JOHANNES KEPLER
UNIVERSITÄT LINZ

TECHNISCH-NATURWISSENSCHAFTLICHE FAKULTÄT
INSTITUT FÜR INFORMATIK

A Microprogrammed CSECT Monitor

E. Feilmair*, K. Stadler*

SYSPRO 13/80 April 80

INFORMATIK-BERICHTE

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

A Microprogrammed CSECT Monitor

E. Feilmair*, K. Stadler*

SYSPRO 13/80 April 80

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

*Informatik - Systemprogrammierung
Universität Linz
A-4040 Linz/Auhof

A Microprogrammed CSECT Monitor

E. Feilmair, K. Stadler

Institute for Informatics
J. Kepler University Linz
Linz, Austria

Abstract:
In order to investigate the influence of different programming techniques on the paging behaviour of the resulting object program, a monitor to observe the dynamic transition of control between control sections (CSECTs) was implemented in firmware.
The paper discusses the general design philosophy and properties of this tool, describes the specific implementation and shows some results of its use.

1.0 CONTROL SECTIONS (CSECTS) AND PAGING BEHAVIOUR

1.1 CONTROL SECTIONS

An executable load module (/IBM10/) as generated e.g. by the IBM PL/I compilers (/IBM11/, /IBM09/) usually consists of several so-called 'control sections' ('CSECTS'). CSECTs are a means to combine (link-edit) programs compiled at different times. They cater for different lifetime of various objects (e.g. STATIC versus AUTOMATIC variables) and increase the flexibility of loading code into storage.

A Control Section is a piece of code (instructions or data), which forms a unit. All parts of a control section are loaded with constant offsets from one to another. Therefore a CSECT is the smallest relocatable unit. To generate an executable 'load module' the linkage editor (/IBM10/) combines all control sections and resolves all references between addresses in different control sections (/IBM18/).

1.2 PAGING

When the linkage editor prepares the load module it ignores the location of CSECT with respect to page boundaries and loads the CSECTs with minimal space between them into storage. CSECTs are usually ordered in the sequence in which the linkage editor resolves the external references. This is not necessarily a disadvantage since this sequence often corresponds to the actual dynamic sequence, it is - on the other hand - not optimal since the resolution of references follows <u>static</u> criteria.

During execution only the necessary pages are resident (/IBM13/). If there is a CSECT with 8 bytes, say, still the whole page containing this CSECT must be loaded, although the rest of the page might not be needed.

It has been shown (/HA71/, /FE74/, /FE76A/, /FE76B/) that by rearranging the CSECTs almost always the number of page faults can be reduced. A basis for such a performance improvement is the detailed analysis of the Linkage-Editor Cross reference list, which shows all references between every pair of CSECTs (cf. /PR78/).

A logical consequence of such initial improvements is the monitoring of the actual dynamic behaviour of the program in order to find information on the order in which CSECTs are referenced (and thus building a 'CSECT reference string') and which CSECTs are referenced 'concurrently' (where 'concurrency' means 'referenced within a specified time interval'). Based on these data about the dynamic program behaviour one can try to rearrange CSECTs in such a way that 'concurrently used' CSECTs are located in as few pages as possible.

One should point out, however, that the success of such a reordering of CSECTs is heavily dependant on the data used by the monitored program. It is a necessity to monitor 'representative' programs, otherwise the paging behaviour could be improved for some special data and deteriorate for others. Based upon an appropriate set of data, however, one can achieve improvements for all possible input data.

2.0 MEASURING CSECT-TRANSITIONS

2.1 FIRMWARE MONITORING

Firmware Monitoring attracts increased attention due to the fact that it combines many advantages of hardware and software monitors without entailing many of their disadvantages (/AR79/, /BA74/, /DE77/, /CH80/). A main advantage of firmware monitoring with respect to a CSECT monitor is its speed (the test for CSECT change must be made at every instruction) and its ability to directly access all internal registers (especially address registers).

2.2 PROGRAM EVENT RECORDING (IBM)

One method to record programming behaviour is to utilize IBM's 'Program Event Recording' ("PER",cf. /IBM02/). It is a standard firmware utility on Model IBM/370-115. PER allows optionally to monitor the following events:

- successful execution of a jump/branch instruction
- change of the contents of specified general purpose registers
- execution of instructions in specified ranges of main storage
- change of the contents of specified ranges of main storage.

PER is activated by setting certain bits in Control Register 9 and one bit in the Program Status Word (PSW). Depending on the setting of these bits PER will recognize an event and will upon on occurrence of this event generate a program interrupt.

The actual type of the event will be remembered by setting specific bits in main storage location 150. In addition location 152 contains the address of the instruction which caused the PER interrupt.

Application of PER causes considerable reduction in execution time for all instructions. It is therefore wise, to activate PER only for those parts of the program, where it is actually needed (by changing the 'PER'-bit in the PSW).

It has to be noted that the PER-interrupt has to be handled by a user-specified Interrupt-Service-Routine (ISR). The regular ISR supplied with the system does not cater for handling of PER interrupts. At first we used PER together with a service routine to handle the generated interrupts. For practical usage it turned out that the execution time increase (including the ISR) amounted to a factor of 8 to 10. Since we wanted to investigate large programs this factor was very serious. We recognized that for our investigation of the CSECT reference strings a monitor was needed which registers only the entry and exit from CSECTs.

2.3 THE CSECT-MONITOR

To improve the execution time a special CSECT Monitor ('CSM') was developed, which tried to reduce the overhead in the firmware. This proved to be feasible, since PER is a very general monitoring tool and execution time improvements could be achieved at the cost of less flexibility and generality.

The key to the simplification was the fact that only branches can cause a CSECT change. Thus only branch instructions have to be subjected to monitoring at all.

CSM has to fulfil certain requirements:

* It should be possible to replace CSM by PER any time without change to the ISR or the monitor routine (this would bring a certain portability).

* The execution time overhead should be as small as possible.

* A program interrupt should be generated whenever a new CSECT is entered.

* Switching on and off should be done via the PER-Bit in the PSW (in the same way as for PER).

* the address of the first executed instruction in the new CSECT should be stored at address 152 (same as PER).

* Bit 8 of the Program Interrupt Code should be set (flag signalling a PER (=CSM) interrupt).

3.0 IMPLEMENTATION

CSM consists of two parts (cf. Fig. 1) which communicate via a FLAG (Bit 0 in Control Register 5) and via a control storage location where the first halfword of the current machine instruction is stored.

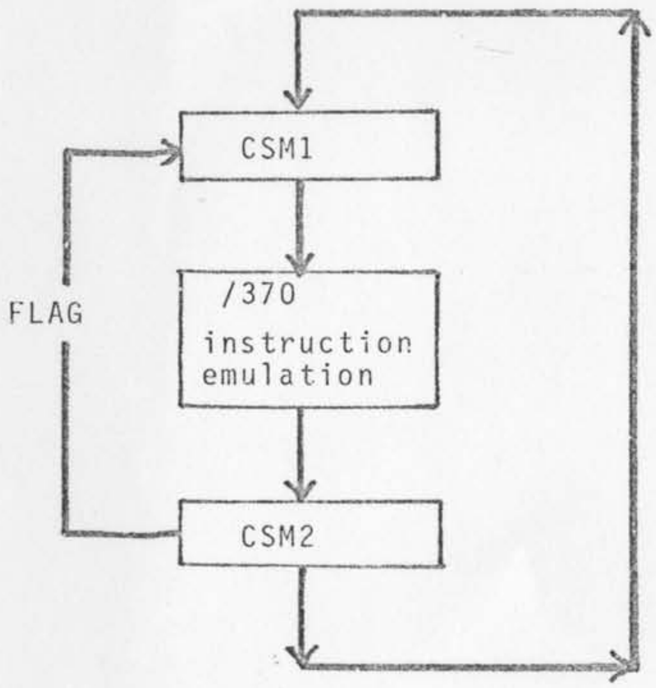

Fig. 1

Whenever CSM2 detects a branch instruction it sets a flag for CSM1. When the FLAG is on (and CSM is enabled, as indicated by the Control Register 9) then CSM1 generates the current instruction address (which will invariably be one of the two possible targets of a branch). In addition the /370 emulator's STATUS register is flagged, signalling an 'exceptional condition'.

CMS2 uses the generated instruction to test whether it lies within or outside the CSECT boundaries recorded in Control Registers 10 and 11. The CSECT boundaries are set by the ISR when a new CSECT is entered. If the address lies outside the current CSECT a change has taken place and a monitor event is recognized. This is then handled by the regular PER microprograms.

The use of the FLAG reduces the address computation in CMS1 to instructions following a branch instruction and thus saves considerable overhead.

The above description is necessarily simplified. In addition some more tests on the PER enablement are included to save unnecessary execution and a few precautions are made to avoid illegal data to be used in case of interrupts.

4.0 APPLICATIONS

The CSM has been successfully used to monitor the in-house developed text processing system MTVS. It is a large PL/I program consisting of 94 CSECTs and a size of some 220k Bytes (/MUE78/). Based on the data provided by CSM its CSECTs were rearranged. The improvements are summarized in Table I. The data show that a greater improvement can be achieved for longer programs (TEST2), since the initialization and termination of a program was not reordered.

CSM has also been applied to the MODULA Compiler being developed in Linz (/PO80/). To our surprise no improvements could be achieved, a fact which seems to imply that this compiler's CSECT organization is near optimal.

5.0 SUMMARY

By stripping an existing firmware monitoring tool (IBM's PER) to the absolute necessary functions the CSECT Monitor (CSM) could achieve a reduction of the slow-down of execution time by almost half. The slow-down for CSM-monitored programs is still high (a factor of 5 to 6), but considerable less than the equivalent PER-monitored program (factor 8 to 10). The improvement depends on the type of monitored program: For programs with many branch instructions and frequent CSECT changes CSM is not much faster than PER, since the major part of the time is spend in interrrupt generation and in the interrupt service routine (ISR), which are the same for both CSM and PER.

TABLE I

	original version	reordered version	% change
TEST 1			
page-in	3613	3071	- 15
page-out	942	928	- 1.5
CPU-time (sec.)	343	337	- 1.5
Start/Stop Time (sec.)	614	580	- 5.5
TEST 2			
page-in	8954	6669	- 25
page-out	2672	1969	- 26
CPU-time (sec.)	1120	1088	- 2.8
Start/Stop Time (sec.)	1721	1552	- 10

6.0 FUTURE PLANS

6.1 VERTICAL MIGRATION OF INTERRUPT SERVICE ROUTINE

Since most of the time is spent in generating and processing program interrupts we currently work on an improved version of CSM, where the time-consuming generation of interrupts is avoided and at the same time the function of the ISR is taken over by a microprogram.

6.2 MONITORING OF DATA CSECTS

Currently the method is only applied to program CSECTs, i.e. to pieces of code containing executable code. We study the possibilities of implementing a data CSECT monitoring tool on the basis of firmware monitoring. Basically our approach could be applied to data CSECTs, i.e. to areas of storage which contain data on which the program works. It would be necessary to monitor all instructions which access (read or write) memory. The implementation effort is thought to be manageable, since operand fetch/store is done in a few subroutines of the emulator (at least on our machine, but very likely on similar machines, too).

Our current impression is, that major improvements of program performance can be achieved via reordering of data CSECTs. The rational is that an executable program contains more data CSECTs than program CSECTs (L. Richter cites a ration of 2:1) and that references to data CSECTs are supposedly at least twice as high as references to program CSECTs. Furthermore data CSECTs are usually smaller (e.g. for each PL/I STATIC EXTERNAL variable a separate CSECT is generated) and thus can be more effectively combined into a single page.

8.0 REFERENCES

/AR79/ 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.

/BA74/ Barnes D.H., Wear L. L.: Instruction Tracing via Microprogramming. MICRO 7, 7th Annual Workshop on Microprogramming, ACM 1974, pp. 25 - 27.

/CH80/ Chroust G., Kreuzer A., Stadler K.: A Microprogrammed Page Fault Monitor. Kepler Univ. Linz, Informatik-Berichte: SYSPRO 1980.

/DE77/ de Blasi N., degli Antoni G.: Profile Finder - A Firmware Instrument for Program Measurements. EUROMICRO Newsletter Vol. 3 (1977), No. 1, pp 27-33.

/FE74/ Ferrari D.: Improving Locality by Critical Working Set. CACM, Vol. 17, No. 11, pp. 614 - 620, 1974.

/FE76A/ Ferrari D., Lau E.: An Experiment in Program. Restructuring for Performance Enhancement. Proc. of 2. Int. Conf on Software Engineering, IEEE, pp. 146 - 150, 1976.

/FE76B/ Ferrari D.: Improvement on Program Behaviour. Computer Vol. 9 (1976), No. 11, pp. 39 - 47.

/HA71/ Hatfield D.J., Gerald J.: Program Restructuring for Virtual Memory. IBM Systems J. Vol. 10, (1971), No. 3, pp. 168 - 192.

/IBM02/ IBM System /370 Model 115 Functional Characteristics, GA33-1510-1, 1974.

/IBM09/ OS PL/I Optimizing Compiler: Programmers Guide, SC33-0006-3.

/IBM10/ OS/VS Linkage Editor and Loader, GC26-3813-5.

/IBM11/ OS PL/I Checkout and Optimizing Compilers: Language Reference Manual, GC33-0009-4.

/IBM13/ OS/VS1 Planning and Use Guide, GC24-5090-6.

/IBM18/ OS PL/I Optimizing Compiler: Execution Logic, SC33-0025-2, 1973.

/MUE78/ Muehlbacher J.R., Losbichler B.: Textverarbeitungssystem MTVS - internal documentation, Kepler Univ. Linz 1978.

/PO80/ Pomberger G.: MODULA-Compiler, - report, in preparation, 1980.

/PR78/ Projekt: Effizientes Programmieren in seitenverwalteten Systemen, Zwischenbericht Nr. 1, SYSPRO-TRP 2/79.

1026642705