Using the cache memory structure as applied to a 12-bit minicomputer (PDP-8/E), and the 100-ns processor constructed of H and S series TTL gates, the resultant performance is 5 to 10 times the conventional implementation with a corresponding improvement in the performance/cost ratio

Implementation of a Buffer Memory in Minicomputers

C. Gordon Bell and David Casasent

Carnegie-Mellon University
Pittsburgh, Pennsylvania

The cache memory concept was developed in an effort to improve the performance of large-scale computers. This has produced machines with the advantages of short cycle times without the added cost of all solid-state high-speed memories.

The implementation of this scheme in a minicomputer is more difficult, because it requires a determination of the performance gained for each increase in cost and, therefore, a firm understanding of the schemes directly usable in a minicomputer. Basically, the complexity added to a minicomputer by the use of a cache is a much larger portion of the machine's basic price than that of a large-scale machine. As a result, the cache scheme should not aim at finding 98% of the addressed words in cache memory. The overall cost of such a scheme would far outweigh any performance advantages gained.

C. Gordon Bell received a BS degree in Electrical Engineering and an MS degree from Massachusetts Institute of Technology, Cambridge, Mass. He is presently a professor of electrical engineering and computer science at Carnegie-Mellon University, Pittsburgh, Pa, and consultant for Digital Equipment Corp, Maynard, Mass. His research interests center around the design of computer systems.

David P. Casasent received a PhD degree from the University of Illinois. He is currently an assistant professor of electrical engineering at Carnegie-Mellon University doing research in computer architecture, high-speed digital circuits, electro-optics, and electron-optics.

The Computer

A minicomputer using a cache memory has been simulated and constructed to test the use of high-speed logic and high-speed memories. Initially a single bipolar memory was considered, but as the use of cache memories in larger computers became better understood, it became clear that the cache structure could work equally well in minicomputers.

The cache memory size may be increased in increments. Each size increase improves the performance but also increases the cost. Thus each user can add on enough memory to achieve the performance/cost combination suitable for his needs. The cycle time of the computer with an all bipolar memory is about 100 ns, and with a combined 512-word fast cache memory and a 1-μs core memory the effective cycle time is about 200 ns; thus an improvement of 5 in performance can easily be obtained over the all core memory.

The Cache

The cache memory concept has been described in various forms in the literature: the lookaside memories,[1] slave memories,[2] and associative memories[3] are several versions of the cache, and two excellent survey articles have been written by Meade[4] and Conti.[5] The purpose of the cache is to achieve the effect of an all high-speed

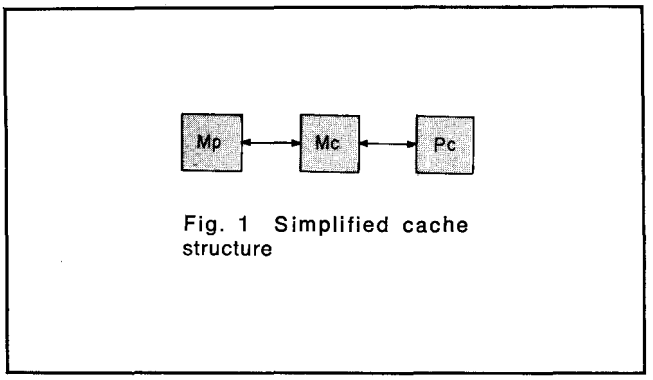

Fig. 1 Simplified cache structure

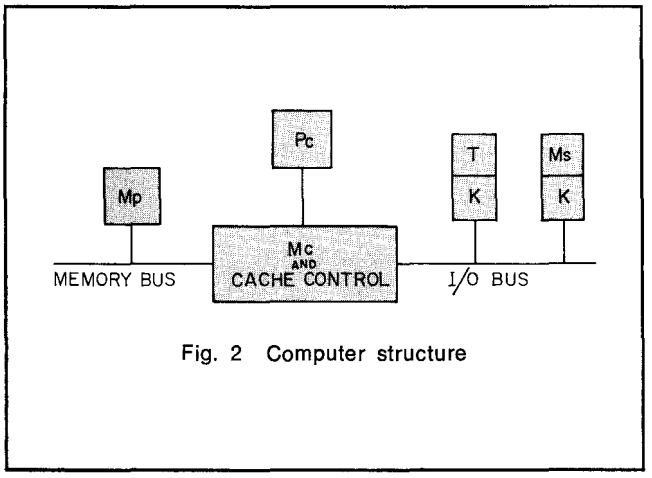

Fig. 2 Computer structure

memory by using two memories—one slow and one fast—and insuring that the data being used are in the fast memory nearly 95% of the time. This provides the advantage of speed without having to resort to an all solid-state memory.

A simplified diagram of a cache memory system is shown in Fig. 1. The operation is: the central processor, Pc, requests a word to be read from memory. If the word is in the cache memory, Mc, it is given to the processor. If the word is not in the cache, the cache requests the word from the slower primary memory, Mp. When the word is read from Mp, it is given to both Pc and Mc. Mc holds it for future reference. This system works well because references to primary memory ar in general not random; access is rather to a small loca set of addresses and usually contains series of loops.

One familiar with the cache memory literature[1-6] will recognize that all prior discussions of cache type systems have pertained to large systems where high performance was the design goal. The memory/computer price ratio is quite different between a large machine and a minicomputer. If the intriguing cache schemes of the large computers were implemented directly here, the memory/processor cost ratio would be too large. The memory size and long word length of the large machines forces some cache memory referencing techniques to be abandoned in large computers. Thus some schemes not feasible in a large computer are possible with minicomputers. The large resultant cache memory size, 16K to 32K bytes,[5] often results in a data type dependent allocation in the cache and second level associative cache memories for large computers. The problem of what word of fast memory to overwrite (eg, the least recently used one) can be completely ignored in the minicomputer case with surprisingly little difference.

Structure

A PDP-8/E was chosen as the experimental basis because of its simple structure and because if it proved feasible there, almost any other minicomputer could take advantage of the cache structure (Fig. 2) equally well.

Conti[5] has defined four main cache memory configurations. The disadvantages of each are discussed here only briefly.

(1) Fully associative buffer—This system requires extensive control circuitry. In it any word in Mp can be mapped onto any word in Mc. This scheme does not use the fact that memories have linear explicit addresses.

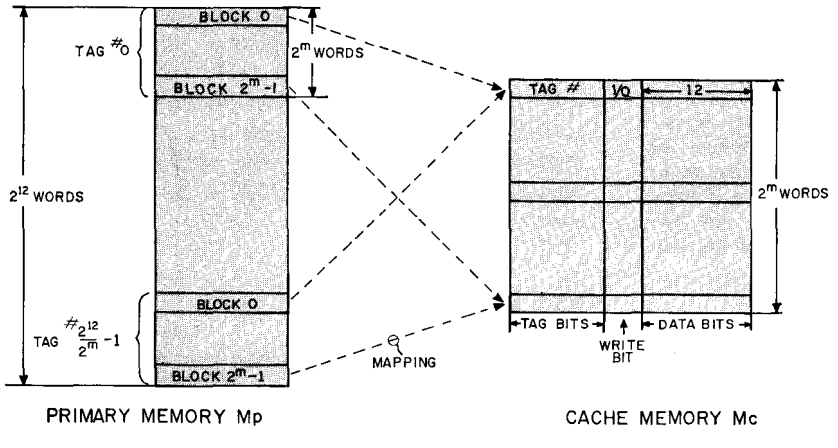

Fig. 3 Direct mapping of 2^m word cache memory from a 2^{12} word primary memory

TABLE 1

Cache System Parameters—1st Order

	Name	Explanation
Processor Parameters	t_p	Processing time
	t_x	Instruction decoding, effective address calculation, and execution times
	P_x	Conditional probabilities of various operand lengths and locations
Primary Memory Parameters	t_{cp}	Primary memory cycle time
	t_{ap}	Primary memory access time
	t_{rp}	Time to access a memory location, read it, and reset it to 0 or destroy its contents
	t_{wp}	Time to write back information given that the memory contents are zero
	m	Number of independent memory modules
	w_p	Information width of memory (number of bits simultaneously accessed from a memory module)
	w_t	Width of link between Mc and Mp (number of bits transferred in parallel)
	t_t	Time to transfer w_t bits (in both directions)
Cache Memory Parameters	t_{ac}	Cache access time
	t_{cc}	Cache cycle time
	w_c	Information width of memory
	l_h	Length of linearly addressed memory
	l_a	Number of separate associatively accessed memory parts

(2) Sector buffer—This requires that the age of each register (when it was used last) be kept track of for replacement.

(3) Direct mapping—This scheme was first described by Scarrott[6] and can be considered as a content addressable memory with hash coding with modulo 2^n. In this scheme, only one cell in memory *can* contain the address, hence the name direct mapping. It requires only one comparison circuit; its disadvantage is that in a loop two instructions might occupy the same cell and necessitate rewriting.

(4) Set associative buffer—This is a direct mapping with the memory words of double length. By choosing the correct parameters, all of the other three cases can be shown to be special cases of this one.

Due to its ease of implementation, the direct mapping scheme seemed most appropriate. Other schemes were also simulated. Because of the lack of an adequate model of minicomputer programs and due to the large number of variables that characterize a cache system (Table 1 contains a first order list), simulation was used to determine the system parameters

The organization of the 4096-word primary memory, Mp, and the 2^m word cache, Mc, are shown in Fig. 3. Mp consists of 2^{12} words (extended memory can also be added) divided into 2^{12-m} sections, each denoted by a tag number. Each section contains 2^m words or blocks. Block x from any section of Mp can only be contained in word x of Mc. Mc contains 2^m words, each word divided into three parts. The least significant 12 bits are the 12-bit word, the next most significant bit is the write bit, and the most significant $12-m$ bits are the tag bits associated with that word. The location of the word in the Mc denotes which block of Mp the word is in, and the $12-m$ tag bits of the Mc word denote the section of the Mp from which it was taken.

In operation the processor presents a 12-bit address to Mc, the least significant m-bits are used to access a word in cache, and the tag bits of the accessed word are compared with the tag part (the first $12-m$ bits) of the

TABLE 2

Simulation Results—Cache Size Variable

Mc size (words)	FOCAL	Fast-Fourier	Assembler
		(Effective cycle times in ns)	
64	450	470	420
128	360	370	320
256	290	220	290
512	200	190	170

address. If equal, the word is in Mc, otherwise the word must be fetched from Mp.

Writeback Strategy

The writeback strategy determines the action when the processor requests a word to be written back into memory. There are several possibilities: (1) always write the word in the cache and in primary memory, (2) write the word in the cache and then always write the word in primary memory when the cache cell must be replaced, (3) place a control bit (the write bit of Fig. 3) in the cache, which indicates when a word has been written and thus has to be written back in primary memory.

The simulation showed scheme (3) to be better than schemes (1) and (2). Although more complex schemes are slightly better, we chose this one. Fig. 4 presents flowcharts for processor read and write request using scheme (3), which essentially amounts to writing information back from Mc to Mp only when a word of Mc is about to be written over with a word with a new tag number. If the write bit is a 1, the Mc has been changed since it was entered from the Mp and thus the Mp must be updated.

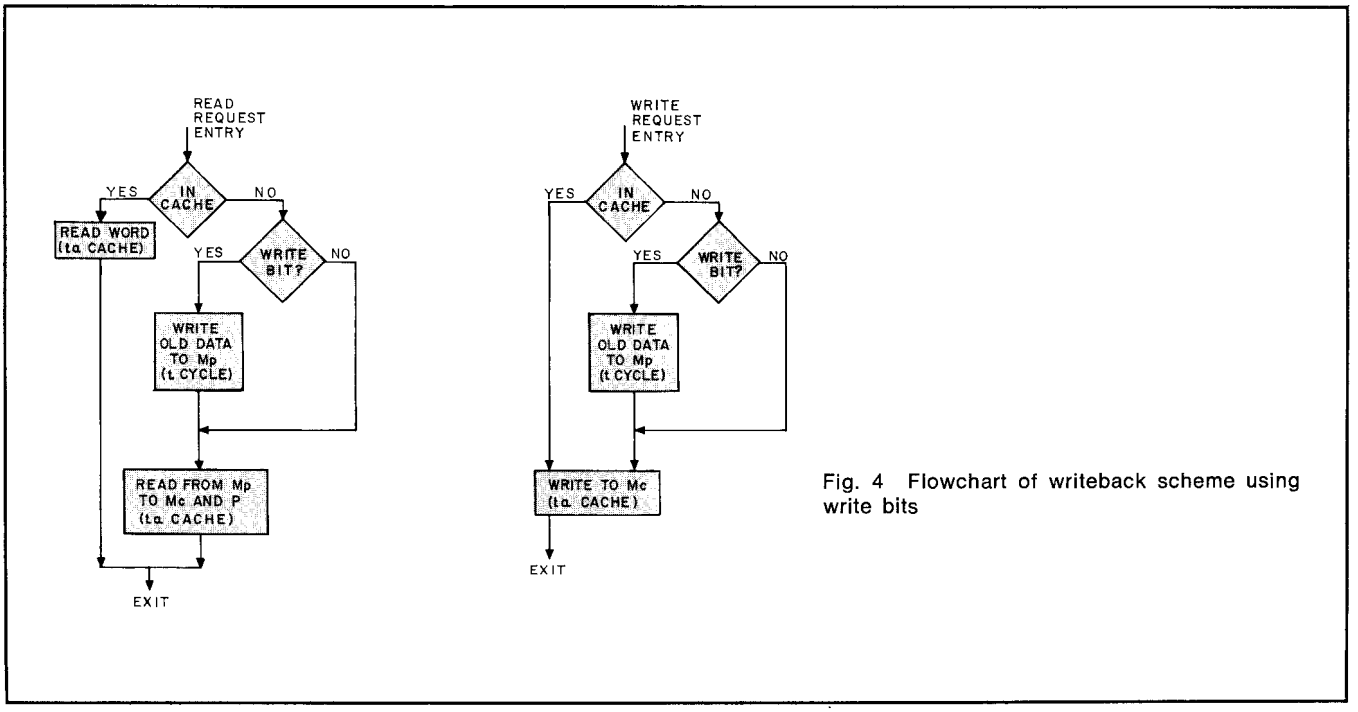

Fig. 4 Flowchart of writeback scheme using write bits

Simulation

Three benchmark programs were chosen on which to base the design: the assembler assembling a small program, fast Fourier transform, and FOCAL—an interactive interpreter—executing a small program.

	Assembler	FFT	FOCAL
Program size (locations) (active)	1800	≈1000 data, 512 program	3000
Instructions simulated	900K	1000K	600K

The simulation process was: (1) a conventional instruction simulator was modified to generate a file which recorded each access and its type (eg, instruction, defer, data) for each benchmark program, (2) access distributions were made for each program, and (3) the access file was used as input to test each simulated cache structure. FOCAL was taken as the worst-case benchmark for most of the cache simulations.

Simulation Results

The effectiveness of the scheme varies with cache size. This effect is shown in Table 2 for the three benchmarks, assuming $t_{ac} = 50$ ns, $t_{cc} = 100$ ns, $t_{wp} = t_{rp} = 500$ ns, and $t_{cp} = 1$ μs. (Refer to Table 1 for symbol glossary.)

The effect of varying the policy under which data and instructions are assigned to the cache was also considered. Table 3 gives the effective cycle times for various allocation strategies using the worst-case benchmark

TABLE 3
Simulation Results—Cache Allocation Variable

Undivided Cache—1 word wide
200 ns

Divided Cache for Instructions/data—1 word wide
Instructions only = 325 ns
Instructions ¼/data ¾ = 260 ns
Instructions ½/data ½ = 225 ns
Instructions ¾/data ¼ = 225 ns
Data only = 395 ns

Undivided Cache—2 words wide
230 ns

Divided Cache for Instructions/data—2 words wide
Instructions only = 395 ns
Instructions ¼/data ¾ = 290 ns
Instructions ½/data ½ = 250 ns
Instructions ¾/data ¼ = 185 ns
Data only = 290 ns

Divided Cache—128 words for page 0, 256 words of instructions, 128 words of data
165 ns

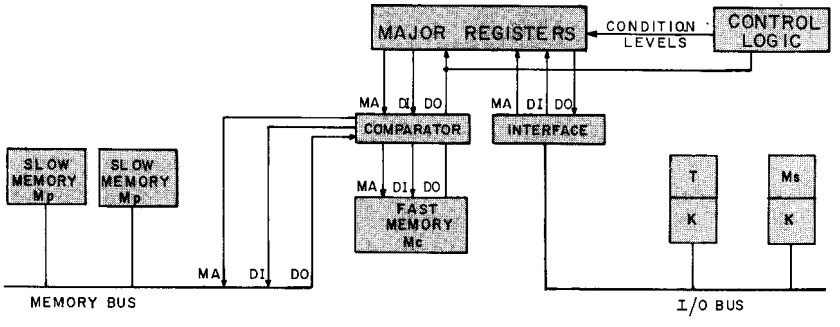

Fig. 5 Computer block diagram

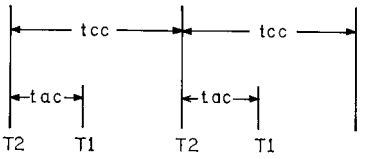

Fig. 6 Simplified timing diagram

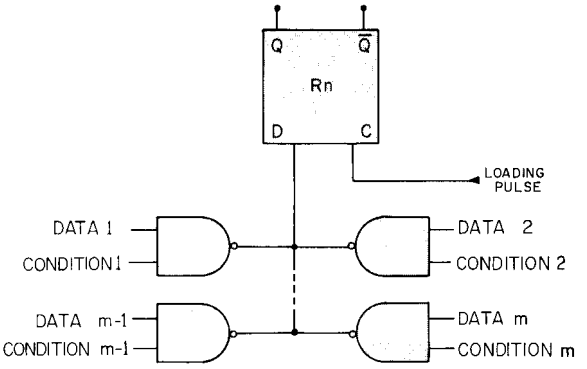

Fig. 7 General example of register loading

program, FOCAL, and a 512-word cache. In the cases considered, the cache allocation was varied from containing only instructions to containing only data. In all cases, except when one page (128 words) of cache was allocated for the frequently referenced page zero, the undivided cache yielded better simulation results.

Prototype Processor Design

A prototype has been constructed to verify the concepts and to ascertain the ease of construction. The write scheme of Fig. 4 was used. The processor was designed primarily for speed although cost was also considered. The improved performance was achieved through the use of faster circuits (H and S series TTL logic gates), and extensive use of open collector gates, plus a greatly increased parallelism of operations. Basically all operations are carried out in one clock time by calculating a number of next state alternatives.

Central Control Logic

The central control logic (Fig. 5) requires 45 ns after receipt of data-out (DO) from memory before it will deliver the new data-in (DI) to memory (in the case of a write) and/or the new memory address location (MA). This 45 ns is six gate delays plus one flip-flop set-up, hold, and propagation time. The resultant system with seven levels of logic was the most reasonable compromise between speed and cost.

The central processor unit (CPU) contains the major registers of the system plus the control logic. The comparator determines in which memory the desired data is located. The control logic produces condition levels which are used in loading the major registers with the correct data at the proper time.

The timing diagram for one fast memory cycle is shown in Fig. 6. At time T1 the DO from fast memory is valid. By time T2, the new MA and DI must be valid. In T2-T1, the processor must determine by the comparator if the data (DO) from the Mc is indeed the desired data; if not, the fast memory cycle is terminated and a slow memory cycle is begun to retrieve the dat or instruction word from Mp. Assuming that the DO is valid and is an instruction word (worst case), the processor then: (1) decodes the instruction, (2) generates the proper condition levels and the proper data inputs for the major registers, and (3) activates the write or read mode line for memory and the cycle repeats at T2. From T2 to T1, while the next word is being accessed, the processor completes its bookkeeping by loading the accumulator register, setting the major state register, and initializing the various control lines for the start of the new cycle.

The critical path occurs between T1 and T2. To enable the MA and DI to be loaded during this time the scheme of Fig. 7 is used to load bit n of some register, R. Each register has a small number of possible data inputs, m. All of these are formed each memory cycle whether they are to be used or not. While these data inputs to the registers are being formed, the control logic generates a set of condition levels corresponding to the data inputs to the registers such that, at any register loading time, only one condition level for each register is true and all others are false. Only the data associated with the true condition level can be loaded. The timing and flowchart of sequential operations are arranged to minimize the number of data and condition lines and still maintain speed.

The timing generator that controls the sequence of operations consists of two delay lines connected in two separate rings or loops. One loop is for Mc timing; the other is for Mp timing. Each loop must be able to be entered from the other during operation in the buffer

TABLE 4

Performance/Cost Comparisons

Processor	Performance Factor	Cost in $1K			Performance/Cost Ratio			P/C Ratio in Relation to All-Core System		
		Configuration			Configuration			Configuration		
		Min	Avg	Large	Min	Avg	Large	Min	Avg	Large
All Core Memory	1.0	5	10	35	0.2	0.1	0.029	1.0	1.0	1.0
Semiconductor Memory (MOS)	2.0	5	10	35	0.4	0.2	0.057	2.0	2.0	2.0
Cache Memory	5.0	10	15	40	0.5	0.33	0.125	2.5	3.3	4.3

memory mode. To increase versatility, the system timing and logic is so arranged that only either the cache or fast memory can be operated; or the slow main memory, a core or an MOS, can be operated alone, or both in a buffer memory mode. This facilitates system checkout and allows a wide range of cache memories to be investigated.

Initial prototype results have indicated that cycle and processor times slightly over 100 ns are possible. Many additional requirements are placed on the design and the system such as a preclear of the solid-state memory whenever power is first turned on.

Conclusion

Table 4 compares three system configurations (minimum, average, and large), each implemented by three different processor/memory arrangements in an existing minicomputer using: (1) all core memory, (2) all semiconductor MOS memory, or (3) cache memory.

If a performance factor of 1 is assigned to the all core memory arrangement, an improvement in performance by factors of 2 and 5 will be achieved by the MOS memory and cache memory arrangements, respectively. The cost differential between the different configurations is due, basically, to the larger memories used. As more primary memory, Mp, is added to each system the performance/cost ratio decreases for each larger configuration. However, in the case of the cache memory, the ratio decreases at a slower rate since the initial outlay for the faster processor with cache memory becomes an ever smaller part of the cost of the growing system. As the system configuration increases in size, the performance/cost ratio of the cache system also increases in comparison to that of the smaller systems which, in essence, explains the success of the cache memory in large computers as well as in minicomputers. It is conceivable that an all solid-state memory would be feasible if its improved speed versus its added cost were sufficiently high.

In this discussion we have tried to show the technical feasibility of how a cache memory would be used with a small minicomputer. Whether such a machine is truly useful can only be determined by trying to apply it. There are identifiable applications which can use the additional performance. The PDP-8 time-sharing system, TSS-8, could effectively use the additional power because there are instances of it being compute bound. Real-time processing could also use the additional speed to allow more problems to be solved. In addition, there are simple computers being used to study microprogramming; this structure would provide the equivalent amount of power and still make it possible for the user to program the machine. Finally, there are already 10,000 PDP-8s in existence, some of which might require more capacity without the inconvenience of changing to a basically large computer.

In summary, if such a machine were made available at, say, a cost of $5K for the faster processor, it would have the cost and performance characteristics shown in Table 4: a performance gain of 5 or more at a cost increase of 2 or less.

References

1. F. Lee, "Lookaside Memory Implementation," Project MAC Memorandum MAC-M-99, Aug 19, 1963
2. M. V. Wilkes, "Slave Memories and Dynamic Storage Allocation," *IEEE Transactions on Computers*, 1965, pp 270-271
3. F. Lee, "Study of 'Look-Aside' Memory," *IEEE Transactions on Computers*, Nov 1969, pp 1062-1064
4. R. M. Meade, "Design Approaches for Cache Memory Control," *Computer Design*, Jan 1971, pp 87-93
5. J. Conti, "Concepts for Buffer Storage," *Computer Group News*, Mar 1969, pp 9-13
6. G. G. Scarrott, "The Efficient Use of Multilevel Storage," *Proceedings IFIP Congress*, 1965, Vol 1, p 137