addressees is complex enough. Imagine a check coming up during initialization. Firstly, it is a linear rather than a cyclic process, i.e. measuring is tough unless a storage oscilloscope is used. Secondly, many of the built in functions used for error recognition and analysis are implemented in microprogram and, consequently, not yet available during the load process. All this implies that the logic controlling the initialization must be highly intelligent and at the same time extremely reliable.

A clean solution to these problems is a small microprogrammed controller driving an integrated loading device. Since speed performance is of no importance, it can be implemented with minimal hardware and thus provide high reliability. At the same time, any desired level of intelligence can be achieved through microprogramming it. A read only storage can be used to provide the minimal intelligence required for bootstrapping. Once the complete microprogram for this controller is loaded its full intelligence is available for the complex task of total systems initialization. Such a hierarchical structure as shown in Figure 4 is at the same time very helpful for diagnostic and maintenance as will be outlined in paragraph 5.

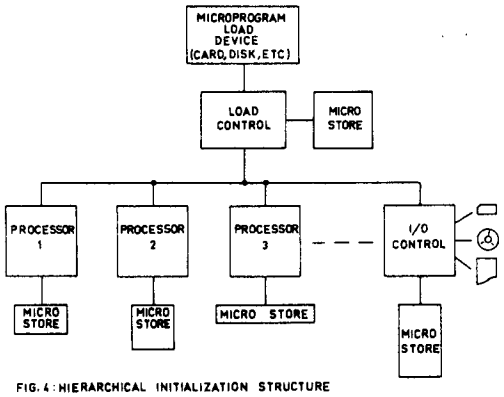

FIG. 4: HIERARCHICAL INITIALIZATION STRUCTURE

A rather complex problem is the distribution and implementation of microprogram changes, e.g. in order to add features to an installed system. Besides the cost of the distribution media organizational implications are to be considered. Let us use the case of the disk pack resident microprogram as an example. To ship a new disk pack would be prohibitive not only for cost reasons. There is also the engineering change control problem of two data sets controlled by different authorities on the same medium. Consequently, changes have to be shipped to the system with a different medium (e.g. cards) using the system itself to reconfigure (reassemble) the microprogram and then to rewrite it onto the disk. An integrated loading device like the one shown in Figure 3 solves these organizational problems since the cost of the flexible disk is low enough to allow replacement with a new one with each change.

5. Positive Sideeffects

As tough as all those problems may seem, there would not be a microprogrammed machine unless the advantages outweigh the problems. Some of the advantages were already mentioned in the previous paragraphs, like cost optimization, emulation of different architectures, ease of changes etc. But over and above these the mere presence of the initialization paths and devices opens up a whole new dimension of advantages and possibilities mainly concentrated in the areas of diagnostics and maintenance. Some examples may highlight the situation.

- The same storage can hold different information at different times. Rather than loading the actual systems identity at power-on time a test routine can be loaded that checks

all the hardware, issues error messages in case of hardware failures and even diagnoses the location of the failure. In case of an error-free run the normal microprogram is then overlaid. Thus the user has an increased assurance that the system is error-free when turned over to him. In case of an error during normal operation this test routine can automatically replace the normal microprogram through a new initialization and thus perform the necessary analysis.

- Most input/output devices have cams that generate timing pulses. These cams have to be checked and adjusted periodically. Rather than using an oscilloscope the microprogrammed controller can be used to start the device and sample the timing pulses using a microprogrammed raster. The result can be displayed on any given output device like a printer or a cathode ray tube. Figure 5 shows an example. Besides the actual time measured at the various cams this printout also lists the permissible tolerances and a lot of supplementary information. Such a printout is much more instructive than any oscilloscope picture.

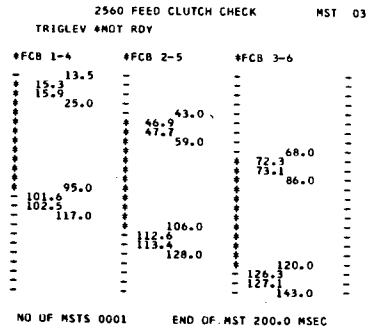

Fig. 5: "IMPULS CHECK ROUTINE", an example of microprogrammed maintenance aids

- In case a hierarchical concept like the one shown in Figure 4 is used the initialization controller can be used as a diagnostic tool for the rest of the system. It can be loaded with programs that allow checking of any lower member of the hierarchy. If the loading device has write capability it can even be used to store error logs for a later analysis.

6. Summary

The trend to high integration suggests that we should implement the diverging architectures and structures in microprograms using a limited set of highly integrated standardized functions as basic building blocks for the hardware. The vast spectrum of functions and architectures to be implemented forces us to either compromise performance or to optimize the microinstruction sets in the various areas resulting in several formats in a system. Many of the problems associated with the complexity of the initialization process can best be solved by using a special microprogrammed controller. A separate integral loading device eases the solution of organizational problems in the distribution and implementation of engineering changes.

Last, not least, such a systems concept offers a multitude of benefits most of which are centered around reliability, availability and serviceability.