A SURVEY OF MICROPROCESSOR ARCHITECTURES FOR MEMORY MANAGEMENT

B. Furht
Department of Electrical and Computer Engineering
University of Miami, Coral Gables, Florida 33124

V. Milutinović
School of Electrical Engineering
Purdue University, West Lafayette, Indiana 47907

UDK: 681.3.325.6.08

This paper presents an overview of current microprocessor architectures which support memory management. Basic requirements for a processor to support the memory management are defined, and the hierarchically organized memory is introduced. Several address translation schemes, such as paging, segmentation, and combined paging/segmentation are described, and their implementation in current microprocessors is discussed. A special emphasis is given to the application of the associative cache memory. Single-level and multi-level address mapping schemes are analyzed and compared. Furthermore, the paper discusses the capabilities of current microprocessors to support virtual memory, which includes abilities to recognize an address fault, to abort the execution of the current instruction and save necessary information, and the ability to restore the saved state and resume normal processing. Two methods to restart the interrupted instruction, instruction restart and instruction continuation, are evaluated, and their implementation in current microprocessors is discussed. Protection and security requirements are defined, and two protection schemes, hierarchical and non-hierarchical, are evaluated.

I. INTRODUCTION

New generation 16-bit and 32-bit microprocessors are extensively used in multiuser and multitasking environments. Therefore, there is an increased demand for the support of memory management. Furthermore, as shown in Figure 1, the capacity of primary and secondary memories in advanced microprocessors is increasing, which in turn requires an increased virtual memory space, as well as more sophisticated virtual memory management mechanisms.

In the 16-bit microprocessor arena, the techniques applied to solve memory management problems are relatively inadequate, and inefficient. At the 32-bit level, a more standardized approach can be found, and significantly more sophisticated architectures for memory management have been designed. The paper evaluates various architectures for memory management and virtual memory support, and their implementations in existing microprocessors. Several important issues are addressed, such as selection of a virtual memory organization, multi-level memory mapping schemes, associative cache memories applied to address translation, virtual memory support techniques, dynamic memory allocation algorithms, as well as protection and security techniques.

The implementation of these techniques in current 16-bit and 32-bit microprocessors, such as Intel 286, 386, and 432, Motorola 68010 and 68010, National 32032, and Zilog Z80,000, is discussed.

The paper is organized in eight sections. The Section 2 discusses the requirements for a processor to support memory management. Two main strategies applied in current microprocessors are presented: memory management unit (MMU) on-the-CPU chip versus off-the-CPU-chip. Two memory addressing schemes, linear and augmented, are evaluated. Section 3 deals with the various address translation techniques, such as paging, segmentation and combined paging/segmentation, and their implementations in current microprocessors. Both single-level and multi-level address mapping schemes are evaluated. Techniques to support virtual address mechanism are presented in Section 4. The implementations of two methods, which resume operation after an address fault is detected and corrected, are discussed. Section 5 describes the security and protection techniques applied in current microprocessors.

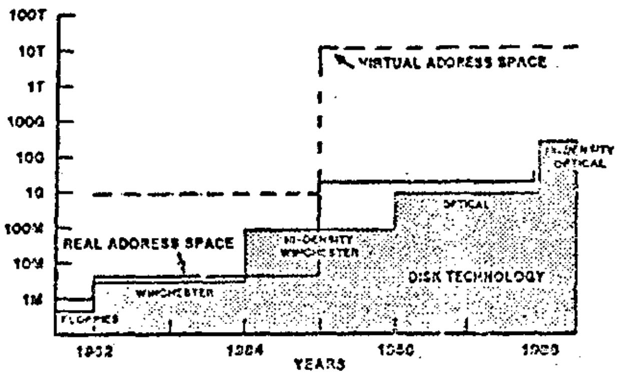

Figure 1. Addressing range needs [54]

2. MEMORY MANAGEMENT REQUIREMENTS

Advanced microprocessor system architecture, which is able to support memory management, uses the hierarchically structured memory system, as shown in Figure 2.

The memory system consists of three levels and involves the maintaining of a large address space based on a hierarchy of memory devices, which differ in memory capacity, speed, and cost. At the first level is the high-speed cache memory, which is the most expensive and, therefore of the lowest capacity. At the second level is the real (primary) memory, which is slower, but less expansive than the cache memory. The

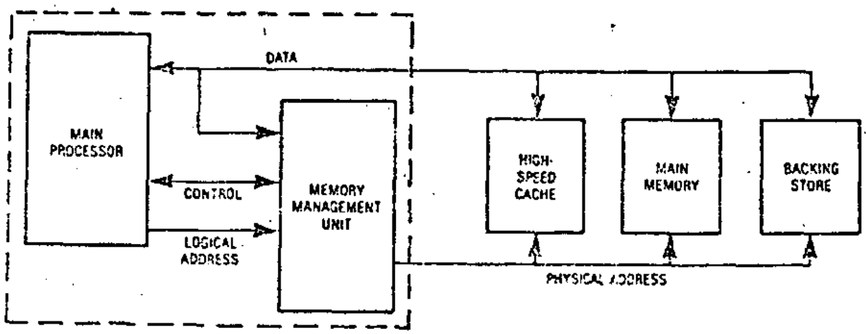

Figure 2. Microprocessor system architecture with three levels of hierarchically organized memory to support memory management [36]

third level consists of large capacity storage devices, such as disks, which hold the programs and data that cannot fit in the first two levels. When a process is to be run, its code and data are brought into primary or cache memory, where cache memory always holds the most recently used code or data.

In this hierarchical memory structure, the basic requirements of the memory management system can be specified as follows:

1. ability to translate addresses and support dynamic memory allocation,
2. ability to support virtual memory, and
3. ability to provide memory protection and security.

There are two basic strategies in creating the microprocessor system architecture for memory management:

1. memory management unit is on the CPU chip, and
2. memory management unit off the CPU chip.

Both strategies, as well as the list of microprocessor systems which apply them, are indicated in Figure 3.

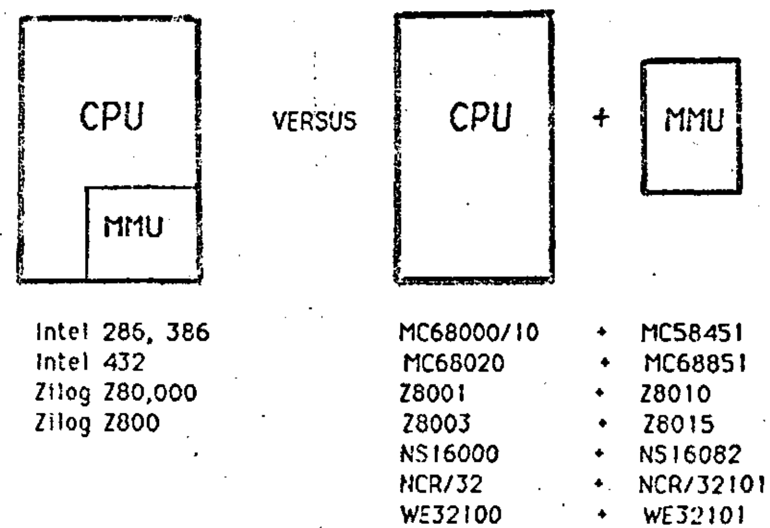

Figure 3. On-chip versus off-chip memory management unit

The main advantages of having the memory management on the CPU chip are:

1. access time improvement, because there is no off-chip MMU-related delays,
2. maximum portability of operating system and application programs, and
3. parts-count reduction.

On the other hand, the memory management on the CPU chip requires additional transistor count, which could be invested into other more frequently used resources. For example, the Motorola 68020, which applies memory management off the CPU chip, uses the saved transistor count to implement the instruction cache on the chip.

Another important issue related to memory management is selection of the memory organization scheme. Basically, there are two types of memory organization schemes: linear and segmented.

In the linear addressing schemes, addresses typically start from zero, and proceed linearly. The memory may later be structured, by software, at the level of address translation.

In the segmented addressing schemes, the programs are not written as a linear sequence of instructions and data, but rather as modules of code and data. The logical address space is broken into several linear address spaces, each of the specified length. An effective logical address is computed as a combination of the segment number, which is a pointer to a block in memory, and the segment offset, which defines the displacement within the segment.

Table 1 shows memory addressing schemes applied in various advanced microprocessors.

Note that Intel and Zilog offer both segmented and linear addressing on their 32-bit processors i80386 and Z80,000, respectively, as software programmable options.

In general, a linear addressing scheme is better suited for the applications that manipulate large data structures, while the segmented addressing scheme facilitates programming, enabling the programmer to structure software into segments. In addition, the segmented addressing scheme simplifies protection and relocation of objects in memory. As an example of the segmented addressing scheme, Intel's i8086 processor contains four 16-bit segment registers, which point to four objects in the memory: code, stack, data, and extra segment (alternate data), as shown in Figure 4a. The address calculation mechanism, which produces 20-bit physical address for the i8086, is shown in Fig. 4b.

3. ADDRESS TRANSLATION TECHNIQUES

Regardless of the memory organization scheme, the processor must have an address translation mechanism to handle virtual memory. The address translation mechanism also provides a method of protecting memory objects.

The address translation is a process of mapping logical to physical memory

TABLE 1
Memory addressing schemes in advanced microprocessors

PROCESSORS	ADDRESSING SCHEME	
	Linear	Segmented
Intel		
8086,80286,432		•
80386	•	•
Motorola		
68000, 68010, 68020	•	
National		
16032, 32032	•	
Zilog		
Z8000 family		•
Z80,000	•	•
AT&T		
WE32100	•	
NCR		
NCR/32	•	

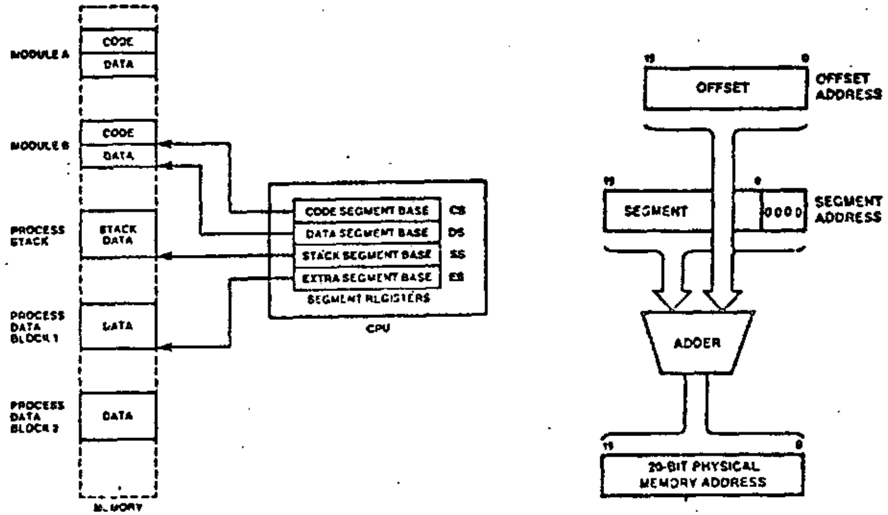

Figure 4 Segmentation and address calculation
a. Segmented addressing scheme of the i8086 [28]
b. Address calculation in the i8085 [28]

addresses. The address translation mechanism divides the memory into blocks, and then performs mapping of a block of logical addresses into a block of physical memory addresses. It allows programs to be relocated in the primary memory. It also provides the base for virtual memory system design, where the logical address space can be larger than the physical address space. The virtual memory mechanism allows programs to execute even when only few blocks of a program are in the primary memory, while the rest of the program is in the secondary memory (on the disk). The other important processor requirements for virtual memory support are discussed in Section 4. Three basic address translation schemes are:

1. paging
2. segmentation, and
3. combined paging/segmentation.

In the paging systems, the primary memory is divided into fixed-size blocks (pages), while in the segmentation systems, the blocks are of various size (segments), as shown in Figure 5.

Generally, the segments can overlap, while pages cannot. so pages are usually of a relatively small size, compared to total memory. Typical page size is between 256 and 2048 bytes, while segments can be 64K bytes or more.

The paging/segmentation systems combine the features of both paging and segmentation addressing schemes. The segmentation part of the scheme manages virtual space by dividing the programs into segments, while the paging part manages physical memory, which is divided into pages. Each segment consists of a number of pages, as shown in Figure 6.

Selection of the address translation mechanism has a crucial impact on the memory management techniques, which have to be implemented by the operating system, to handle page or segment fetching, placement, and replacement. For example, the paging address translation system is well suited for page placement and replacement, because all pages are of uniform size, while the segmentation system needs more complicated placement and replacement algorithms to match incoming segments with available memory space in the segmentation systems, a segment must reside entirely in physical (primary) memory in order to be executed, because the minimum unit that can be swapped is the segment itself. The available memory space becomes then fragmented into many small pieces, and there is not enough contiguous memory for storing one large segment. Because of the fragmentation problem associated with the segmentation systems, the paging systems are more efficient with respect to memory utilization. In the paging systems, all pages are of equal size, thus pages can be swapped without leaving unusable fragmented spaces. Also, it is not necessary to swap in all pages of a program at once, in order to execute it, but only the pages required ("demand paging"). This significantly reduces the swapping time.

For all these reasons, the demand paging address translation system seems to be

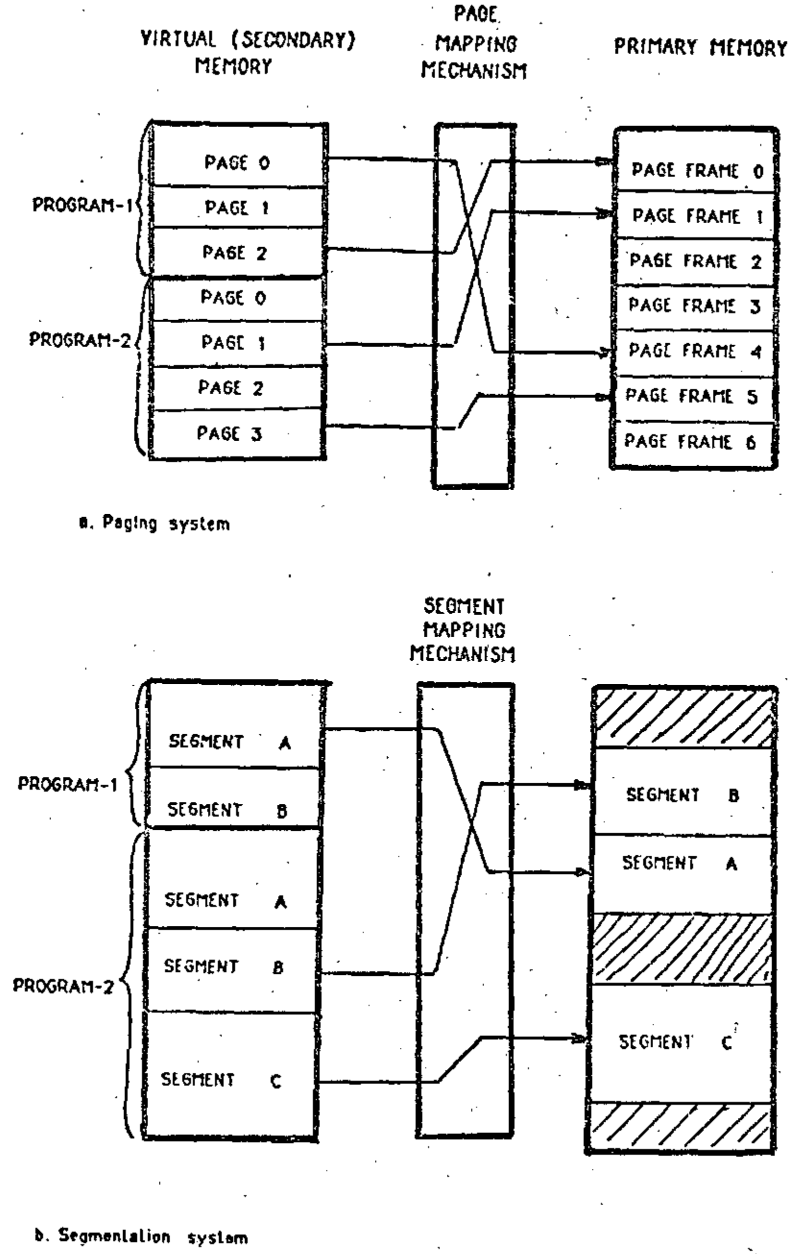

Figure 5. Address translation schemes
a. paging
b. segmentation

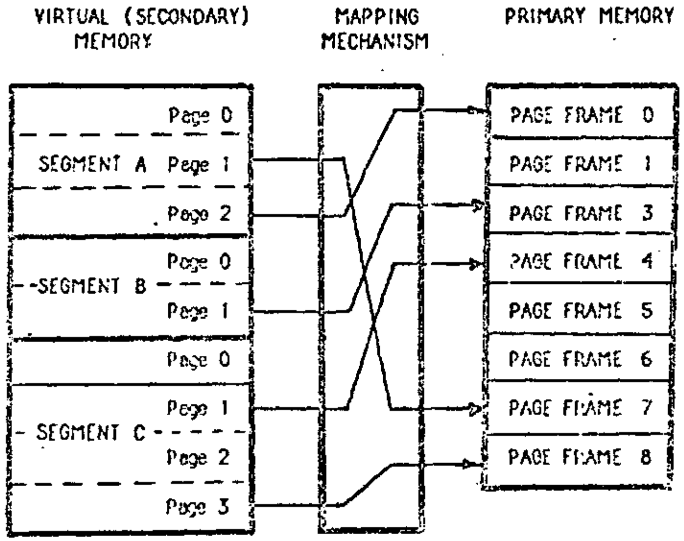

Figure 6 Address translation by combined paging and segmentation (paging/segmentation)

the way to go. As a matter of fact, all advanced 32-bit processors, as well as several 16-bit processors, fully support demand paging technique, which may become a standard address translation mechanism in future microprocessors.

Furthermore, when one selects the address translation scheme (paging, segmentation, or combined system), there are two additional issues which should be addressed:

1. implementation of the selected address translation mechanism, and
2. selection of the number of mapping levels

3.1 Implementation of the address translation schemes

Regardless of the address translation organization, the implementation method is always based on translation tables located in primary memory: page map tables (PMT) in the paging systems, and segment map tables (SMT) in the segmentation systems [10,11,14,16]. The table entries contain information to translate the logical into the physical address, as well as additional data for protection purposes, and to support placement and replacement algorithms. A typical format of a translation table entry is shown in Figure 7.

As an example of the address translation implementation, the virtual address of the i286 processor consists of a pair: segment selector and displacement v=(s,d). The

RESIDENCE BIT	ACCESS RIGHTS & PROTECTION	SUPPORT FOR REPLACEMENT	PHYSICAL ADDRESS

Figure 7. Typical format of a page or segment table entry

segment selector points to the segment descriptor in the segment map table, as shown in Figure 8. The segment descriptor contains the primary memory address s', at which the segment begins. The displacement d is added to s' forming the real physical address, r=d+s', corresponding to the virtual address v.

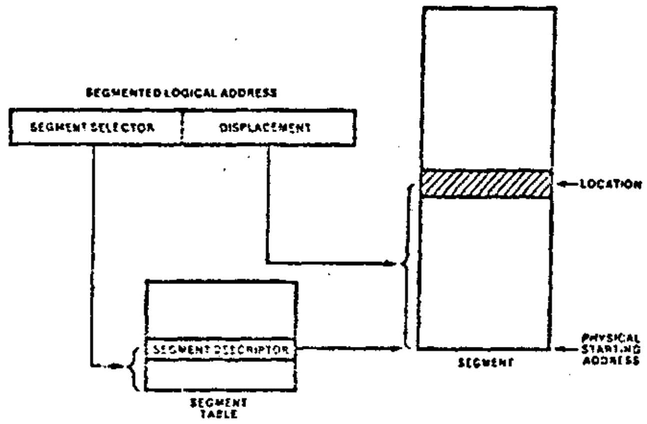

Figure 8. Address translation mechanism of the i286 [28]

The described address translation implementation method is known as direct mapping. Translating a logical address to a physical address, using direct mapping, requires an additional memory access operation to obtain segment (or page) base address, and therefore the use of direct mapping can cause the computer system to run programs at lower speed. There are several solutions applied in modern microprocessor architectures to overcome this problem. These solutions are discussed below.

In the Intel's i286 processor standard, four segment registers are extended with the corresponding four 48-bit segment descriptor cache registers, as shown in Figure 9 [26,28].

Segment registers are loaded by the program, while the CPU loads the explicit cache registers, which are invisible to programs. Explicit cache speeds up the operation by eliminating the need to refer to a descriptor table for every memory reference instruction. Loading the explicit cache is performed in four steps:

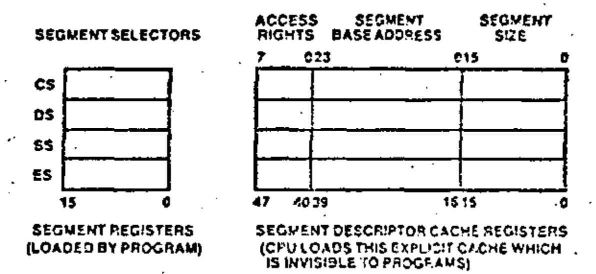

Figure 9. Descriptor data type in the i286 [28]

1. Program places a selector in the corresponding segment register.
2. Processor adds the selector index to the base address of the descriptor table, to select a descriptor.
3. After the processor verifies segment access rights, it copies the descriptor to the data segment register in cache.
4. The processor uses the descriptor information to check segment types and limits, as well as to form the effective address.

The described technique based on explicit cache registers speeds up the direct mapping, but still is not efficient enough, because it requires cache loading whenever control is transferred from one to another segment of the same type.

A much more sophisticated solution is based on a special associative cache (32 to 64 locations), which holds the most recently used set of translation values. Then, the translation process is performed in the following steps, as shown in Figure 10:

1. First, the virtual address received from the CPU is searched through the cache. If the address matches with one of the cache entries, then the corresponding physical address stored in the cache is used by the CPU to access the primary memory directly.
2. If the received virtual address does not match with cache entries, but the page or segment is in the primary memory, then the physical address will be fetched from the translation tables located in the primary memory, and then stored in the cache. Then, the CPU will access the required physical memory.
3. Finally, if the page or segment is not in the primary memory, it must be first swapped from the secondary memory into the primary memory. The translation tables must be updated, and the physical memory address will be fetched from the translation tables into the cache.

The associative cache memory is usually organized as a Translation Lookaside Buffer (TLB). When the address translation mechanism receives a logical address, every entry in the TLB is searched simultaneously for the logical address.

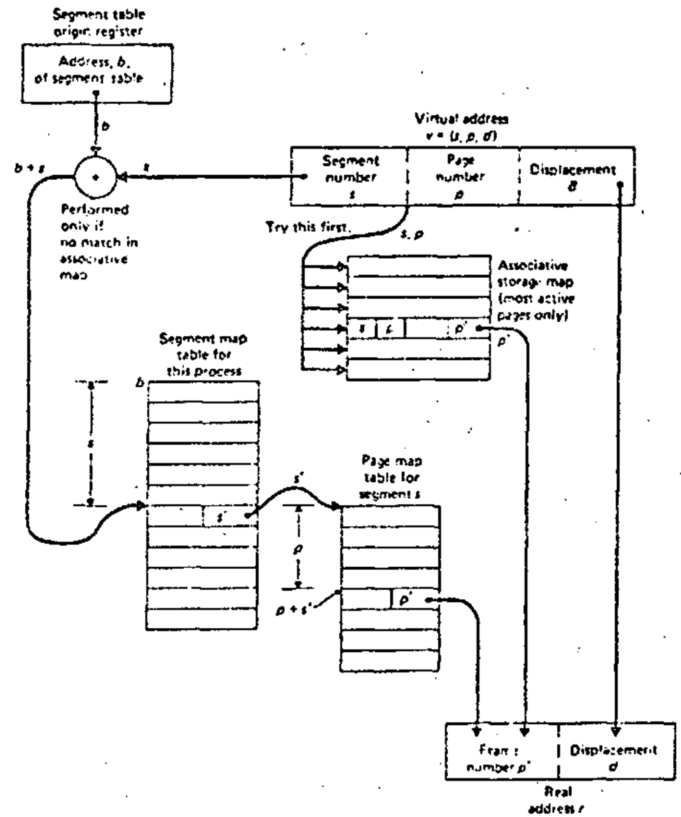

Figure 10. Address translation mechanism using the associative cache memory [10]

A number of simulation studies have proven that the small associative cache significantly speeds up the system operation, because the hit ratio of finding the address in the cache riches 98%. Many recent processors, (such as Motorola 68000 family with its MC68451 MMU, Intel 80386, Zilog Z8000 family and Z80,000, National NS16000 family with its NS16082 MMU, and others) have implemented the address translation scheme by using the TLB method [5,33,34,35,37,50,51,54]. Although translation mechanisms based on the TLB method vary in complexity, they can be classified in two basic groups: address-accessable TLBs, and content-addressable TLBs. In the address-accessable TLB approach, a logical address field identifies the register in the TLB that holds the physical base address. As an example of this technique, the Z800 with on-chip MMU is shown in Figure 11 [33].

The virtual address consists of a 4-bit TLB pointer, and a 12-bit offset. The TLB pointer selects one of the 16 translation registers of the TLB. Then, the 24-bit physical

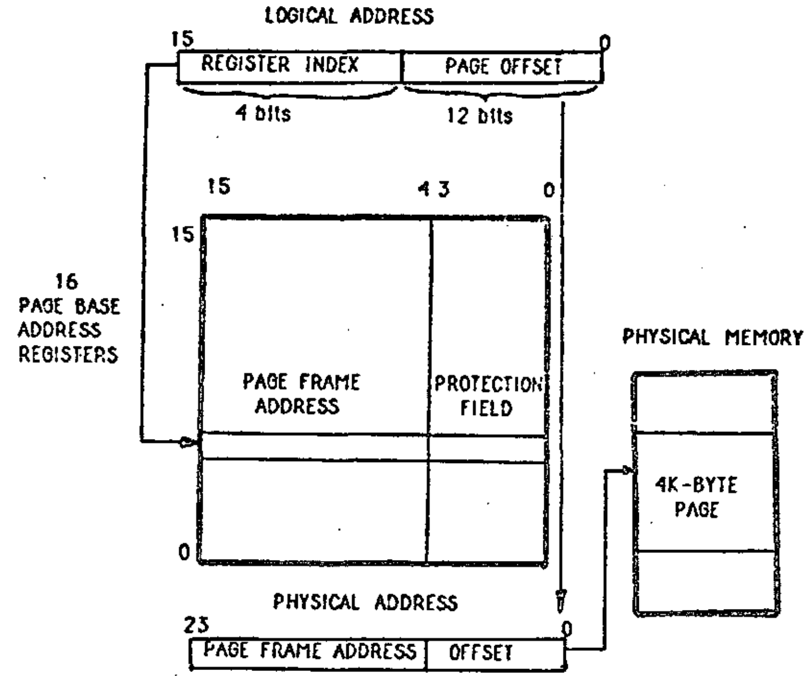

Figure 11. The Z800 address translation based on the address-accessable TLB [33]

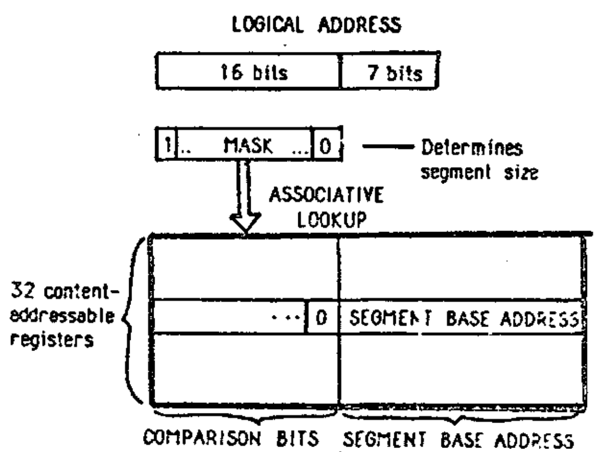

Figure 12. Content-addressable TLB in the MC58451 MMU [44]

address is formed, as a selected 12-bit page base address from the TLB, concatenated with the 12-bit offset.

The address-accessable TLB technique is not practical for large systems, because accessing the TLB by addresses requires a segment register for each logical segment or page that can be relocated.

The content-addressable TLB is more suitable for large systems. This method has been applied in several microprocessors, such as the MC68451 MMU, Z8015 PMMU, Intel 80386, and NS 16082 MMU. To illustrate this method, Figure 12 shows the content-addressable TLB applied in the MC68451 MMU.

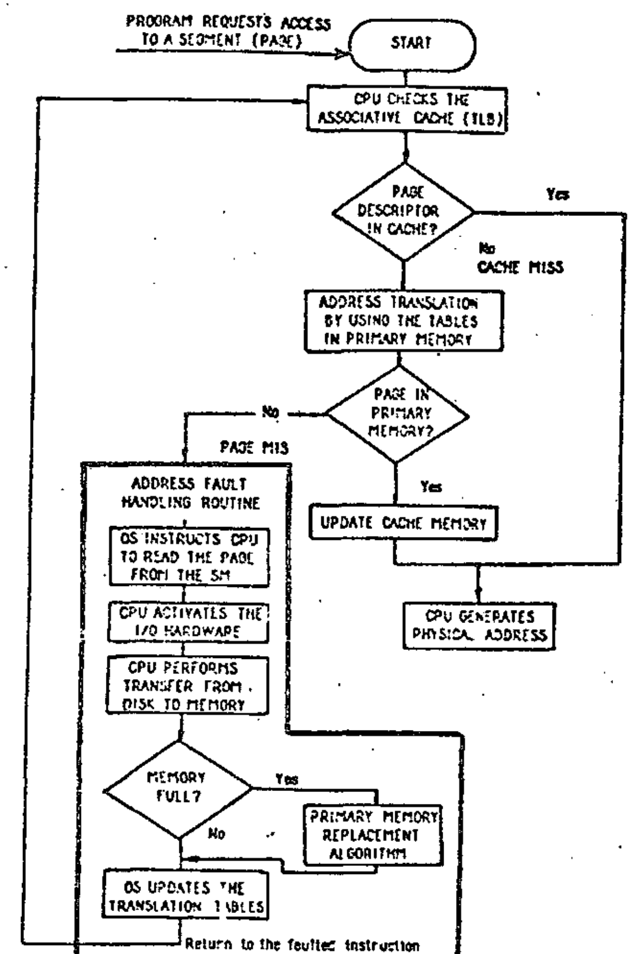

Figure 13. The flow-chart of a paging virtual memory system with an associative cache memory

The MMU receives a logical address (23 bits), and the mask register masks the low order bits to determine the segment size. Then, the MMU compares the rest of the most significant bits with the comparison field values of 32 content-addressable registers. If a match is found, the MMU performs address translation. If there is no match, the MMU generates a fault condition, and activates a trap routine. The trap routine will update the TLB from translation tables stored in primary memory.

The flow-chart in Figure 13 illustrates the necessary operations in a paging-based virtual memory system with an associative cache memory for recently used pages.

The virtual memory is activated whenever program requests an access to a page. The flow-chart in Figure 13 indicates three different control paths:

1. when the page descriptor is found in the associative cache,
2. when the page descriptor is not found in the cache ('cache miss'), but the page is in the primary memory, and
3. when the page descriptor is not found in the associative cache, and the page is not in the primary memory ('page miss'). Then, the address fault handling routine is activated.

In addition to address translation mechanism, the MC68451 MMU supports dynamic memory allocation. The dynamic memory allocation mechanism is able to allocate the memory to a process, while it is running. The Binary Buddy system, an algorithm for dynamic memory allocation, is implemented in the MC68451. The algorithm divides the entire physical address space into buffers, the size of which varies from 256 bytes to 256K bytes (in the MC68451). The algorithm maintains these buffers by using the buffer lists for all sets of buffers of the same size, as well as buffer descriptors for each buffer independently [52].

When a memory request is received, the algorithm searches through the list of available buffers in order to find the best fitted buffer. If the best-fitted buffer is not available, the search process is continued for the next larger size buffer. The flow-chart of the Binary Buddy algorithm is shown in Figure 14.

A detailed description of the algorithm, as well as additional issues related to it, are discussed in [45,48,52].

3.2 Single-level versus multi-level address mapping

The second issue closely related to address translation architectures is dealing with the number of mapping levels in address translation schemes. The conventional address mapping scheme consists of just one mapping level, such as in most of the 16-bit processors (i286, Z8010, and Z8015 MMUs). On the other hand, almost all 32-bit processors use multi-level mapping schemes, which brings some new features in the memory management.

The basic advantages of multi-level mapping schemes versus single-level mapping schemes can be summarized as follows:

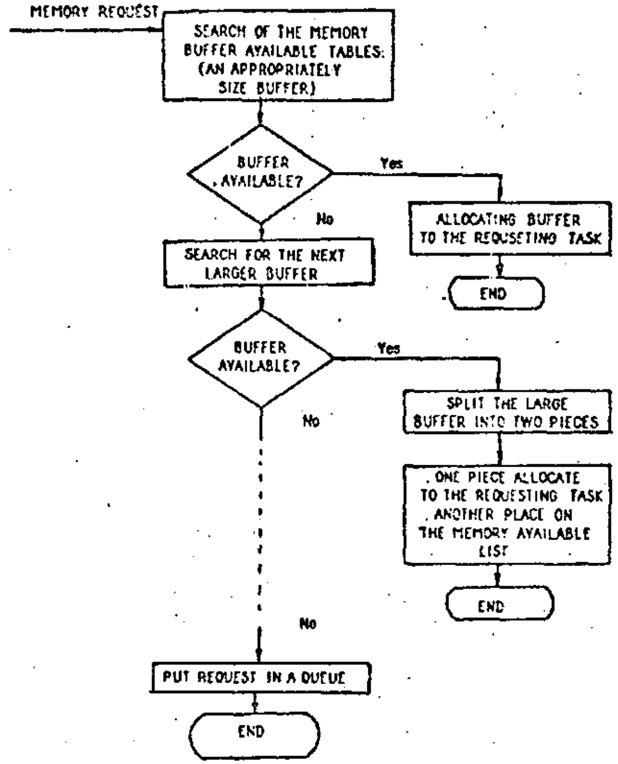

Figure 14. Binary Buddy algorithm for dynamic memory allocation

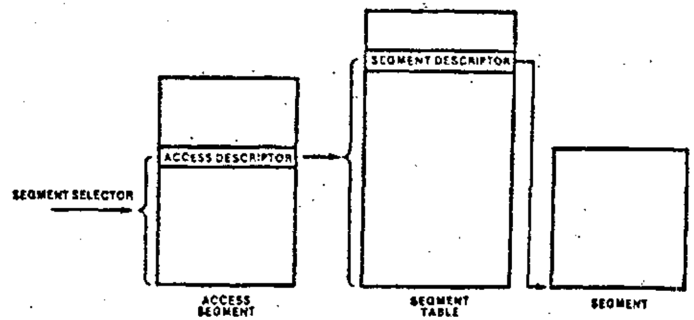

Figure 15. Two-level address mapping scheme in the i432 processor [27]

1. they provide more sophisticated protection mechanism,
2. they are able to accommodate larger address space, and
3. they provide page sharing.

Several multi-level mapping schemes are evaluated below.

Intel's i432 processor uses two-level mapping in order to provide more sophisticated protection mechanism, as shown in Figure 15 [27,40,47].

The segment selector register points to an entry of the access segment, where the access rights are stored and are thus associated with program modules. The access descriptor contains the pointer to the segment table, and finally the segment descriptor contains a pointer to the beginning of the selected segment in the primary memory. Because the access rights are stored independently of the segment descriptors, several modules can share the same segment, each with different access rights to it. In Figure 15, the module A can write and read the selected segment, while the module B can only read the segment.

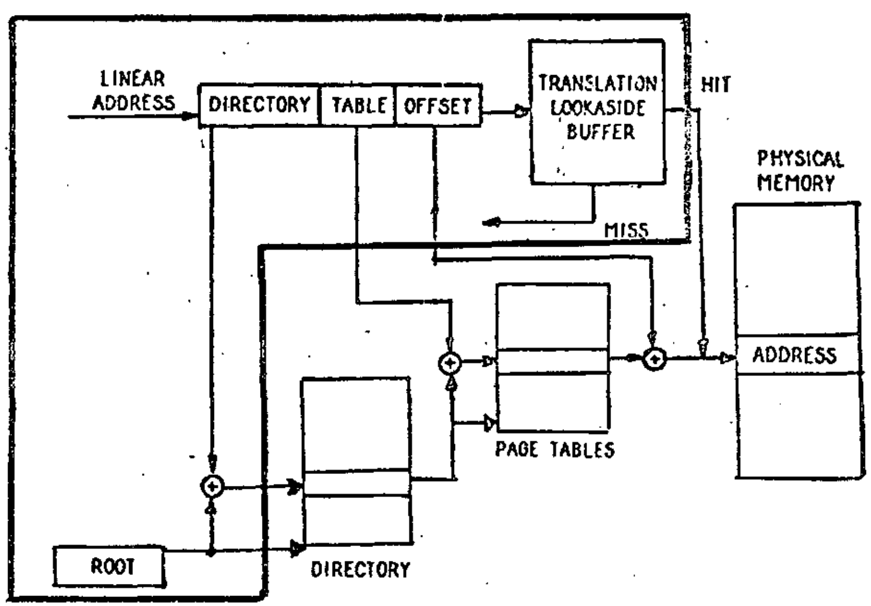

Figure 16. Paging system architecture in the i386 processor [54]

In addition, the two-level mapping scheme makes it possible to restrict the number of segments accessable by a given program. In single-level mapping systems, such as i286, any program may address any segment in memory, simply by pointing to it through the segment table.

The two-level scheme of the i432 also enables fewer address bits to point to a particular segment.

The Intel's i386 provides two options, which are user selectable: segmentation system (same as in the i286), or paging system. The paging system architecture uses two-level mapping scheme, along with a translation lookaside buffer, designed as a cache memory. The complete architecture is shown in Figure 16.

The linear virtual address consists of three fields (directory, table, offset), and address translation is performed in the following steps:

1. first, the address is searched through the TLB. If the address is found, the translation is performed in the TLB, and the primary memory is accessed directly.
2. if the address is not found in the TLB, the miss signal is generated, and the translation is performed through the two-level mapping built on the CPU chip, as shown in Figure 15.

The two-level on-chip mapping scheme enables fast address translation, and page tables can be shared and/or swapped.

A similar two-level mapping scheme has been implemented in the NS16082 MMU [6,25,38]. The total physical address space is divided into 32,768 fixed pages of 512 bytes each. The virtual address consist of 24 bits divided into three fields: index-1 and index-2 of the page selector, and the offset, as shown in Figure 17.

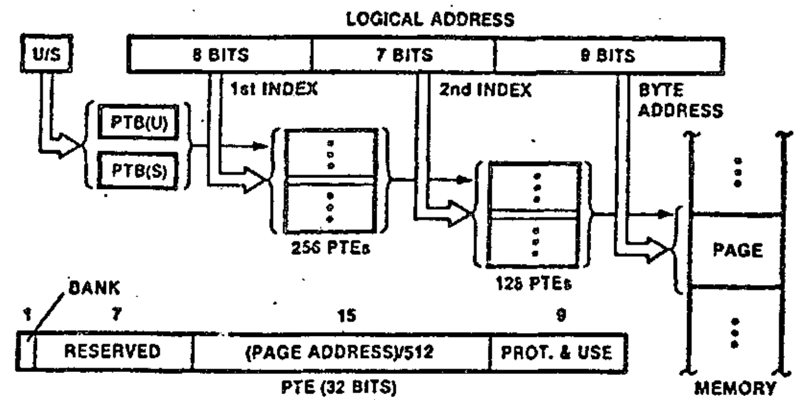

Figure 17. Two-level mapping scheme of the NS16082 MMU [38]

The index-1 (8 bits) of the page selector is used to locate one of the 256 entries of the page table. The contents of the page table PTE-1 points to the beginning of one of 256 pointer tables, each of which contains 128 entries. Then. the pointer to the pointer table is combined with the index-2 (7 bits) of the page selector, to locate one of the entries within the pointer table. The selected entry contains the actual page number in primary memory. The offset field is then used to locate data within the page. The NS 16082 MMU contains the associative cache to hold 32 recently used page address entries, as well.

The Z80,000 processor uses three-level mapping scheme based on the set of three translation tables located in primary memory [2,33,53]. It also contains an associative memory for the TLB, where 16 most recently referenced pages are stored. The CPU automatically loads the TLB from translation tables, when a logical address is missing.

The NCR/32 processor uses an address translation chip (ATC) for address translation based on paging system with one-level mapping [22]. The chip contains 16 associative memories for recently used pages.

The Z8010 MMU, which is used with the Z8001 processor, applies one-level segmentation system, based on 64 content-addressable segment descriptor registers. For more details see [33,56].

The Z8015 MMU differs from the Z8010 MMU in that the logical address is translated into page frames rather than segments. It applies one-level mapping scheme and uses 64 page descriptor registers, which are also content-addressable [33,56].

The WE32100 32-bit processor uses off-chip 32101 MMU, which supports both demand paging and segmentation systems, which are user selectable [15, 17, 18]. The MMU contains an on-chip cache memory: a 32-entry segment descriptor cache, and a 64-entry page descriptor cache, to hold recently used segment and page descriptors, respectively.

Table 2 summarizes address translation features of some 16- and 32-bit microprocessors.

4. VIRTUAL ADDRESS SUPPORT TECHNIQUES

A virtual memory system allows the user to execute programs on a very large memory of virtual address space, much larger than the actual physical memory. This is accomplished by the capability of a microprocessor to detect access to memory pages (or segments) which are not present in the physical memory. When the virtual memory system detects such a reference, it will fetch the required page from the secondary memory into the primary memory.

In order to support virtual memory capabilities, besides the address translation, a microprocessor must provide the following attributes:

1. to recognize a page or segment fault, if the page or segment is not present in the primary memory. The memory manager must then inform the processor so that the missing page or segment can be fetched from the secondary memory, and eventually one of the current pages or segments can be

TABLE 2
Address translation charateristics of advanced microprocessors

PROCESSOR	ADDRESS SPACE REAL	ADDRESS SPACE VIRTUAL	ADDRESS TRANSLATION SCHEME	MAPPING LEVELS	ASSOCIATIVE CACHE
Intel 80286	16M	1G	SEGMENTATION	1	4 segment descr.reg.
Intel 432	16M	1T	SEGMENTATION	2	Assoc. cache TLB
Intel 80386	4G	64T	PAGING&SEGMENTATION	2	Assoc. cache TLB
MC68000	16M	4G	USER SELECTABLE	1	32 content-addr. reg.
MC68010 + 68451 MMU	16M	4G	USER SELECTABLE	1	32-content-addr. reg.
MC68020 + 68851 MMU	4G	?	USER SELECTABLE	1	32-content addr. reg.
Z8001 + Z8010 MMU	16M	4G	SEGMENTATION	1	64 segm. content-addr.reg.
Z8003 + Z8015 MMU	8M	4G	PAGING (page size=2K)	1	64 page descr. registers
Z800	16M	4G	PAGING (page size=4K)	1	16 addr.-accessable reg.
Z80,000	16M	4G	PAGING (page size=1K)	3	Assoc. cache TLB
NS16032 NS32032 + 16082 MMU	16M	4G	PAGING (page size=512)	2	32 content-addr. cache
NCR/32 + NCR32101	16M	4G	PAGING (page size=1K)	1	16 associative memories
WE32100 + WE32101 MMU	4G	?	PAGING&SEGMENTATION	1	64-entry page descr. table 32-entry segm.descr.table

replaced,

2. to abort execution of the current instruction *(instruction abort capability)*,
3. to save necessary information needed later to recover from the fault,
4. to call and execute the fault service routine in the operating system, which will swap the required page(s) or segments(s), from secondary memory to primary memory,
5. to provide necessary information for the operating system, in order to support page (or segment) placement and replacement algorithms *(indication of access activities)*, and
6. to restore the saved state and resume the normal processing *(instruction restart capabilities)*.

Although very different in complexity, all advanced microprocessors provide instruction abort and restart capabilities. Some solutions are presented below. Recognizing the access fault can be performed internally only, if the MMU is on the CPU chip, or both internally and externally, if the MMU is off the CPU chip.

When an access is made to an instruction or data which is not present in primary memory, an address error is internally detected, and it initiates the address error fault handling routine *(internally detected fault)*. If the off-chip MMU detects a fault situation, it will send a signal to the CPU, which will in turn activate the fault handling routine *(externally detected fault)*.

When the CPU recognizes an access fault, it saves the state information needed to recover from the fault. The information is usually saved on the stack. The typical information which must be saved in the program counter (starting address of the instruction), the status register, the fault address, the trap-specific parameters, the access type, the internal temporary registers, various internal statuses, etc. For illustration, the MC68010 processor which supports virtual memory, saves 26 words versus the MC68000 process, which saves only seven words, which is not enough to provide the user with the state of the machine after the fault has been occurred. Figure 18 shows the information saved on the stack for these two processors.

The MC 68010 address stack is divided into two parts: a user visible section, and a non-user visible section in which the internal status and the temporary data are saved.

The memory management unit also has to provide the information related to access activities needed by the operating system (placement and replacement algorithms). This information is usually stored in the transition table entries. There are three information bits which are present in typical systems:

1. *the valid bit* - which is controlled by the operating system, and specifies whether or not a block (page or segment) is in the primary memory.
2. *the references bit* - where the MMU typically sets this bit to indicate if access to the corresponding block in primary memory is on. The operating system may reset this bit to keep track of the access history.
3. *the modified bit* - which is set by any write operation to the corresponding

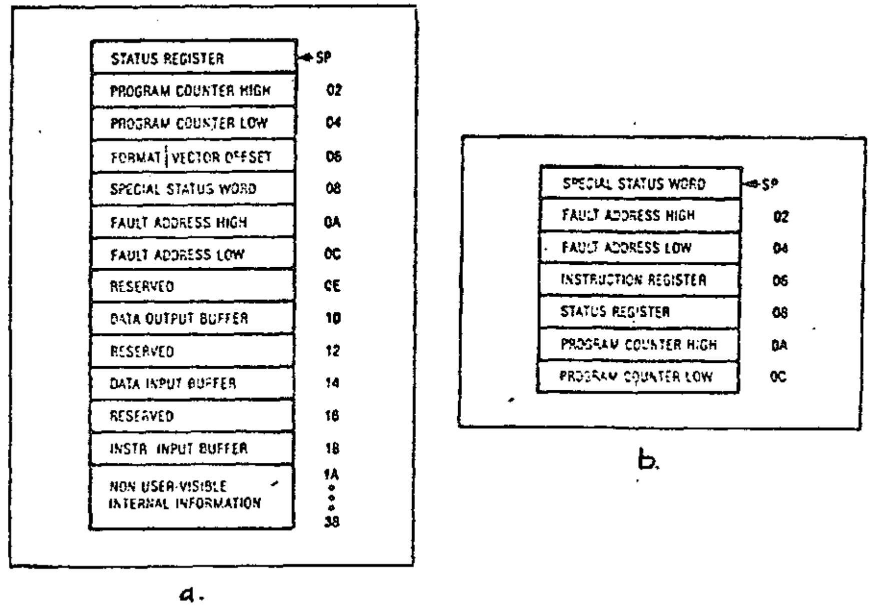

Figure 18. Address error stack [36]
a. MC68010
b. MC68000

block. This bit indicates whether the block must be written back to the secondary memory, before being replaced from the primary memory.

For illustration, the i432 processor contains four access activity bits in its segment descriptor, as shown in Figure 19.

The valid bit (V) indicates whether or not the segment is in the memory. The storage allocated bit (S) indicates whether any memory has been associated with this descriptor. The accessed bit (AC) indicates whether the segment has been accessed, while the altered bit (AL) indicates whether the information contained in the segment has been modified.

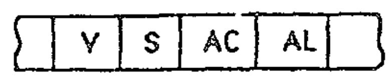

Figure 19. Access activity bits of the i432 processor contained in the segment descriptor [27]

The operating system uses V and S bits to detect when a physical segment is not present in memory, while the AC and AL bits are used by the replacement algorithm to decide which of the currently present segments should be swapped out by the new segment.

In addition, several fields in the segment descriptor can be used by the operating system to record other useful information about the segment (frequency of use, etc.).

The other advanced processors contain similar information on access activities used by the operating system. Commonly used page replacement techniques are Least Recently Used (LRU), Least Frequently Used (LFU), and First-In-First-Out (FIFO) [1,5,8,9,55]. The described information maintained by the CPU (referenced and modified bits), as well as some additional user-defined fields, can be used to design the page replacement algorithm in the operating system.

One of the popular schemes for the LRU algorithm classifies the pages into four groups:

Group 1: unreferenced (R =0) and unmodified (M=0)
Group 2: unreferenced (R =0) and modified (M=1)
Group 3: referenced (R=1) and unmodified (M=0)
Group 4: referenced (R=1) and modified (M=1)

The pages from the lowest groups are replaced first, and the pages from the highest groups are replaced last. The referenced bit is set by the CPU whenever the page is referenced. The operating system (OS) periodically clears the referenced bit. A sophisticated LRU algorithm, "software caching," has been implemented in the VAX/VMS operating system [31]. The LFU algorithm can also be incorporated into this scheme. Whenever the referenced bit is cleared, the OS can count the frequency with which the pages were used. The modified bit is set by the CPU whenever the page is written. When the page is swapped, the OS checks this bit to see if there is a need to update the copy of the page in the secondary memory.

The last attribute of a processor to support virtual memory is the most complex, and refers to reloading of the state of the program, and resuming the operation, after the address fault routine is completed. Two methods of implementing the resume operation on a processor are:

1. instruction restart method, and
2. instruction continuation method.

Advantages and drawbacks of these two methods are discussed in the following two subsections.

4.1 Instruction restart method

In this method, after the address fault error handling routine has completed all activities, the instruction in which fault occurred is restarted from the beginning. Figure 20 illustrates the execution of the microcode in the case when no address fault is present (Fig. 20a), and in the case when the restart method is applied, with an address fault occurred (Fig. 20b).

In Figure 20 it is assumed that a machine instruction consists of several microinstructions [m1, m2, m3, m4]. If there is no address fault, these instructions will execute sequentially, as shown in Fig. 20a. If the MMU detects an address fault in the microinstruction m2, the control will be transferred to the address error routine. The address error routine will first save the information state, and then the routine will handle the address error (the required page or segment will be fetched from the secondary memory). Finally, the saved information state will be restored, and the faulted instruction will be restarted from the beginning - at the machine instruction level. Therefore, the sequence [m1, m2, m3, m4] will be executed again.

The main problem in the instruction restart method is that the processor must reconstruct the state of the machine, as it was at the beginning of the machine instruction, while he faulted instruction was interrupted in the middle of its execution. There are some situations when this is very complex, such as when a resource is used both as input and output parameter in the same instruction. For example, in extended precision arithmetic operations, a carry (or borrow) bit from the previous operation is used in the instruction as an input parameter, but the instruction itself also sets the same bit as the result of the current operation. If the address fault is detected after this bit is updated, the original value must be restored before the instruction is restarted. A similar case is with autoincrement and autodecrement addressing modes.

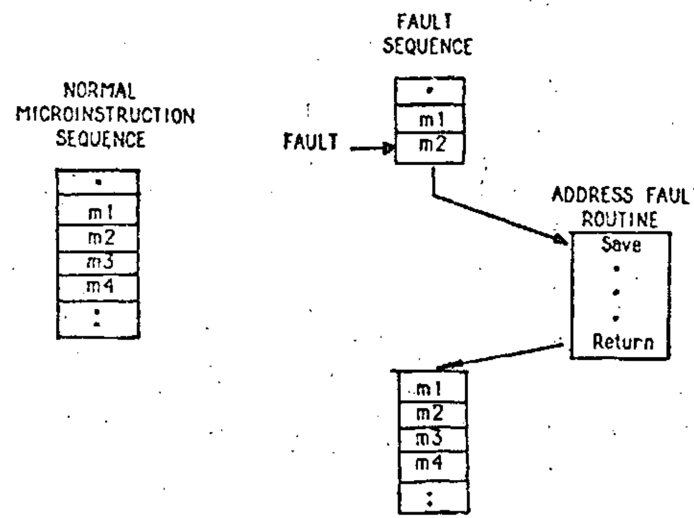

Figure 20. Microinstruction sequence [36]
a. No address fault
b. Instruction restart method

Several techniques have been proposed to solve this problem, and are discussed below:

1. The processor may postpone the modification of user-visible resources (such as carry bit), until the end of the instruction. Then, if the address fault has not occurred, the resources will be updated.
2. All modifications of the user-visible resources will be recorded by the processor if the address fault occurs. On the basis of this information, the processor will be able to restore the original values of the modified resources.
3. The processor maintains the copies of all user-visible resources, that are modified. Because the copy always contains the original value, if the address fault occurs, it will be easy to restore the original state.

4.2. Instruction continuation method

In the instruction continuation method, when the address error routine has been completed, the machine instruction will not be resumed from the beginning, but from the same location within the instruction at which the execution was suspended. The execution of the same sequence of microinstructions (m1, m2, m3, m4), in the case of the continuation method, is shown in Figure 21.

The address fault was detected in the microinstruction m2, and the control was transferred to the address error handling routine. After the routine has been completed, the processor will resume operation, by executing the microinstruction m3. The

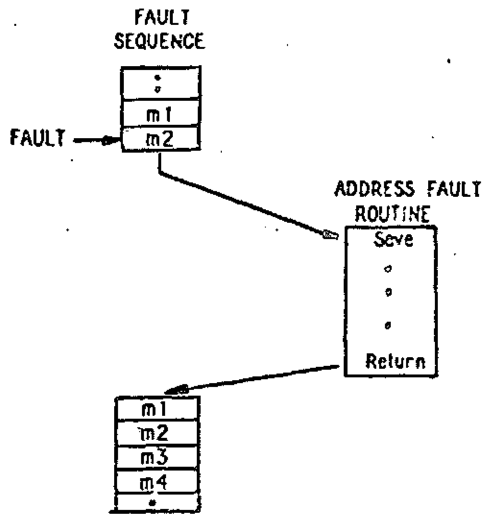

Figure 21. Microinstruction execution - the instruction continuation method [36]

continuation method is analogous to an interrupt operation at the microinstruction level.

In order to support the instruction continuation method, the processor must be able to save the entire state of the machine, when an address fault is detected. Therefore, the processors which apply this method usually have a large address error stack, to save all necessary information (e.g. MC68010). Regardless of this requirement, another problem with the continuation method is related to the instructions that require execution without interruption. In addition, this method requires the additional time and silicon resources for saving and restoring the complete state of the machine.

The instruction continuation method has been implemented in the MC68010 and the MC68020 processors only [35,36], while all other advanced processors use the instruction restart method.

The NS16082 MMU sends an abort signal to the CPU (NS 16032 or 32032), which will stop the execution and will return the CPU into the state before the aborted instruction. Then, all needed information (contained in program counter, machine status, stack pointer, and several other registers) is automatically saved. When the address fault routine is completed, a return-from-trap instruction is executed, which will resume the aborted instruction from the beginning [38].

Zilog processors also implement the instruction restart method. The Z8001/Z8015 system contains a special data count register which counts the number of successful data accesses before an address fault. This information is used to restore the machine state, which existed before the address fault.

The Z80,000 and Z800 processors, which have the MMU on the CPU chip, apply an improved instruction restart method compatible with their pipelining architecture. The Z80,000 executes instructions by using six-stage pipelining, and therefore the page fault can be detected before memory access. The address translation is performed in the third stage of the pipeline, and if an address fault is detected, the execution stage will be suspended, before any change of register contents is made [33,36]. The Z800 applies a similar technique, because it has a three stage pipeline allowing the instruction suspension, before any register is changed.

Intel processors i286 and i386 apply the instruction restart method, as well [26,28,54]. They are also able to detect an address fault before executing instruction, and thus faulted instruction restart becomes simple. After completing the execution of the address fault handling routine, the CPU places the address of the interrupted instruction into the instruction pointer, and resumes the program execution.

5. PROTECTION AND SECURITY TECHNIQUES

In multitasking and multiuser environments, it is required from processor architecture to support protection and security, in order to increase system performance and simplify system implementation. Basically, protection and security issues can be divided into the following topics:

1. memory protection,

2. program protection,
3. user protection, and
4. information security.

Memory protection mechanism should detect any addressing error before it caused damage. Each instruction should be checked to verify that it performs the intended operation. The MMU unit performs this check, and if there is an address error detected, it generates an address fault. The address fault handling routine is then activated, which analyzes the address error, eventually fixes it, and returns to the interrupted program. *Program protection mechanism* should prevent application program from making illegal modifications of the operating system. It also should control the transfer between system modules to achieve total reliability. *User protection mechanism* should protect users against each other. *Security mechanism* should provide limited access to information.

Two basic architectures that provide program and user protection are:

1. hierarchical protection system, or *ring protection system*, and
2. non-hierarchical protection system, or *capability-based protection system*.

These two systems are discussed in the following paragraphs.

Hierarchical protection system consists of a hierarchy of protection levels, or rings, starting from the most privileged to the least privileged. Basic principles of the ring system are:

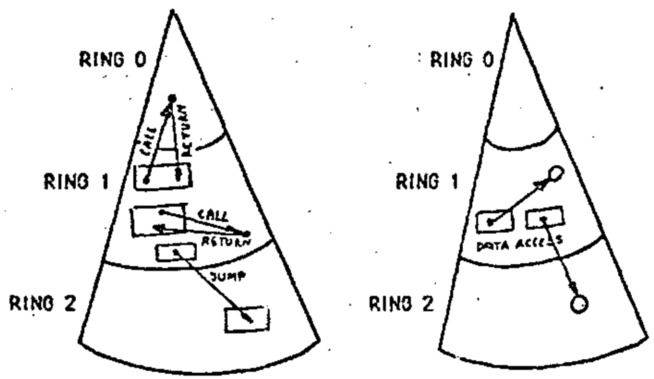

Figure 22. Principles of ring protection system [28]
a. control transfer between programs
b. data access

1. A program may access only data that reside on the same ring, or a less privileged ring,
2. A program may cal services that reside on the same, or a more privileged ring.

These two protection approaches are illustrated in Figure 22.

The ring system has been implemented in the i286 and the i386 processors [21,26,28,54]. Their ring protection system consists of four privilege rings, as shown in Figure 23.

Different priorities are assigned to different programs (segments) within the system. Greater privilege is assigned to more important programs. Typically, the operating system occupies the most-privileged ring, thus it is protected from the application programs. The programs may access the OS with a high-speed call instruction, rather than using the context switching technique, which is the traditional way to implement the call of OS services.

Second and third rings are typically used for system services and custom extensions, respectively, while the application programs are usually located at the least-privileged ring.

The i286/i386 protection model also provides task isolation, by having separate descriptor tables. The entire isolation between rings is provided by a separate stack for each ring.

In non-hierarchical protection systems (or capability-based protection systems), for each task a table of operations is defined. This table of operations specifies operations that may affect other tasks in the system. In order to perform an operation

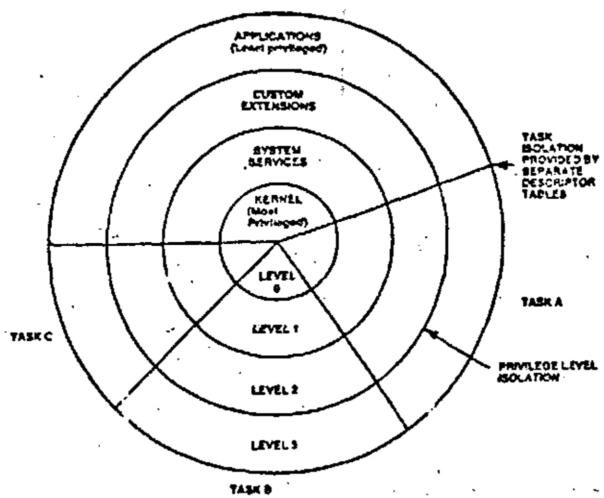

Figure 23. The ring protection system of the i286/i386 processors [28]

which could affect another task, a task must have the corresponding capability in its table of operations.

The capability-based protection system is more complex, and the current processors still do not implement it in the architecture, but in the operating system. The current processors provide some protection features, which can be used when designing a sophisticated protection system in software [3,4,5,25,35].

The MC68000, the Z8000, and the NS 16000 processors have two operating modes (or privilege levels) of the CPU: supervisor mode, and user mode. In the supervisor mode, the CPU can execute the complete set of instructions, while in the user mode, only a subset of instructions can be used. In Zilog processors, these two modes are called system and normal.

Typically, the operating system functions are placed at the supervisor level, while application programs execute at the user level, thus the operating system is protected from the application programs. The supervisor level typically has access to all of the processor resources, as well as to all external resources, such as memory and I/O. This enables the operating system to control both processor and external functions.

In addition, the NS16000 processors provide separate address spaces for each running process, thus protecting one user from another.

The MC68020 implements a concept of multiple access levels, which provides expansion on up to 256 hierarchical levels, which present a superset of ring architecture.

Security refers to the limited access to information. The basic principle is to allow a program to access only what it needs to know. For example, Linden suggests that "...almost every procedure should run in a protection domain that gives it an access to exactly what it needs to accomplish its function, and nothing more [32]." The security is provided by giving each process certain access rights to a page or a segment. The most commonly used access rights are:

1. *read access:* a process may obtain any information from the page or the segment.
2. *write access:* a process may modify the page or the segment, and may place additional information in it. The process may destroy all of the information in the page or the segment.
3. *execute access:* a process may run the page or the segment as a program. Execute access is given to pages or segments which are programs, and denied to data pages or segments.

Current processors typically store the access rights in page or segment descriptors. Before the processor accesses a page or a segment, it first checks its access rights, and if they are verified, it may access the selected page or segment. The diagram in Figure 24 illustrates the described mechanism, based on access rights stored in the page or segment descriptors. The character N indicates that the corresponding page or segment cannot be accessed at all.

The segmentation virtual memory system provides a more natural security system in a paging system. The logical address space is divided into pages, and the described mechanism cannot protect the program modules precisely. It either protects too little or too much. In the segmentation system, each segment is of specific length, and the way to protect segments by using access rights is more natural.

Regardless of the implemented virtual memory system, the drawback of the described security mechanism is that all users have the same access rights to common pages or segments, because the access rights are associated with the pages or the segments, and not with the users.

This problem can be solved by using two-level mapping scheme, as described in Section 3.2, for the case of the i432 processor [27]. In this two-level mapping scheme, the access rights are stored independently of the segment (or page) descriptors, and are associated with the users, and not with the segments (or pages).

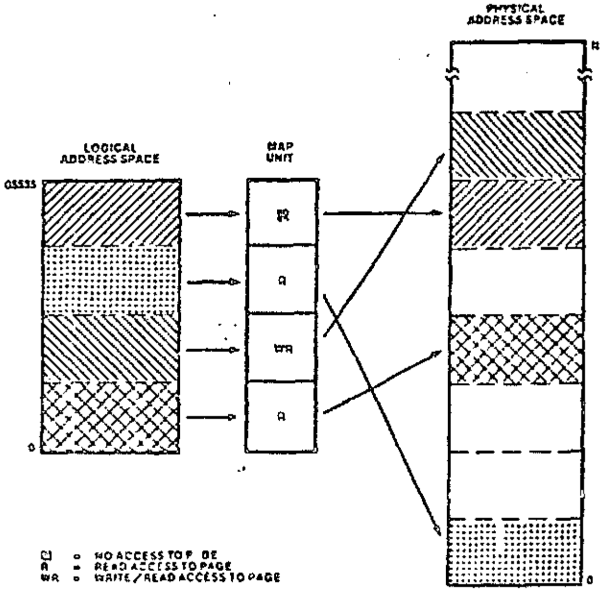

Figure 24. Security technique based on access rights stored in the page or segment descriptors [27]

6. DISCUSSION AND CONCLUSION

We have discussed in this paper several issues related to memory management in advanced microprocessors. All these concepts are not new; they are known for years from the operating system's theory and practice, however the approaches are sometimes modified, and implementation techniques may be different, in comparison with the minicomputer and mainframe environments.

The processor architect must make several crucial decisions related to the processor architecture, which have to support memory management and virtual memory. The main decisions to be made are listed in Figure 25.

The on-chip MMU versus off-chip MMU is one of the basic decisions which has to be made. Both concepts have advantages, as well as drawbacks. These have been discussed in the paper. In addition, the on-chip MMU has an advantage over the off-chip MMU which is related to cache memory design. An external MMU requires logical address caches to bypass the MMU delay, while the internal MMU implements the physical address cache. The logical address cache requires special address tag hardware, large operating system overhead on task switch, and flush cache when sharing data.

The issue related to virtual memory system: paging versus segmentation, is of crucial importance. Again, some microprocessors support paging, other processors support segmentation, while few microprocessors support both systems, in which case the mode is user selectable. Anyhow, it seems that the paging system has advantages over the segmentation system, and almost all 32-bit microprocessors support it.

The next two questions are related to the implementation of the address translation mechanism: levels of mapping and use of an associative cache memory. Both multi-level mapping and a small associative cache memory significantly improve system performance, and thus they should be built into an advanced microprocessor architecture. Practically, all 32-bit microprocessors have implemented these two concepts in their architecture.

MMU ON-CHIP	versus	MMU OFF-CHIP
PAGING	versus	SEGMENTATION
ONE-STAGE MAPPING	versus	MULTI-STAGE MAPPING
CACHE MEMORY-YES	versus	CACHE MEMORY-NO
INSTRUCTION RESTART	versus	INSTRUCTION CONTINUATION
PROTECTION BUILT-IN	versus	PROECTION IN SOFTWARE

Figure 25. A list of questions for the processor architect

Techniques to support the virtual memory system, especially the choice of the techniques to implement resume operation, after an address fault is detected and corrected, is also an important decision for the architect. The instruction restart method seems to be more efficient than the instruction continuation method, especially if the MMU is on the CPU chip. Then, due to pipelined nature of architectures in modern microprocessors, the address fault can be detected before a memory access. This significantly simplifies the restart of the faulted instruction.

Finally, the protection mechanism built in the architecture (such as the ring system in the i286/i386 processors) provides a powerful tool for an operating system designer, and reduces software overhead. On the other hand, because the protection system is already defined in the architecture, there is no choice for the OS designer, but to implement the available mechanism, whether he (she) likes it or not.

The other approach, in which the processor provides some basic protection elements, but not the whole protection system (such as supervisor/user modes and access concepts in the MC68020), requires from the OS designer to create the protection system in software, thus increasing the software overhead. However, this approach is more flexible.

We may conclude that the memory management architectures in current microprocessors are coming of age. However, one of the most challenging aspects of future processor design will be to provide more elegant solutions to all these problems, as well as to enable a more complete integration of memory management and virtual memory support.

7. ACKNOWLEDGEMENT

The authors are thankful to Jeff Pridmore and Walt Helbig, of RCA, for their comments.

8. REFERENCES

1. Aho, A.V., Denning, P.J., and Ullman, J.D., "Principles of Optimal Page Replacement," *JACM*, Vol. 18, No. 1, January 1971, pp. 80-93

2. Alpert, D., "Powerful 32-bit Micro Includes Memory Management," *Computer Design*, October 1983, pp. 213-220.

3. Alpert, D., Carberyy, D., Yamamura, M., Chow, Y., and Mak, P., "32-bit Processor Chip Integrates Major System Functions," *Electronics*, July 14, 1983, pp. 113-119.

4. *"An Architectural Contrast: The M68000 Microprocessor Family and the 8086/iAPX 286,"* Motorola Corp., November 1983.

5. Andrews, R., "The Z80,000 Processor Chip Integrates Major System Functions," *Proceedings of the IEEE Mini/Micro Southeast*, Orlando, Florida, January 1984, pp. 5.2.1-5.2.7.

6. Bal, S., et al, "The NS16000 Family - Advances in Architecture and Hardware," *IEEE Computer*, June 1982, pp. 58-67.

7. Beyers, J.W., et al, "A 32-bit VLSI CPU Chip," *IEEE Journal of Solid-State Circuits*, Vol. 16, October 1981, pp. 537-541.

8. Chamberlin, D.D., Fuller, S.H., and Lin, L., "An Analysis of Page Allocation Strategies for Virtual Memory Systems," *IBM Journal of R&D*, Vol. 17, 1973, pp. 404-412.

9. Chu, W.W., and Opderbeck, H., "Program Behaviour and the Page-Fault-Frequency Replacement Algorithm," *IEEE Computer*, November 1976, pp. 29-38.

10. Deitel, H.M., *"An Introduction to Operating Systems,"* Addison-Wesley Publishing Company, 1984.

11. Denning, P.J., "Virtual Memory," *ACM Computing Surveys*, Vol. 2, No. 3, September 1970, pp. 153-189.

Denning, P.J., "Working Sets Past and Present," *IEEE Transactions on Software Engineering*, Vol. 6, No. 1, January 1980, pp. 64-84.

13. Denning, P.J., and Schwartz, S., "Properties of the Working-Set Model," *Communications of the ACM*, Vol. 15, No. 3, March 19872, pp. 191-198.

14. Dennis, J.B., "Segmentation and the Design of Multiprogrammed Computer Systems," *JACM*, Vol. 12, No. 4, October 1965, pp. 589-602.

15. Diodato, P.W., et al, "CAD Construction of a VLSI Memory Management Unit," *Proceedings of the ICCAD*, 1983.

16. Doran, R.W., "Virtual Memory," *IEEE Computer*, October 1976, pp. 27-37.

17. Goksel, A.K., et al, "A Memory Management Unit for a Second Generation Microprocessor," *Proceedings of the Compcon*, 1984.

18. Goksel, A.K., et al, "A VLSI Memory Management Chip: Design Considerations and Experience," *IEEE Journal on Solid-State Circuits*, Vol. 19, No. 3, June 1984.

19. Gupta, A., and Toong, H.D., "An Architectural Comparison of 32-bit Microprocessors," *IEEE Micro*, Vol. 3, No. 1, February 1983, pp. 9-22.

20. Hansen, D.J., "Programming Motorola's 32-bit Microprocessor the 68010," *Proceedings of the IEEE Mini/Micro Southeast*, Orlando, Florida, January 1984, pp. 4.1.1-4.1.9.

21. Heller, P., "The Intel iAPX 286 Microprocessor," *Proceedings of the Wescon*, 1981, pp. 1.3.1-1.3.4.

22. Heller, P., Childs, R., and Slager, J., "Memory Protection Moves Onto 16-bit Microprocessor Chip," *Electronics*, Vol. 55, Feb. 24, 1982, pp. 133-137.

23. Hirschberg, S.D., "A Class of Dynamic Memory Allocation Algorithms," *Communications of the ACM*, Vol. 16, No. 10, October 1973, pp. 615-618.

24. Hoeschene, H.A., et al, "A Second Generation 32-bit CMOS Microprocessor," *Proceedings of the Compcon*, 1984.

25. Hunter, C.B., and Farquhar, E., "Introduction to the NS16000 Architecture," *IEEE Micro*, April 1984, pp. 26-47.

26. *"IAPX 286 Operating Systems Writer's Guide,"* Intel Corporation, Santa Clara, 1983.

27. *"Introduction to the iAPX 432 Architecture,"* Intel Corporation, Santa Clara, 1981.

28. *"Introduction to the iAPX 286,"* Intel Corporation. Santa Clara, 1982.

29. Kaminker, A., et al, "A 32-bit Microprocessor with Virtual Memory Support," *IEEE Journal of Solid-State Circuits*, October 1981, pp. 230-231.

30. Knowlton, K.C., "A Fast Storage Allocator," *Communications of the ACM*, Vol. 8, No. 10, October 1965, pp. 623-625.

31. Levy, H.M., and Lipman, P.H., "Virtual Memory Management in the VAX/VMS Operating System," *IEEE Computer*, March 1982, pp. 35-41.

32. Linden, T.A., "Operating System Structures to Support Security and Reliable Software," *ACM Computing Surveys*, Vol. 8, No. 4, Dec. 1976, pp. 419.

33. Look, H., "Virtual Memory for Zilog's 8-, 16-, and 32-bit Microprocessors," *Proceedings of the IEEE Mini/Micro Southeast*, Orlando, Florida, January 1984, paper 3.3.

34. MacGregor, D., "Hardware and Software Strategies for the MC68020," *EDN*, June 20, 1985, pp. 89-98.

35. MacGregor, D., Mothersole, D., and Moyer, B., "The Motorola MC68020," *IEEE Micro*, August 1984, pp. 101-118.

36. MacGregor, D., and Mothersole, D.S., "Virtual Memory and the MC68010," *IEEE Micro*, June 1983, pp. 24-38.

37. Martin, G., "Virtual Memory Management Expands Microprocessors," *Computer Design*, June 1983, pp. 169-178.

38. Mateosian, R., "Elegance is Everything in NS 16000 Memory Management," *Proceedings of the IEEE Mini/Micro Southeast*, Orlando, Florida, January 1984, paper 3.2.

39. Mateosian, R., "Operating System Support - the Z8000 Way," *Computer Design*, May 1982, pp. 255-261.

40. Mazor, S., Wharton, S., "Compact Code iAPX 432 Addressing Techniques," *Computer Design*, May 1982, pp. 249-253.

41. Mazor, S., Wharton, S., "Promote User Privacy Through Secure Memory Areas," *Computer Design*, October 1982, pp. 89-92.

42. *"MC68020 32-bit Microprocessor User's Manual,"* Prentice-Hall, 1984.

43. Myers, G.J., *"Advances in Computer Architecture,"* John Wiley & Sons, 1978.

44. Philips, D., "Memory-Management Strategies Suit Different Application Areas," *EDN*, September 1984, pp. 135-143.

45. Peterson, J.L., Theodore, N., "Buddy Systems," *Communications of the ACM*, June 1977, Vol. 20, No. 6, pp. 421-431.

46. Pohn, A.V., and Smay, T.A., "Computer Memory Systems," *IEEE Computer*, October 1981, pp. 93-110.

47. Pollack, F. J., et al, "Supporting ADA Memory Management in the iAPX-432," *ACM* 1982, pp. 117-130.

48. Purdom, P.W., and Stigler, S.M., "Statistical Properties of the Buddy System," *Journal of the ACM*, Vol. 17, No. 4, October 1970, pp. 683-697.

49. Saltzer, J.H., and Schroeder, M., "The Protection of Information in Computer Systems," *Proceedings of the IEEE*, Vol. 63, No. 9, September 1975, pp. 1278.

50. Skoog, S.K., "Memory Management with the NCR/32 Scipset," *Proceedings of the IEEE Mini/Micro Southeast*, Orlando, Florida, January 1984, paper 3.1.

51. Stockton, J.F., "A Virtual Breakthrough for Micros, *Computer Design*, pp. 153-162.

52. Stockton, J.F., "The M68451 Memory Management Unit," *Electronic Engineering*, Vol. 54, May 1982, pp. 54-73.

53. Timms, B., "Z80,000 Mainframe Resources Optimize the Software Environment," *Proceedings of the IEEE, Mini/Micro Southeast*, Orlando, Florida, January 1984, pp. 4.4.1-4.4.13.

54. *"Touch the Future,"* Intel Design Seminar, Miami, Florida, 1985.

55. Turker, R., and Levy, H., "Segmented FIFO Page Replacement," *Proceedings of the ACM Conference on Measurement and Modeling of Computer Systems*, Las Vegas, Nevada, September 1981, pp. 48-51.

56. Walters, S., "Memory Management Made Easy with the Z8000," *Proceedings of the Wescon*, 1981, pp. 9.3.1-9.3.9.

9. ABOUT THE AUTHORS

Dr. B. P. Furht is on the faculty of the Department of Electrical and Computer Engineering, University of Miami, Coral Gables, Florida. He has published over 60 technical papers, and 2 books. He is the author of *Microprocessor Interfacing and Communication* (Reston 1985), and coeditor of the *Tutorial on Advanced Topics in Computer Architecture* (IEEE Press, 1985). His current research activities include high-level language computer architectures, multiprocessor systems, and architectures for virtual memory management. He presented over 30 invited lectures in Europe, North, and Latin America on various topics related to computer architecture. He has been involved in consulting activities for a number of companies such as NASA, RCA, Cordis, Honeywell, and others. He is a member of the IEEE, and a chief editor of the International Journal of Mini and Microcomputers.

Dr. V. M. Milutinović is on the faculty of the School of Electrical Engineering, Purdue University. He has published over 60 technical papers, 2 original books, and 4 edited books. His research papers have been published in IEEE Transactions, IEE Proceedings, IEEE Computer, and other refereed journals. One of his books has been republished (in various forms) in several languages. He is the editor of the IEEE Press *Tutorial on Advanced Microprocessors and High-Level Language Computer Architecture*, and the coeditor of the IEEE Press *Tutorial on Advanced Topics in Computer Architecture*. He is the editor and the contributing author for two multiauthor books on computer architecture. His pioneering paper on GaAs computer architecture for VLSI has been scheduled to appear in the September issue of IEEE Computer. He presented over 40 invited lectures in Europe, North, and Latin America. His current interests include VLSI computer architecture for GaAs, high-level language computer architecture, and microprocessor systems for AI. His current research support is equal to about $250k per year, predominantly in the area of VLSI computer architecture for GaAs. He has consulted for a number of high-tech companies, including Intel, Honeywell, NASA, RCA, and others. He is currently involved in the industrial implementation of a 32-bit VLSI microprocessor in the GaAs technology, with responsibilities in the microarchitecture domain. He is a member of the IEEE, and is on the EUROMICRO Board of Directors.