JOHANNES KEPLER
UNIVERSITÄT LINZ
TECHNISCH-NATURWISSENSCHAFTLICHE FAKULTÄT
INSTITUT FÜR INFORMATIK

A Microprogrammed Page Fault Monitor

G. Chroust*, A. Kreuzer**,
K. Stadler**

SYSPRO 12/80 March 80

INFORMATIK-BERICHTE

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

A Microprogrammed Page Fault Monitor

G. Chroust*, A. Kreuzer**,
K. Stadler**

SYSPRO 12/80 March 80

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

This paper will be publsished elsewhere. .
As a courtesy to the publisher its distribution is strictly limited.

*IBM Laboratory Vienna
Obere Donaustraße 95
A-1020 Wien

**Institut für Informatik
Universität Linz
A-4040 Linz/Auhof

A Microprogrammed Page Fault Monitor

G. Chroust*, A. Kreuzer**, K. Stadler**

*IBM Laboratory Vienna, **Kepler University Linz

Abstract:
In order to investigate the influence of different programming techniques on the paging behaviour of the resulting object program, a page fault monitor was implemented in firmware.
The paper discusses the general design philosophy and properties of firmware monitors and describes the specific implementation. It also shows some results of the use of the page fault monitor.

1.0 INTRODUCTION

1.1 MOTIVES

At the Kepler University Linz a project which investigated the influence of different programming methodologies on the paging behaviour of the resulting object program (/Muehlbacher-79/) required to measure the paging rate of a computer system under various conditions (Similar investigations on paging behaviour are reported in /Ferrari-76/, /Freeman-75/ and /Hatfield-71/). Since at the same time investigations on microprogramming and on firmware monitoring were being performed (/Chroust-78/), it was decided to implement a page fault monitor in firmware.

The experiment was intended to

* estimate the difficulties of microprogrammed changes to an existing complex emulator,
* demonstrate the feasibility of implementing a firmware monitor,
* understand the properties of firmware monitors,
* provide a useful tool for the investigation of paging behaviour.

This paper discusses the general design philosophy of firmware monitors, describes a specific implementation of a page fault monitor and shows some results of its application.

1.2 VIRTUAL STORAGE CONCEPT

The main purpose of a virtual storage system (Fig. 1) is to link two or more memory systems of drastically different speed and capacity (and thus cost) in such a way that

* the apparent capacity for the user approaches that of the larger system,
* the actual speed approaches that of the fastest memory unit, and
* the resulting costs for a given virtual capacity and speed are minimized.

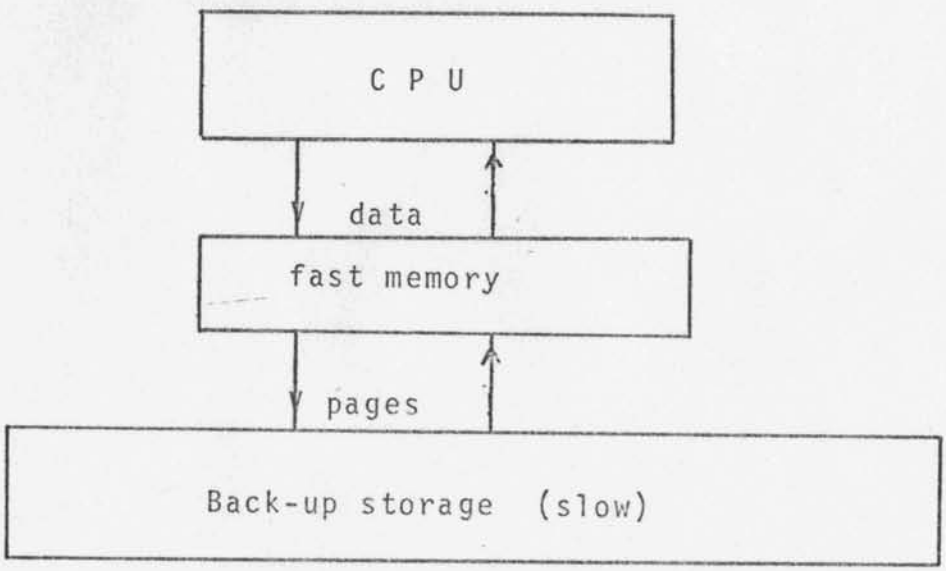

Fig. 1: Virtual Storage System

The logical foundation for virtual storage systems are the observed facts (cf. /Buzen-73/, /Denning-70/, /Hellerman-75/, /Tanenbaum-76/) that

1. most programs tend to access only a comparatively small region of storage during a given time period ('locality'),
2. during execution these regions tend to change comparatively slowly with respect to size and position in the complete virtual storage space ('dynamic locality').

In a paging system the address space of the virtual memory available to a program is divided into pages. Only a few of the 'virtual pages' used by the program are physically in

fast memory at any one time (Fig. 2). The remaining pages are stored in the back-up memory. The system must translate virtual addresses into physical ones and must make sure that virtual memory locations needed by the program are in memory when needed.

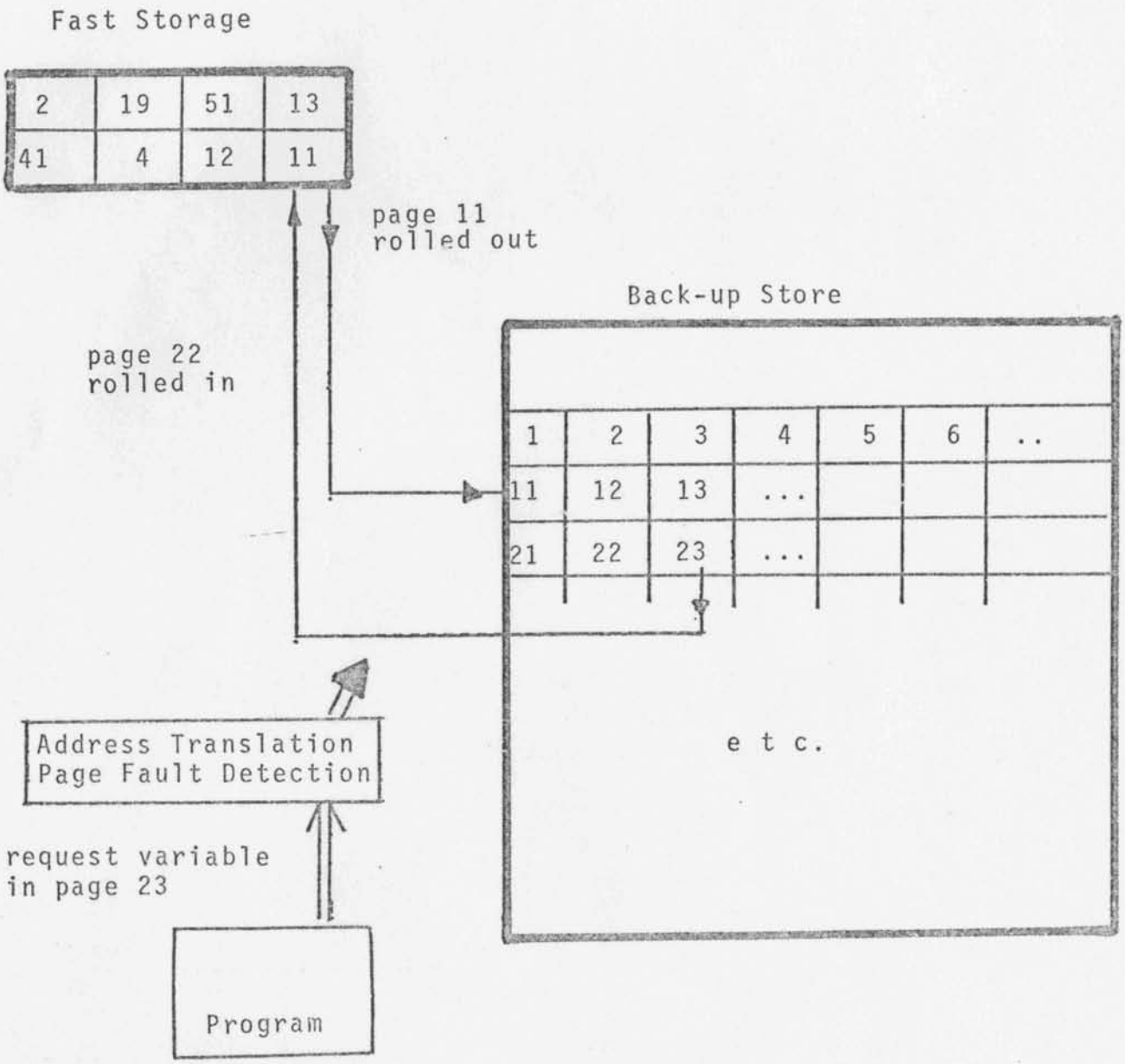

Fig. 2: Scheme of a paging system

A page fault occurs if a required page is not physically present in the fast memory. In this case the page has to be loaded from back-up storage. This usually requires to roll out a page from the fast memory to the back-up memory. Loading a new page is by far the most time consuming single

operation in a virtual memory system (1000 instructions and more is not unusual). The amount of page changes, the page rate is thus of major influence on the execution time of any one program.

There exist indications that certain classes of programs can be successively adapted with respect to their programming style in order to better fulfill the requirements of locality and dynamic locality. Applying the knowledge about the correlation between programming techniques on one side and locality and dynamic locality on the other side would allow to achieve considerable run time gains by applying appropriate programming techniques to source programs. Therefore a page fault monitor was needed.

1.3 MICROPROGRAMMING, FIRMWARE AND EMULATION

We will understand microprogramming (/Husson-70/) as a tool to define and build machine architectures on top of a given hardware (/Berndt-77/). It is this usage for which Opler has coined the term firmware (/Opler-67/). The interface to the user is called machine architecture and is built via firmware. The notion of computer architecture was introduced by Amdahl, Blaauw and Brooks (/Amdahl-64/) to "describe the attributes of a system as seen by the programmer ...". Firmware is the only reasonable way to implement a computer family with a common architecture on a set of drastically different hardware processors (Fig. 3). The architecture - the actual personality of the computer - is defined by specific microprograms residing in the control store.

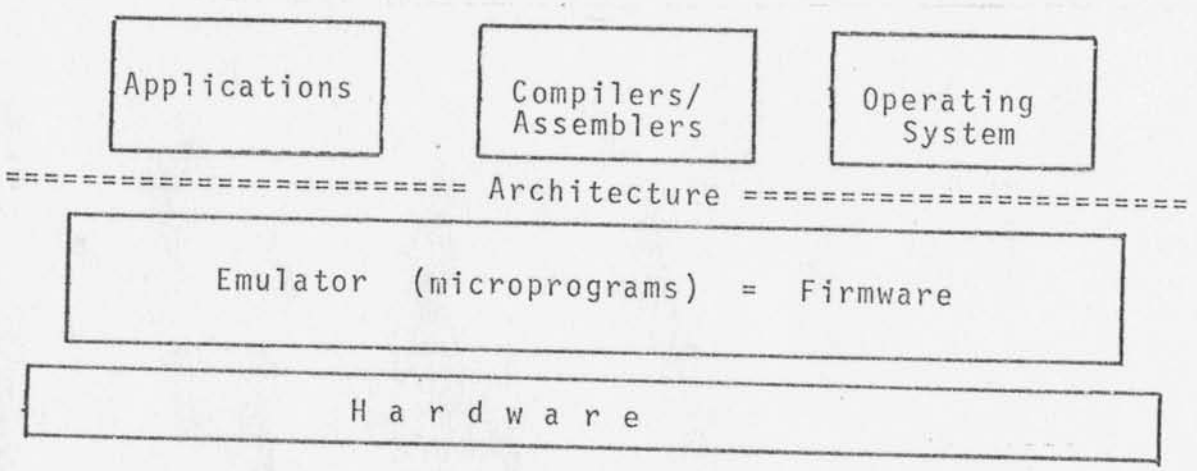

Fig. 3: Layered System with firmware level

2.0 FIRMWARE MONITORING

2.1 HARDWARE AND SOFTWARE MONITORS

Classically two techniques for monitoring are available: hardware and software monitors (/Ferrari-78/, /Klar-71/, /Svobodova-76/).

Hardware monitoring allows a practically disturbance-free measurement of system activities. The accumulated data are essentially counts of electronic pulses on certain lines. In most cases, however, one is not interested in pulses per second over a certain line but rather in values of system parameters like supervisor time, problem time, length of system queues etc. It is very difficult to derive such operating system characteristics from pulse counts.

Software monitors allow measurements at the system architectural level, and thus make the identification of the required data easy. However, they usually distort the time behaviour of the original process, since the measuring activities themselves are implemented on the same level as the system to be measured. Monitoring competes with other software components for system resources (main storage, processor, files, busses etc.). This may even change the behaviour of the system under observation: The data file for the monitor data, for example, will probably be in a separate page and thus will provoke a different paging behaviour.

2.2 FIRMWARE MONITORS

It seems (/Armbruster-79/, /Barnes-74/, /de Blasi-77/) that monitoring via firmware is very promising because there is founded hope that firmware monitoring allows a combination of the advantages of hardware and software monitors without entailing their difficulties (/Chroust-78/).

The advantages of firmware monitoring are:

* The data generation algorithms in the monitored processor are written in microcode and are thus an order of magnitude faster than the equivalent software routines; thus the time distortion is much smaller and can usually be ignored.

* Operating system characteristics can still be measured directly at the microprogram level. In many cases the investigated system functions are partially implemented in microcode and thus directly accessible (e.g. paging algorithms in IBM/370-115, /IBM-74/).

A system is called a 'Firmware Monitor System' if the core measuring process are handled by firmware (i.e on the level of microprogramming). The total system therefore need not consist primarily of firmware. In some cases the actual percentage of microcode might even be rather small, especially when the time distortion of the process is not critical and the microprograms only provide the 'raw data' for monitoring, whereas the further processing and accumulation is done by software.

2.3 DESIGN CONSIDERATIONS FOR A PAGE FAULT MONITOR

2.3.1 Requirements

For the project in question the following objectives for the microprogrammed Page Fault Monitor (PFM) are to be taken into account:

1. The object machine is an IBM /370-115, with an OS/VS1 Operating System.
2. It must be possible to start and stop the PFM and to read the resulting data manually and under program control (Assembler and PL/I).
3. The counters should be settable/resettable independantly of the starting/stopping of the monitor.
4. The monitor should disturb the process to be monitored as little as possible.
5. It must be possible to count separately those page faults which occurred in the problem state and those which occured in supervisor state.
6. The PFM should be easy to use.
7. No change to the operating system, a compiler or assembler should be required (In this way the montoring process is transparent on the machine architecture level and no additional expertise on software products is necessary).

2.4.1 Hardware Properties of a firmware monitor

A firmware monitor behaves, as seen from the user, like a hardware monitor (/Rueberg-78/), without needing any special hardware resources. The main reasons are:

* A firmware monitor allows essentially distortion-free measurements. Since the measurement activities are performed in microcode, the distortion is usually negligeable.

* The use of the monitor is transparent on all levels which are accessible to programmers, even on the level of machine instructions. This means that the behaviour of the machine as specified in the architecture is not changed, not even if interrrupts etc. occur.

* Microprogramming allows to utilize hardware elements which are not used and not accessible otherwise.

2.4.2 Software Properties of a firmware monitor

Many properties of firmware monitors are common with software monitors:

* Most machine and system control blocks and entitities (queues, page tables etc.) are equally well accessible from firmware and software.

* The flexibility is almost the same as for software monitors.

2.4.3 Special Considerations for Firmware Monitors

The implementation of a firmware monitor demonstrates some additional considerations in comparison with software or hardware monitors.

1. Communication with the monitor
   Means must be found to start/stop/read/reset the monitor. Ease of use, simplicity and transparency have to be aimed at as far as possible.

2. Choice of 'Monitor hooks'
   One has to identify those points in the emulator where the desired information is available. This is usually the most challenging part of the study, since it is difficult to identify such points. Secondly one must make sure that one does not measure too much, i.e. that the specific point is not executed under some other conditions, too.

3. Compatibility with the standard machine environment
   Every change of the emulator of a computer is a change in the architecture of the machine. This means that a change usually is always in effect. One must make sure that this change does not have adverse effects for other programs and/or users. It necessitates sometimes the use of special 'flags' to indicate if the changed emulator code should be effective or not. Furthermore - due to the complexity of the emulator - it is absolute necessary to avoid destroying registers and other temporaries of the emulators (often by saving/restoring them).

4. Estimating the sizes of counters.
   Usually rather large numbers are generated. Due to the small size of the hardware registers this often involves overflow handling at the microprogram level.

5. It is more difficult to read and print the values accumulated by the firmware monitor as compared to a software monitor. Usually, however, it is possible to hand the firmware-accumulated data to existing software routines which then perform the input/output function.

3.0 IMPLEMENTATION

This section gives an overview over some of the details of the chosen implementation. For more details see /Muehlbacher-79/.

3.1 PAGING IN OS ON IBM SYSTEM 370/115

To understand the implementation of the PFM a short introduction to the emulation of System/370 and the operating system OS/VS1 as used on the IBM /370-Mod. 115

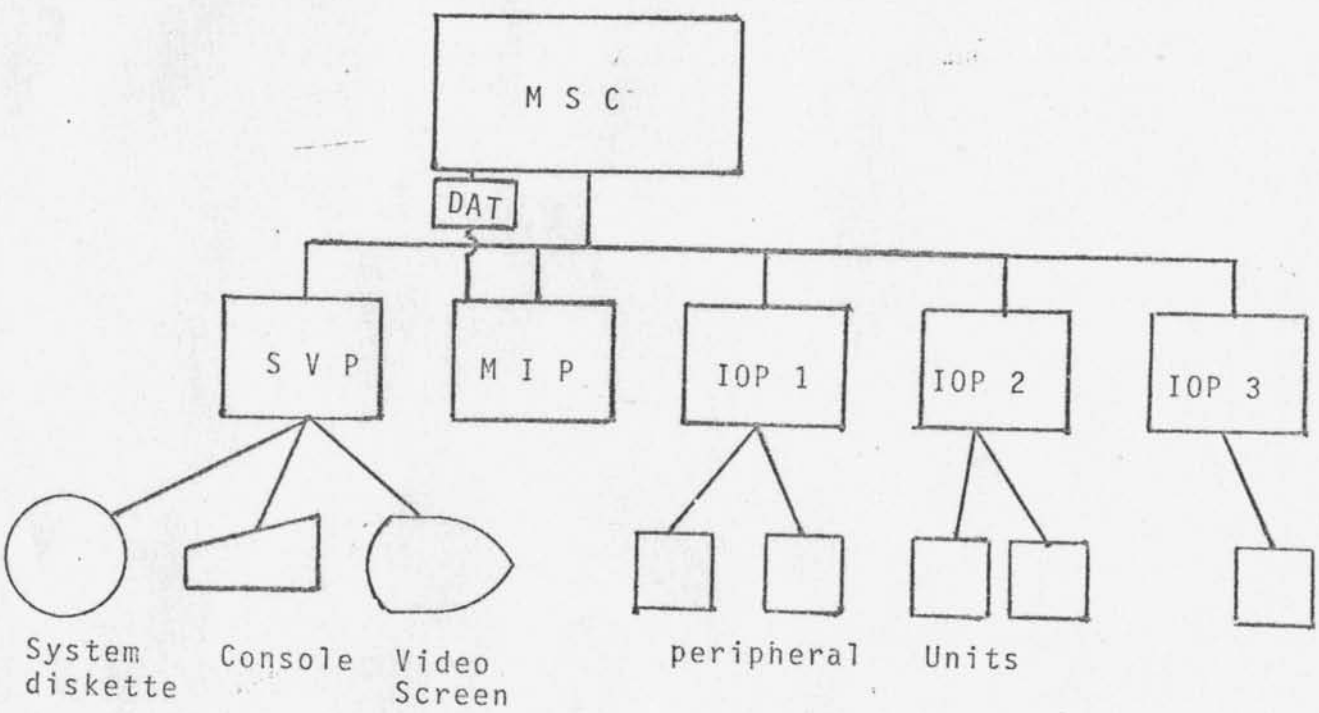

Fig. 4: Diagram of the IBM/370-115

hardware is necessary.

The IBM /370-115 (/von Krogh-74/, /IBM-74/) is internally organized as a multiprocessor consisting of several individual processors (Fig. 4). The Machine Instruction Processor (MIP) handles all machine instructions. I/O instructions are only superficially examined by the MIP in

order to determine at which Input-Output Processor (IOP) the corresponding peripheral unit is attached. Then the instruction is transferred to this IOP which in turn executes the I/O operation asynchronously and interrupts the MIP upon termination (/Assmuth-74/). The Service Processor (SVP) together with its independent bus is responsible for communication with the operator and for general reliability and supervisory tasks, not directly related to instruction procession.

Storage access is controlled by an independent and asynchronous Main Storage Controller (MSC). It contains, permanently assigned to each other processor, several address registers. In order to access Main Storage (MS) the respective processor firstly loads the corresponding address into one of 'its' address register (there are some mechanism to ease consecutive access to contiguous memory locations) and then requests a memory read or write using this address. If the page is in memory the corresponding data are written from/to the data bus. To make paging efficient a special unit ('Dynamic Address Translation' - 'DAT') is inserted between MIP and MSC. It contains an associative memory which allows fast translation between virtual addresses (as issued by the MIP) and real addresses (as accepted by the MSC). In case the page is not found a page fault is recognized. The interpretation of the machine instruction causing the page fault is interrupted (trapped) upon a signal from DAT. A similar trap is generated by the MSC when the value in the address register transgresses a page boundary. However, this not necessarily induces a page fault, since the next page might be already in storage. A trap causes the MIP to transfer control to a different microprogram. The microprogram initiates some software programs which load the appropriate memory page from the external medium and then reinterprets the whole interrupted machine instructions.

All processors except the MSC are microprogrammed and microprogrammable.

3.2 THE PAGE FAULT MONITOR

This section will address the points raised in 2.4.3.

1. To communicate with the monitor three unused Control Registers of the IBM/370-115 (Registers 6, 12 and 13) are used. The use of the control register has the advantage of easy access from microprogram, program and console.

   Control Register 6 is used to start and stop the monitor,
   Control Register 12 counts those page faults which occurred in the problem state,

Control Register 13 counts those page faults which occurred in the supervisor state.

2. In order to find the appropriate point to count page faults the emulator code was investigated. In our specific case it was rather easy: Whenever a page fault occurs the DAT raises a signal which is detected by the MIP. Actually the MIP traps to a special code for all exceptional conditions. A trap can have many different causes, the (original) microprogram tests various flags to determine whether the trap was caused by a page fault and and goes to a special routine to handle page faults. To count the page faults the accumulating code had to be inserted into that path: Actually the monitoring microcode is handled like a 'patch': one of the original instructions is replaced by a jump to some free area in the microstore where the monitor code is stored. The count therefore shows how often this specific path in the emulator has been executed.

   The inserted code is essentially as shown in Fig. 5.

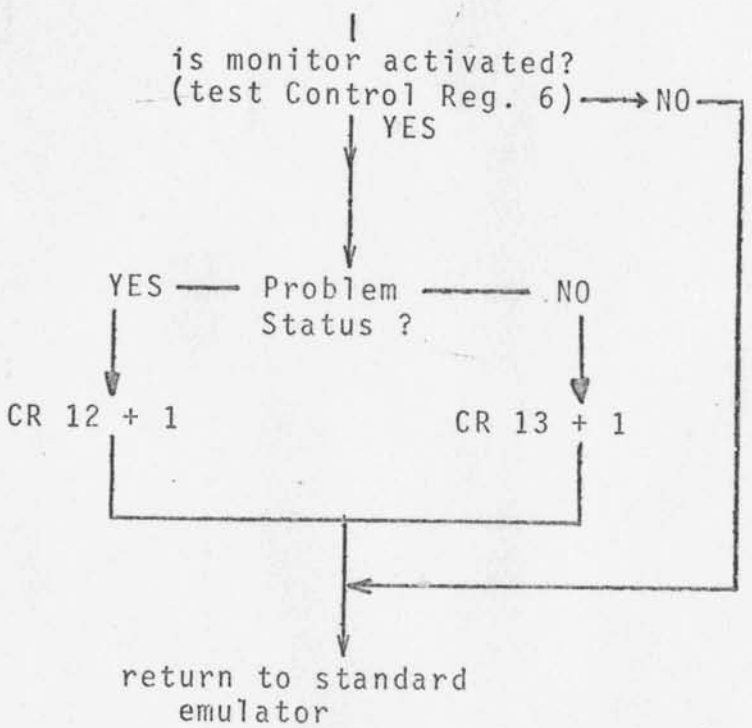

Fig. 5: Scheme of monitor code.

Naturally the insertion of an additional piece of microcode also requires (like any other patch) to save some of the MIP's register, to replace the original microinstruction by a branch and repeat the replaced microinstruction later.

3. Compatibility did not pose a problem here, since the monitor code was an extension rather than a change to the existing firmware. The communication with the outside world was handled via otherwise unused control registers.

4. It was decided that the size of the control registers (2**31-1) = approx. 2.10**9 was sufficiently large to ignore overflow (the fastest machine instructions on the IBM /370-115 are in the order of 6 microseconds, thus allowing for several hours of continuous monitoring, even in the worst case).

5. Reading and printing of the accumulated values was avoided by leaving the data in the control registers. Output is the user's responsibility.

4.0 USE OF THE PAGE FAULT MONITOR

4.1 COMMUNICATION WITH THE PFM

Four functions are necessary for communication

* Resetting the counters:
  Control Registers 12 and 13 are set to X'00000000'

* Starting the monitor
  Control Register 6 is set to X'FFFFFFFF'

* Stopping the Monitor
  Control Register 6 is set to X'00000000'

* Reading the Counters
  Control register 12 contains the count of the page faults which occurred in problem state since the last resetting. Counting is only performed during those time periods where the monitor was switched on.
  Analogous Control Register 13 contains the page faults in supervisor state.

The convention that stopping the monitor is independent of reading the counters and resetting the counters allows to accumulate statistics on several non-contiguous execution paths.

4.2 MANUAL CONTROL OF THE PFM

Manual Control is accomplished by changing Control Registers 6, 12 and 13 via the console as described in /IBM-75/.

4.3 CONTROL OF THE PFM FROM ASSEMBLER PROGRAMS

To control/communicate with the monitor from an assembler program it is necessary to access the control registers using the instructions: LOAD CONTROL (LCTL) and STORE CONTROL (STCTL), (cf. /IBM-76/)

In the standard architecture, however, these two instructions are privileged. The /370 emulator was therefore changed such that these two instructions can be executed in problem state. It should be noted that this measure was sufficient for the intended application of the PFM, since 'well-behaviour' of all users was expected. Otherwise access to all control registers makes the

operating system extremely vulnerable. A safe implementation would restrict the availability in the problem state to Control Registers 6, 12 and 13.

In /370 Assembler Language the following statements are needed:

```
Starting PFM:    LCTR 6,=X'FFFFFFFF'

Stopping PFM:    LCTR 6,=X'00000000'

Resetting the counters:
                 LCTR 12,=X'00000000'
                 LCTR 13,=X'00000000'

Reading the results:
                 STCTR 12,FWP
                 STCTR 13,FWS
```

where FWP and FWS are two fullword locations for the counts of page faults in the problem state and the supervisor state respectively.

4.4 COMMUNICATION FROM HIGH-LEVEL LANGUAGES

In order to be able to communicate with the PFM from higher-level languages (specifically PL/I) 4 assembler subroutines were written:

PFMON: start the monitor
PFMOFF: stop the monitor
PFMZERO: reset the counters to zero
GETPFM (prob, sup) : read the counters
where prob and sub (each a fullword) return the values of the counter for problem and supervisor state respectively.

The code for individual functions is the same as described for assembler programs. These subroutines may be activated by standard procedure calls in PL/I.

5.0 VERIFICATION OF THE MONITOR

A considerable part of the design of a page fault monitor system is the verificiation of its results. It has to be checked whether

* all relevant page faults are counted and
* no unrelated events are also included in the count.

Basically 4 methods can be applied:

1. Tracing the different execution paths of the emulator manually in the microcode (desk method).
2. Tracing dynamically (using some system support) the execution of the monitor. If available the intended point of insertion of the monitor code can be check-pointed in order to verify the soundness of the assumptions.
3. Comparison with other monitoring aids (some manufacturers provide monitoring aids with their operating system, for example IBM's SMF (/IBM-76b/).
4. Measuring using synthetic jobs with known paging rates.

For the PFM all methods were applied.

ad 1) Manuals and microcode were followed and it was verified that

- o each page fault must go through the point of measurement,
- o only page faults have the proper flags set such that flow of control will go through the monitoring code.

ad 2) The service and test aids of the IBM /370-115 allow to stop a given microprogram in any processor at a predefined location (/IBM-74/). One can then inspect the status of registers, memory etc. This was done to verify the information extracted from the manuals, especially the value of the flags indentifying different causes for traps were checked.

ad 3) The IBM supplied SMF program (/IBM-76b/) was executed in parallel with the PFM and the results compared. It turned out that the results by SMF were 10 to 15% higher: The difference stems from the fact that SMF measures according to slightly different criteria as compared with PFM. Some functions which logically belong to the object program but which are executed in supervisory mode are included in the counts of SMF but not in those of PFM (cf.

/Muehlbacher-79/, /IBM-76b/). The data showed a sufficient similarity to have confidence in the PFM.

ad 4) For this purpose a PL/I program was written which used two very large arrays (A1 and A2). Each array is larger than the (known) maximal available space for in-memory pages. At first the data in A1 are accessed in such a way that all available page space will be occupied by pages containing A1. This assures that no page of A2 is in storage. Then the monitor is started and elements of A2 are accessed such a way that the number of page faults can be predicted. At the end the PFM is switched off. The calculated page faults must correspond with the measured page faults.

6.0 APPLICATION : PAGING IN OS ON IBM 370/115

6.1 GENERAL EXPERIENCE

The PFM has been used by various applications at the Kepler University Linz to measure page faults caused by programs and parts of programs in order to yield statements about the locality of algorithms and of data structures. It proved to be a practical measuring tool. Especially the ability to switch PFM on and off via subroutines proved to be advantageous since it allows to gain data about the interesting areas without distortion.

6.2 APPLICATION TO SPARSE MATRICES

A case study for programming in a paged environment was done by implementing Gustavson's Algorithm for the multiplication of two-dimensional sparse matrices (/Muehlbacher-79b/). The same algorithm was implemented using two different representations for its data structures. Measurements with the PFM showed the superiority of one method. The improvements of locality could be measured without distortion by the PFM.

6.3 APPLICATION TO COMPILER ORGANIZATION

The PFM was used to measure the page faults caused by the individual modules of a MODULA compiler. Based on these measurements the structure of the compiler was rearranged in such a way that the run time was reduced (/Pomberger-79/).

6.4 APPLICATION TO STORAGE MAPPING OF NONLINEAR DATA

One part of a project (/Losbichler-75/, /Muehlbacher-79/) is involved in the mapping of binary trees (search trees) into the address space such that e.g. for searching, a minimum number of page faults is achieved. Again the PFM provides a valuable tool.

7.0 SUMMARY

The implementation of a microprogrammed page fault monitor at the Kepler University Linz provided at the same time an experiment in firmware monitoring and a useful tool for the investigation of the influence of programming techniques on the paging behaviour of the object program.

It showed

* that firmware monitoring can be implemented with reasonable effort: In our case, finding the proper 'monitor hooks' was fairly easy and the implementation of the monitoring program was rather straighforward, utilizing our general expertise in microprogramming the IBM/370-115.

* that a page fault monitor provides a highly useful tool, as indicated by the applications in section 6,and

* that the claims about the advantages of firmware monitors are justified (low disturbance, transparency to the user and flexibility).

8.0 REFERENCES

/Amdahl-64/ Amdahl G.M., Blaauw G.A., Brooks F.P. Jr.: Architecture of the IBM System/360.- IBM J. of Research & Dev., vol. 8 (1964), No. 2, pp. 87-97;

/Armbruster-79/ Armbruster C.E.: A Microcoded Tool to Sample the Software Instruction Address.- SIGMICRO Newsletter vol. 10(1979), no. 4, pp.68-72 and ACM (ed.): Proc. MICRO-12.

/Assmuth-76/ Assmuth R., Irro F., Reich L., Schaal H.: Input/Output Control of IBM System/370 Model 125 through dedicated Input/Output Processors.- EUROMICRO Newsletter vol. 2 (1976) No.3, pp. 41-46.

/Barnes-74/ Barnes D.H., Wear L. L.: Instruction Tracing via Microprogramming.- MICRO7, 7th Annual Workshop on Microprogramming, ACM 1974, pp. 25-27.

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

/Buzen-73/ Buzen J.P., Gagliardi U.O.: The Evolution of Virtual Machine Architecture.- Proc. NCC 1973, AFIPS Press 1973, pp. 291-299.

/Chroust-78/ Chroust G., Kreuzer A., Labek F.: Verzerrungsarmes Firmware Monitoring mit unabhaengigen Prozessoren.- 2. Tagung: Berichte aus den Informatik Instituten, Sept. 1978, OCG/OeGI 1978, pp. 29-31.

/deBlasi-77/ de Blasi N., degli Antoni G.: Profile Finder - A Firmware Instrument for Program Measurements.- EUROMICRO Newsletter vol. 3 (1977), No. 1, pp. 27-33.

/Denning-70/ Denning P.J.: Virtual Memory.- Computing Surveys, vol.2 (1970) No. 3, pp. 153-189.

/Ferrari-76/ Ferrari D.: The Improvement of Program Behaviour.- Computer vol. 9 (1976), no. 11, pp. 39-47.

/Ferrari-78/ Ferrari D.: Computer Systems Performance Evaluation.- Prentice Hall, Englewood Cliffs 1978. (especially Ch. 2: Measurement Techniques, pp. 26-100).

/Freeman-75/ Freeman P.: Software Systems Principles.- SRA Chicago 1975, (especially Chapter 6.3.2: Measurement of Program Characteristics, p. 258 ff).

/Hatfield-71/ Hatfield D.J., Gerald J.: Program Restructuring for Virtual Memory.- IBM Systems J. vol. 10 (1971), No. 3, pp. 168-192.

/Hellerman-75/ Hellerman H., Conroy T.F.: Computer System Performance, Chapter 10: Virtual Storage Principles.- MacGraw Hill New York 1975, pp. 272-308.

/Husson-70/ Husson S.S.: Microprogramming - Principles and Practices.- Prentice Hall, Englewood Cl., 1970.

/IBM-74/ IBM Corp.: 3115 Processing Unit, Machine Instruction Processor.- IBM Corp. Form No. SY33-1078, 1974

/IBM-75/ IBM Corp.: IBM /370 Model 115 Operators Library - Procedures.- IBM Corporation, Form No. GA33-1514, May 1975

/IBM-76/ IBM Corp.: IBM System/370 Principles of Operations.- IBM Corp., Form No. GA-22-7000, 1976.

/IBM-76b/ IBM Corp.: OS/VS1 System Management Facilities ('SMF'), Rel. 6, 2nd edition. IBM Corp., Form No. GC24-5115, Sept. 1976.

/Losbichler-75/ Losbichler B., Muehlbacher J.R.: A Note on Programming in Paging Systems.- Proc. of INFORMATICA 75, Bled, YU., p. 1.15.1 - 1.15-5, 1975.

/Muehlbacher-79/ 1) Muehlbacher J.R., Schulz A.: Programmieren in seitenverwalteten Systemen.- Kepler Univ. Linz, Informatik-Berichte: SYSPRO TRP1/79, 1979, (out of print)

2) Muehlbacher J.R., Schulz A.: Effizientes Programmieren in seitenverwalteten Systemen.- Kepler Univ. Linz, Informatik-Berichte: SYSPRO TRP 2/79, 1979

/Muehlbacher-79b/ Muehlbacher J.R.: An Implementation of Gustavson's Fast Algorithm for Sparse Matrix Multiplication.- Angew. Informatik 1980 (in press).

/Opler-67/ Opler A.: Fourth Generation Software.- Datamation, Jan. 1967, pp. 22-24.

/Pomberger-79/ Pomberger G.: personal communication, Kepler Univ. Linz

/Rueberg-78/ Rueberg H., Wesener F.J.: Hardware- und Software Monitoring: Pulsfuehlen in Auskunfts-systemen.- ONLINE 9/78, pp. 662-665

/Svobodova-76/ Svobodova L.: Computer System Measurability. Computer vol. 9, June 1976.

/Tanenbaum-76/ Tanenbaum A.S.: Structured Computer Organization: Chapter 5.5: Virtual Memory.- Prentice Hall Englewood Cl. 1976, pp. 249-279.

/von Krogh-74/ von Krogh C.: Mikroprogrammierung des IBM Systems /370-125. in: Hasselmeier H., Spruth W.G. (eds.): Rechnerstrukturen, Oldenbourg Muenchen 1974.

List of published reports "SYSPRO"

Nr.	Author(s)	Title
2/77	J.R.Mühlbacher	Magische Quadrate und ihre Verallgemeinerung: ein graphentheoretisches Problem
*3/78	J.R.Mühlbacher	F-Factors of Graphs: a generalized Matching Problem
*4/78	G.Chroust A.Kreuzer	Dokumentation zu den IOP-Lade-Programmen
5/78	G.Chroust J.R.Mühlbacher	Rivalling Multiprocessor Organization: An approach to performance increases
*6/78	B.Losbichler	Zur Softwareausbildung im Informatikstudium: Programmiermethodik einer phasenorientierten Softwareentwicklung
7/79	J.R.Mühlbacher	A case study for programming in a paged environment: An implementation of Gustavson's fast algorithm for sparse matrix multiplication
*8/79	J.R.Mühlbacher	µ-LAB "Mikroprozessor - Software - Labor" Ein Beitrag zur Ausbildung in Praktischer Informatik
*9/79	F.G.Duncan J.R.Mühlbacher	Storage structure for rivalling multiprocessor organization
10/79	J.R.Mühlbacher F.X.Steinparz	Canonical F-Factors of Graphs
11/80	J.R.Mühlbacher	Full table scatter storage parallel searching
12/80	G.Chroust A.Kreuzer K.Stadler	A Microprogrammed Page Fault Monitor

List of published reports "SYSPRO/TRP"

*1/79	J.R.Mühlbacher A.Schulz	Programmieren in Seitenverwalteten Systemen
*2/79	J.R.Mühlbacher	Effizientes Programmieren in Seitenverw. Systemen (I)
3/79	J.R.Mühlbacher	Eine Fallstudie über Programmieren in Seitenverw. Systemen: Implementierung des Gustavson'schen Algorithmus zur Multiplikation dünnbesetzter Matrizen
*4/79	J.R.Mühlbacher F.X.Steinparz	Kanonische F-Faktoren auf Graphen
5/79	J.R.Mühlbacher	Hardwareentwicklungen u.ihr Einfluß auf Softwaresysteme
6/79	F.G.Duncan J.R.Mühlbacher	Personal Computing in Computer Science Education: How to solve the education Paradox
7/79	J.R.Mühlbacher	Zukunftsaspekte in der Datenverarbeitung
8/79	J.R.Mühlbacher	Algorithmen u.Datenstrukturen für Rivalisierende Prozesse
*9/79	J.R.Mühlbacher	Effizientes Programmieren in Seitenverw. Systemen (II)
10/80	J.R.Mühlbacher	Zur Verwendung des Siemens-Microset 8080 als PROM-PROGRAMMIERER

*out of print

List of published reports "SYSPRO"

Nr.	Author(s)	Title
2/77	J.R.Mühlbacher	Magische Quadrate und ihre Verallgemeinerung: ein graphentheoretisches Problem
*3/78	J.R.Mühlbacher	F-Factors of Graphs: a generalized Matching Problem
*4/78	G.Chroust A.Kreuzer	Dokumentation zu den IOP-Lade-Programmen
5/78	G.Chroust J.R.Mühlbacher	Rivalling Multiprocessor Organization: An approach to performance increases
*6/78	B.Losbichler	Zur Softwareausbildung im Informatikstudium: Programmiermethodik einer phasenorientierten Softwareentwicklung
7/79	J.R.Mühlbacher	A case study for programming in a paged environment: An implementation of Gustavson's fast algorithm for sparse matrix multiplication
*8/79	J.R.Mühlbacher	µ-LAB "Mikroprozessor - Software - Labor" Ein Beitrag zur Ausbildung in Praktischer Informatik
*9/79	F.G.Duncan J.R.Mühlbacher	Storage structure for rivalling multiprocessor organization
10/79	J.R.Mühlbacher F.X.Steinparz	Canonical F-Factors of Graphs
11/80	J.R.Mühlbacher	Full table scatter storage parallel searching
12/80	G.Chroust A.Kreuzer K.Stadler	A Microprogrammed Page Fault Monitor
13/80	E.Feilmair K.Stadler	A Microprogrammed CSECT Monitor

List of published reports "SYSPRO/TRP"

*1/79	J.R.Mühlbacher A.Schulz	Programmieren in Seitenverwalteten Systemen
*2/79	J.R.Mühlbacher	Effizientes Programmieren in Seitenverw. Systemen (I)
3/79	J.R.Mühlbacher	Eine Fallstudie über Programmieren in Seitenverw. Systemen: Implementierung des Gustavson'schen Algorithmus zur Multiplikation dünnbesetzter Matrizen
*4/79	J.R.Mühlbacher F.X.Steinparz	Kanonische F-Faktoren auf Graphen
5/79	J.R.Mühlbacher	Hardwareentwicklungen u.ihr Einfluß auf Softwaresysteme
6/79	F.G.Duncan J.R.Mühlbacher	Personal Computing in Computer Science Education: How to solve the education Paradox
7/79	J.R.Mühlbacher	Zukunftsaspekte in der Datenverarbeitung
8/79	J.R.Mühlbacher	Algorithmen u.Datenstrukturen für Rivalisierende Prozesse
*9/79	J.R.Mühlbacher	Effizientes Programmieren in Seitenverw. Systemen (II)
10/80	J.R.Mühlbacher	Zur Verwendung des Siemens-Microset 8080 als PROM-PROGRAMMIERER

*out of print