Slave Memories and Dynamic Storage Allocation

M. V. WILKES

Summary

The use is discussed of a fast core memory of, say, 32 000 words as a slave to a slower core memory of, say, one million words in such a way that in practical cases the effective access time is nearer that of the fast memory than that of the slow memory.

Introduction

In the hierarchic storage systems used at present, core memories are backed up by magnetic drums or disks which are, in their turn, backed up by magnetic tape. In these systems it is natural and efficient for information to be moved in and out of the core memory in blocks. The situation is very different, however, when a fast core memory is backed up by a large slow core memory, since both memories are truly random access and there is no latency time problem. The time spent in transferring to the fast memory words of a program which are not used in a subsequent running is simply wasted.

I wish in this note to draw attention to the use of a fast memory as a slave memory. By a slave memory I mean one which automatically accumulates to itself words that come from a slower main memory, and keeps them available for subsequent use without it being necessary for the penalty of main memory access to be incurred again. Since the slave memory can only be a fraction of the size of the main memory, words cannot be preserved in it indefinitely, and there must be wired into the system an algorithm by which they are progressively overwritten. In favorable circumstances, however, a good proportion of the words will survive long enough to be used on subsequent occasions and a distinct gain of speed results. The actual gain depends on the statistics of the particular situation.

Slave memories have recently come into prominence as a way of reducing instruction access time in an otherwise conventional computer.[1,2] A small, very-high-speed memory of, say, 32 words, accumulates instructions as they are taken out of the main memory. Since instructions often occur in small loops a quite appreciable speeding up can be obtained.

One method of designing a slave memory for instructions is as follows. Suppose that the main memory has $64K$ words (where $K = 1024$) and, therefore, 16 address bits, and that the slave memory has 32 words and, therefore, 5 address bits. The slave memory is constructed with a word length equal to that of the main memory plus 11 extra bits, which will be referred to as *tag* bits. An instruction extracted from register r of the main memory is copied into register r (mod 32) of the slave memory and, at the same time, the 11 most significant bits of r are copied into the 11 tag bits. For example, suppose $r = 10\,259$, that is, $320.2^5 + 19$. The instruction from this register is copied into register 19 of the slave and the number 320 is copied into the tag bits of that register.

Whenever an instruction is required, the slave is first examined to see whether it already contains that instruction. This is done by accessing the register that might contain the instruction [namely, register r (mod 32)], and examining the tag bits to see whether they are equal to the 11 most significant digits of r. If they are, the instruction is taken from the slave; otherwise, it is obtained from the main memory and a copy left in the slave. If the system is to preserve full freedom for the programmer to modify instructions in the accumulator, it is necessary that every time a writing operation is to take place, the slave shall be examined to see whether it contains the word about to be updated. If it does, then the word must be updated in the slave as well as in the main memory.

Large Slave Memory

So far the slave principle has been applied to very small superspeed memories associated with the control of a computer. There would, however, appear to be possibilities in the use of a normal sized core memory as a slave to a large core memory, and I will now discuss various ways in which this might be done. I shall be concerned primarily with a computer system designed for on-line time-sharing in which a large number of user programs are held in auxiliary storage and activated, in turn, according to a sequence determined by a scheduling algorithm. When activated, each program runs until it is either completed or held up by an input/output wait, or until the period of time allocated to it by the scheduling algorithm is exhausted. Another program is then activated. See Corbató.[3]

Consider a computer in which a working memory of, say, $32K$ and 1-μs access time is backed up by a large core memory of, say, one million words and 8-μs access time. In the simplest scheme to be described, programs are split into $32K$ word blocks, each user making use of one or more blocks for his program. The large core memory is provided with a base register, which contains the starting address of the $32K$ block currently active. What we wish to avoid is transferring the whole block to the fast core memory every time it becomes active; this would be wasteful since chances are only a small fraction of the $32K$ words will actually be accessed before the block ceases to be active. If the fast core memory is operated on the slave principle, no word is copied into it until that word has actually been called for by the program. When this happens, the word is automatically copied by the hardware into the fast memory, and the fact that copying has taken place is indicated by the first of two tag bits being changed from a 0 to a 1. When any reference to storage takes place the fast memory is accessed first,[4] and, if the first tag bit is a 1, no reference is made to the large memory; this is true whether reading or writing is called for. If a word in the fast memory is changed, a second tag bit is changed from 0 to 1. Two tag bits are all that are required in this system.

As time goes on, the fast memory will accumulate all the words of the program in active use. When the number in the base register is changed so that a new program becomes active in the place of the one currently active (a change that is brought about by the supervisor), a scan of the fast memory is initiated. Each register is examined in turn and, if the first tag bit is a 0, no action is taken for that register. No action is similarly taken if the first tag bit is a 1 and the second tag bit is a 0. If, however, both tag bits are 1's, the word in the register under examination is copied into its appropriate place in the large memory.

Many variants of the simple scheme are possible. The tag bits may, for example, be stored in a separate superspeed memory. A

[1] Takahashi, S., H. Nishino, K. Yoshihiro, and K. Fuchi, System design of the ETL Mk-6 computers. *Information Processing 1962 (Proc. IFIP Congress 62)*, Amsterdam, The Netherlands: North Holland Publishing Co., 1963, p 690.

[2] *Ferranti Computing Systems; Atlas 2*, London: Ferranti Ltd., 1963.

[3] Corbató, F. J. *Proc. 1962 Internat'l Federation of Information Processing Congress*, Amsterdam, The Netherlands: North-Holland Publishing Co., 1963, p 711.

[4] If the design of the large core memory permits, access to it can be initiated simultaneously with access to the fast memory, and cancelled if it turns out not to be required.

1024-word memory, each having 64 bits, would be suitable; such a memory could be made with an access time of about 100 ns, and would enable the scanning process to be completed more rapidly. Similarly, a number of base registers could be provided and the fast core memory divided into sections, each serving as a slave to a separate program block in the main memory. Such a provision would, in principle, enable short programs belonging to a number of users to remain in the fast memory while some other user was active, being displaced only when the space they occupied was required for some other purpose. This would present the designer of the supervisor with problems similar to those presented by an Atlas-type system of dynamic storage allocation.[5]

An alternative, and perhaps more attractive, scheme would be to retain $32K$ (or whatever the size of the fast memory may be) as the block length, but to arrange that the fast memory acts as a slave to more than one block in the main memory, it being recognized that this will lead to some overwriting of information in the slave, but will, nevertheless, on the average, be advantageous. Suppose, for example, that there are seven base registers, each containing an address of a register in the main memory at which a program block starts. Four tag bits are necessary, the first three containing either zeros or the number of one of the base registers. The fourth tag bit indicates whether a word has been altered while in the slave.

At any given time, one of the seven program blocks is active. Whenever access is required to a word in the memory, the hardware looks to see whether that word is to be found in the slave. This is done by reading the word in the appropriate place in the slave and comparing the first 3 tag bits with the number of the base register corresponding to the program block then active. If there is agreement, and if a reading operation is to be performed, the word from the slave is used and operation proceeds. If the three tag bits are all zero, the word is obtained from the main memory and a copy put into the slave memory for future use. If the three tag bits are not zero but correspond to another base register, the fourth digit is examined. If this is a zero, action proceeds as before, the word in the slave being overwritten by the word from the new program block. If, however, the fourth bit is a 1, indicating that the word has been altered while in the slave, that word is copied back into its proper place in the main memory before being overwritten by the word from the new program block. In the case of a writing operation the sequence of events is similar, except that the fourth tag bit is made into a 1 when a word in the slave is modified. Thus, if the seven programs become active in turn, they may be said to share the slave between them and, if each runs in short bursts, there is a fair chance that only a few words belonging to a particular program block get overwritten in the slave before that program block is activated again.

There will, normally, be more than seven program blocks ready to take their turn for running and the supervisor will, from time to time, change the address in one of the base registers. When this happens, a scan of the slave is initiated, and all words which belong to the program block being displaced and which have a 1 in the fourth bit of the tag, are copied into the main memory.

On the face of it, the scheme just outlined appears to offer the basis for a satisfactory two-level core storage system without involving too high a degree of complexity in the hardware.

ACKNOWLEDGMENT

The author wishes to express his gratitude to Prof. R. M. Fano, Director of Project MAC, for inviting him to participate in the project. He is also grateful to his colleagues in Cambridge, England, for discussions, particularly to Dr. D. J. Wheeler and N. E. Wiseman, who designed the slave memory of Atlas 2. G. Scarrot first suggested the idea of a slave memory to them.

[5] Kilburn, T., D. B. G. Edwards, M. J. Lanigan, and F. H. Sumner, One level storage system, *IRE Trans. on Electronic Computers*, vol 11, Apr 1962, pp 223–235.