SYMBOL

A BIT SLICE MICROPROCESSOR BASED PAGE REPLACEMENT CONTROLLER FOR THE SYMBOL 2R COMPUTER SYSTEM

TIM I. MIKKELSON

MCS72-03642-CL7704

CYCLONE COMPUTER LABORATORY
IOWA STATE UNIVERSITY 1977

ISU-CL-7704

A BIT SLICE MICROPROCESSOR BASED PAGE REPLACEMENT CONTROLLER FOR THE SYMBOL 2R COMPUTER SYSTEM

by

Tim I. Mikkelsen

This report was submitted to the Graduate Faculty at Iowa State University in partial fulfillment of requirements for the degree of Master of Science.

(Note that the appendices in the original thesis are not included in this electronic version.)

September 1977

Table of Contents

Introduction

SYMBOL 2R is a multiprocessor computer system with an unusual combination of design premises:

1. User oriented high level language implemented in hardware
2. Provisions for interactive computing
3. Virtual memory.

In the years following the system's design there have been advances in several areas of computer systems, notably page replacement algorithms for demand paging virtual memory systems. This paper describes a controller that implements a new page replacement algorithm for the virtual memory of the SYMBOL system. The approach taken in this design removes some of the memory management functions from the mainframe and puts them in the controller.

There are several reasons for the design and construction of this controller:

1. Demonstrating the applicability of microprocessors within mainframe computers
2. Improving the paging policy of SYMBOL
3. Acquiring address traces
4. Acquiring data as a general purpose programmable monitor for SYMBOL hardware

Using microprocessors to perform mainframe functions is important because it allows the mainframe to spend its time on user programs, not system overhead. The address traces can be used to simulate SY MBOL's memory so that paging policies can be compared. The address traces and general data acquisition can be useful in debugging hardware problems in SYMBOL.

This paper deals with three diverse areas: SYMBOL, page replacement algorithms, and bit-slice microprocessors. Section II contains a description of SYMBOL and its paging policy- Sections III and IV propose two algorithms and possible implementations. Section V discusses how to interface SYMBOL and the new controller. Sections VI and VII explain the criteria for choosing a microprocessor family and the basic architecture of that family as it applies to this controller. Controller circuit diagrams, controller functions, and the microcode for the selected algorithms appear in the appendices.

SYMBOL Background

General

SYMBOL is a non-homogeneous multi-processor system designed for interactive computing. The system consists of several independent, dedicated hard-wired processors communicating over a common bus- Some of the major processors are the central processor (CP) , the system supervisor (SS), the translator (TR), the channel controller (CC) which provides communication between a user and his program, the input output processor (IP) and the memory controller (MC).

The programmer only sees a high level memory, which consists of two data types - character string scalars of arbitrary length and structures of strings of arbitrary size and shape. The size of the strings and structures can vary dynamically and the system takes care of allocation, physical location and de-allocation. The de-allocated memory is collected by another processor - the memory re-claimer (MR). The various processors rely on logical storage, a large supply of storage strings allocated by the MC, to provide the user with the high level storage. These storage strings are allocated in eight word groups and when a processor has filled up a group the MC appends another group to the storage string. For a more extensive description of SYMBOL refer to (8, 9, 10, 12).

SYMBOL Page Replacement

The virtual storage that the MC deals with consists of 16 million words, divided in to 64 K (where K = 1024) pages of 256 words. Each word consists of 64 bits and is equivalent to eight characters. The physical storage that is implemented consists of 32 pages of core storage and 4096 pages of drum memory. The memory controller and the system supervisor take care of managing the physical store. When a request for a page not in core is received by the memory controller a page fault signal is sent to the system supervisor which executes an algorithm to determine which page is the most likely candidate for swapping with the requested virtual page. The current replacement algorithm is a form of first-in, first-out. The information used by the algorithm is updated only on unusual events - page faults, user pauses, user initiations, et cetera. A description of the algorithm follows:

1) When a page is transferred to core the core frame number is added to the bottom of a core frame list with its page residency index (PRI) , which indicates the reason for the page being in core, set to:
    a) PRI=1 if the page belongs to a process in the input output processor or is a non-name-table page belonging to a process in the translator
    b) PRI=2 if the page belongs to a process that is not on the top of the central processor queue or if the page is a name-table page belonging to a process in the translator
    c) PRI=3 if the page belongs to the top process in the central processor.
2) When a process is made inactive for a period of time, for example when a user presses the pause key at his terminal, the page residency index (PRI) is set to 0.

3) When a page fault occurs take note of the page pushing priority (PPP), the reason for bringing the page in, and go through the following:
    a) Search through the core frame list for a page where PPP is greater than PRI. During the search keep track of the last page where PPP equaled PRI. If no page is found with its PRI less than the incoming page's PPP then go on to step b.
    b) If there is a page with PPP = PRI then the selected page is the one to push. If there is no page that has met this criteria then go on to step c.
    c) Temporarily abandon the paging task but leave the process unblocked so when it receives its next quantum of service it will try again.

Note that in the above description, a process is some portion of the execution of a user's interactive program (including text entry, editing, translation, and actual execution).

Algorithm Selection

Programs tend to follow the principle of locality, which states:

1. A program distributes its memory references non-uniformly over its pages
2. The frequency that a page is accessed over time tends to change slowly
3. There is a high correlation between the immediate past and the immediate future of memory references.

It is because of locality and its effects that page replacement algorithms like least recently used and working set provide good performance over random or arbitrary paging policies. A program that has little or no locality will have no need of a complex algorithm.

Agrawal, in his dissertation (1, 11), discussed and simulated several replacement policies. From his report it was shown that page replacement policies do provide an improvement in the SYMBOL paging policy. The policies he simulated were simple stack and queue algorithms. Examples are least recently used and first-in first-out. His simulation showed that least recently used (LRU) worked best by a small margin. So, to begin with, the controller should be able to perform the LRU policy

Since Agrawal has finished his dissertation, as with SYMBOL, new developments have occurred. In particular, Agrawal did not simulate the working set replacement policy. It is difficult to say how working set would perform on an unusual system like SYMBOL. Unfortunately the data collection hardware has been dismantled and the simulation software (with its sample reference strings) is not re-constructible. Remember that data collection is one of the intended functions of the page replacement controller. Even though there is no empirical reason, working set is a prime candidate for implementation by virtue of its acceptance and performance elsewhere.

Algorithm Implementation

Least Recently Used

When a page fault occurs the least recently used algorithm selects for replacement that page not accessed for the longest number of references. An implementation might be a queue with additions (memory references) to the front of the queue and deletions from any part of the queue (these memory references being added to the front). The least recently used page is the one at the end of the queue.

One of the limiting factors in the algorithm is the updating of the queue. This means that deletions and additions to the queue should be as fast as possible. There is no time for searching and reordering a linear list. A circular doubly linked list is used because of its ease of update.

The suggested implementation is:

1) System start up - link all pages together
2) Page fault - take the page at the bottom of the list, put it at the top of the list and then pass its number to the system supervisor
3) Page access –
    a) Page is on the top of the list - do nothing
    b) Page is on the bottom of the list – shift the pointers so that the top pointer points to the page accessed and the bottom pointer points to the page that was above the previous bottom page
    c) Otherwise - remove the element corresponding to the page being accessed and add it to the top of the list and set the top pointer accordingly

Note that this policy will always produce a push-able page. The microcode for the LRU algorithm is in Appendix C.

Working Set

As stated earlier, programs tend to exhibit locality of memory references. The program can be visualized as making transitions from time to time among localities, where each locality is a subset of the program's entire set of pages. The working set of a program can be defined as the pages in the current locality. A strict definition is: the working set W(t,T) at time t is the set of pages a program has accessed in the last T memory references. T is the window size. A program must have all of its working set in core or the program is not allowed to continue execution.

The working set policy can be implemented with a circular list where the list size is equal to the window size (T). Page numbers are inserted into the list on each memory reference – over-writing a previous value. The list is searched on page faults to determine which pages are no longer in a working set. Because the principle of locality deals with the nature of a single program or process it is necessary to maintain a working set list for each process. Note that because of this implementation it is necessary to go through all processes' lists whenever there is a page fault in order to determine the most push-able page).

For SYMBOL there can be 16 users (current maximum - expandable in the hardware to 31) and a typical implementation of working set might have a local list size of 64 words (the 64 word window size is taken from an actual implementation (7)). This means that when a page fault occurs the controller would have to access 1024 locations which in microcode could take up to one millisecond to perform. This would be too long a time to have the system halted. It is assumed that a memory reference operation will take 1 microsecond or less. An alternative would be to monitor which core frame is being overwritten in the queue and to keep track of the number of references to a page in all the lists. So when a page number is overwritten subtract one from the references counter for that page number. One is added to the counter for the incoming page. When a page fault occurs the controller searches through the list of counters for a counter with a value of zero.

A problem is when no page has fallen out of a working set (which will happen when the window size T exceeds the number of pages in core). There are two solutions to this problem. SYMBOL has taken one approach in it's first-in first-out algorithm: postpone any action in hopes that at some time a page will fall out of the working set. This is the approach taken in the example microcode (in Appendix C) for working set because it is simple.

The other approach is to define some policy to select a page from the various processes and seize that page. One possible criteria would be the number of pages that a process currently has in core. The controller could keep track of this value by a set of counters for each process indicating the number of pages in core. When a page is brought in the list would be incremented. When a page is selected to be thrown out it would cause the counter to be decremented. Another criteria would be to select which page has the fewest number of references. This criteria does not allow for a strict implementation of working set because it allows a process to throw out another process's page. However the window size being larger than the number of core frames available means there can not be a strict implementation of working set because a process could potentially have T pages in its window and the strict policy would require halting the program. Because of the uncertain nature of SYMBOL's performance with working set it is difficult, if not impossible, to prescribe the best policy with regards to the case where no page has fallen out of the working set.

Interfacing with SYMBOL

The current page replacement algorithm is implemented in hardware in the system supervisor. The new algorithm will reside in an add-on controller. Therefore, there has to be some hardware change in the system supervisor to circumvent the old algorithm and to allow the new controller to supply the virtual page to be pushed.

The system supervisor hardware executes the algorithm by going through phases, represented by the state of several flip-flops. To circumvent the old algorithm a phase transition will be prevented and the transition signal will be passed to the add-on controller to indicate that the controller needs to select the push-able page - Subsequently it will be necessary to restart the system supervisor at an appropriate point and allow it to perform the actual swapping, housekeeping, and possibly putting the process back on the paging request queue if it was not satisfied- This restart will be accomplished by having the add-on controller present a phase transition signal. It would be possible to have the add-on controller perform more of the system supervisor's page replacement functions, but there would be no need because the rest of page replacement is functioning satisfactorily. If other functions were added to the add-on controller it would complicate the interface and make it very difficult to remove the controller and return the system supervisor to its normal page replacement algorithm. The removal of the add-on controller would be desirable because the controller is intended for other data acquisition applications and to allow SYMBOL to function normally if the controller does not work after the hard ware modifications are made

The algorithms to be implemented will require the following lines from SYMBOL:

1) core frame number (5 bits)
2) valid virtual address (1 bit) (memory reference)
3) terminal number (4 bits) (user number)
4) master reset (1 bit)
5) phase transition (1 bit) (page fault)

The following lines are needed into SYMBOL:

1) resume system supervisor (1 bit) (page found)
2) resume system supervisor (1 bit) (no page found)
3) page to be pushed (5 bits).

In addition to the previous lines there are other lines that might be useful to future page replacement algorithms and for data acquisition applications. Some of these lines from SYMBOL are:

1. processor number
2. page priority
3. memory operation code
4. memory read/write
5. virtual address

Controller Considerations

The major consideration for the controller is speed. From Appendix E one can see that the worst case time between memory references (fastest) is 5.01 microseconds. This time dictated that a bit slice, bipolar microprocessor be used because single chip microprocessors are too slow (1 to 10 microsecond cycle times). Bit slice microprocessors currently have 100 to 300 nanosecond cycle times.

Another consideration that further enforces the need for bit slice microprocessors is their inherent ability to specify several functions to be performed simultaneously. This enhances the controller's speed. This means that the micro program word needs to be very wide in order to specify the functions independently.

Because the controller is intended to be used for several functions it is necessary to provide a micro-program control memory that is writeable. An alternative might be to provide several sets of programmable read only memories (PROM's). It would not, however, be economical to do so because PROM's that are fast enough (35 to 50 nanosecond access time) are of the fusible-link technology and can only be programmed once. There is a sizeable dollar investment in a few sets of PROMs. Another consideration is that PROMs are not conducive to developmental work

The control memory will be designed so that standard 8 bit microprocessors (like Intel 8080’s or MOS 6502’s) can load the control program. This is because microprocessor transceivers on SYMBOL can perform the downloading of the control store. It also allows the controller to function in a stand-alone configuration with an inexpensive microprocessor system like a MOS 6502 based KIM-1. The data memory will be designed so that it too can be accessed by a standard 8 bit microprocessor for similar reasons.

The actual processor family selected is the American Micro Devices Am 2900 family. At the time that the project started the only available bit slice families (that were well established) were the Am2900 family and the Intel 3000 family. In examining both families it appeared that the Am2900 series was the easiest to understand and use.

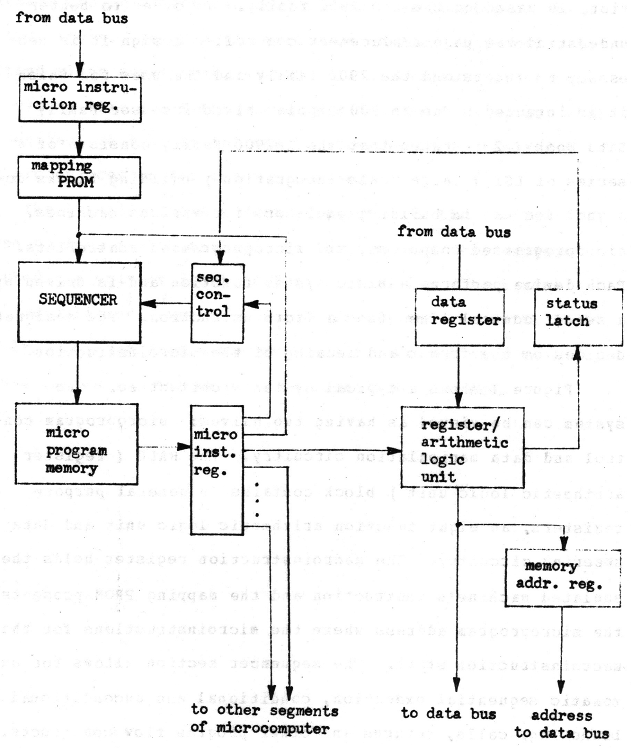
Controller Design

The add-on controller, as stated in the previous section, is based on the AMD 2900 family. In order to better understand the page replacement controller design it is necessary to understand the 2900 family and the uses for which it is intended. The Am 2900 Bipolar Microprocessor Family Data Book (2) states that the Am2900 Family consists of a series of LSI (large scale integration) building blocks designed for use in building emulators for various machines, micro-programmed computers, and micro-programmed controllers. Each device performs a basic system function and is driven by a set of control lines from a microinstruction. The designer decides on the format and meaning of the microinstruction.

Figure 1 shows a typical system architecture. The system can be viewed as having two halves: micro-program control and data manipulation circuitry. The R ALU (register arithmetic logic unit) block contains 16 general purpose registers, an eight function arithmetic logic unit and data steering circuitry. The macroinstruction register holds the emulated machine's instruction and the mapping PROM presents the micro-program address where the microinstructions for that macroinstruction start- The sequencer section allows for automatic sequential execution, conditional and unconditional branching, calls, returns and other program flow constructs.

FIGURE 1

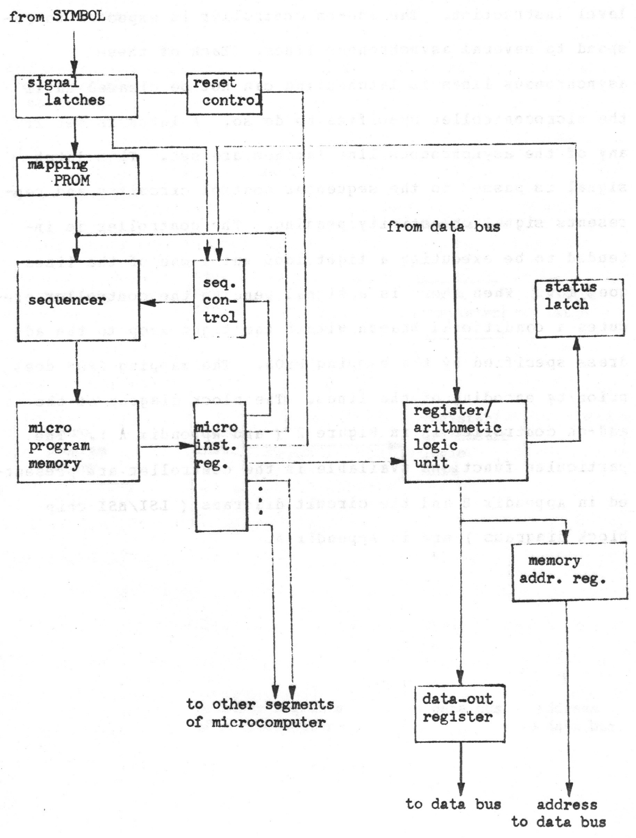
TYPICAL SYSTEM ARCHITECTURE



The major difference between the typical architecture just shown and the architecture being used for the page replacement controller is the source and meaning of the macro level instruction. The add-on controller is expected to respond to several asynchronous lines. Each of these asynchronous lines is latched and cannot be cleared until the microcontroller specifies to do so. A latch is set if any of the asynchronous line latches are set. This latch signal is passed to the sequencer control circuitry and represents signal or activity pending. The controller is intended to be executing a tight loop until one of the lines goes low. When there is a signal pending the controller executes a conditional branch within the tight loop to the address specified by the mapping PROM. The mapping PROM does priority encoding of the lines. The block diagram of the add-on controller is in Figure 2 (and Appendix A). The particular functions available in the controller are presented in Appendix B and the circuit diagrams (LSI/MSI chip block diagrams) are in Appendix A.

FIGURE 2

PAGE REPLACEMENT CONTROLLER ARCHITECTURE



Conclusions, Problems, and Extensions

On paper the design appears to work fairly well. The controller can execute the LRU and working set algorithms within the time constraints. The design shows the applicability of using microprocessors to perform a function, page replacement, within mainframe computers. Further, it shows that even a time-critical function, page replacement data collection, can be performed by microprocessors.

The main reason this design was feasible is the slow memory cycle time of SYMBOL. There are few commercial systems with five-microsecond memory cycle times being produced to day. However, the fact that it is possible to control memory paging for a slow system and the speed-up of logic (and micro-programmed processors) implies that conventional one microsecond cycle time systems might be able to use this approach to virtual memory paging. Even if a controller could not keep up, in real time, with the memory accesses, buffering could be provided. Then the controller need only be as fast as the average memory speed (which is potentially slower than the memory cycle time). When a page fault occurs in a buffered controller- assisted system the designer has a choice of halting the system and emptying the buffer or performing the algorithm without the additional information. Note that the execution of the algorithms with out the additional information does not cause errors but rather only degrades the system performance.

The problem, which arises, is that this approach to page replacement is very limited. The limiting factor is the memory references used in the controller to update the lists. LRU has eight memory references, an input operation, and three register operations in the time-critical memory access segment. This controller takes two micro-cycles for each memory reference. If a controller were designed to execute a memory reference in one micro-cycle LRU would still take at least twelve micro-cycles to update the lists during a memory reference from the mainframe. Therefore, the controller micro-cycle time has to be at least twelve times faster than the memory cycle time. This design works but it is inherently slower than a hardware or hardware-software implementation.

There are any number of possible extensions. One is extending the LRU algorithm to allow a privileged user to remove and add a page to the push-able page selection list. This allows a user to bring in a page and then remove it from the active list so it cannot be pushed out for any reason. A similar feature is currently implemented on SYMBOL that allows adding and deleting from the ICL (in core list). Another extension is to decide on a better policy for page selection in working set (when all the pages belong to some working set - none have fallen out) and implement it.

ACKNOWLEDGEMENTS

I wish to thank Professor R. J. Zingg and Professor D. Kafura for their help in the preparation of this paper. I also wish to thank all of the members of the SYMBOL staff for their comments and assistance

REFERENCES

1) Agrawal, O. P. "Applicability of Buffered Main Memory to SYMBOL 25- like Computing Structures." Dissertation. Department of Electrical Engineering. Iowa State University, 1974.
2) Am2300 Bipolar Microprocessor Family Data Book. Sunnyvale, CA. Advanced Micro Devices, Inc., 1976.
3) Chu , W. and Opderbeck, H. "Performance of Replacement Algorithms with Different Page Sizes." Computer Volume 7 Number 11 (November 1974) : 14 - 20
4) Denning, P - "Virtual Memory" Computing Surveys Volume 2 Number 3 (September 1970): 153 - 187.
5) Denning, P. and Schwartz, S. "Properties of the Working Set Model." Communications of the Association for Computing Machinery Volume 15 Number 3 (March 1972) : 191 - 198.
6) Mick, J- and Brick, J. Microprogramming Handbook. Sunnyvale, CA. Advanced Micro Devices, Inc., 1976.
7) Morris, J. "Demand Paging Through Utilization of Working set on MANIC II. " Communications of the Association for Computing Machinery. Volume 15 Number 10 (October 1972): 867 - 872.
8) Richards, H. Jr. and Zingg., R. J. "The Logical Structure of the Memory Resource in the SYMBOL 2R Computer." Special Report. NSF-OCA-GJ33097-CL7307 - Cyclone Computer Laboratory. Iowa State University, November, 1973.
9) Richards, H. Jr. and Oldehoeft, A. "Hardware-Software Interactions in SYMBOL 2R's Operating System." Special Report - NSF-OCA-GJ 3 3097-CL 7401 - Cyclone Computer Laboratory. Iowa State University, November, 1974
10) Zingg, R.J. and Richards, H. Jr. - "SYMBOL: A System Tailored to the Structure of Data." Special ReportISU-CCL-7302. Cyclone Computer Laboratory. Iowa State University, June, 1973.
11) Agrawal, O. , Zingg, R.J. and Pohm, A.V. "Applicability of 'Cache' Memories to Dedicated Multiprocessor Systems." Digest of Papers CompCon Spring 77. IEEE Computer Society : 74 - 76
12) Richards, H. Jr. and Wright, C. Jr. "Introduction to the SYMBOL 2R Programming Language." Special Technical Report. NSF-OCA-G.J.33097-CL7306. Cyclone Computer Laboratory. Iowa State University, October 1973.