WIKIPEDIA

Memory segmentation

Memory segmentation is a computer (primary) memory management technique of division of a computer's primary memory into **segments** or **sections**. In a computer system using segmentation, a reference to a memory location includes a value that identifies a segment and an offset (memory location) within that segment. Segments or sections are also used in object files of compiled programs when they are linked together into a program image and when the image is loaded into memory.

Segments usually correspond to natural divisions of a program such as individual routines or data tables[1] so segmentation is generally more visible to the programmer than paging alone.[2] Different segments may be created for different program modules, or for different classes of memory usage such as code and data segments.[3] Certain segments may be shared between programs.[1][2]

Segmentation was originally invented as a method by which system software could isolate different software processes (tasks) and data they are using. It was intended to increase reliability of the systems running multiple processes simultaneously.[4] In a x86-64 architecture it is considered legacy and most x86-64-based modern system software don't use memory segmentation. Instead they handle programs and their data by utilizing memory-paging which also serves as a way of memory protection. However most x86-64 implementations still support it for backward compatibility reasons.[4]

Contents

Hardware implementation

In a system using segmentation, computer memory addresses consist of a segment id and an offset within the segment.[3] A hardware memory management unit (MMU) is responsible for translating the segment and offset into a physical address, and for performing checks to make sure the translation can be done and that the reference to that segment and offset is permitted.

Each segment has a length and set of permissions (for example, *read*, *write*, *execute*) associated with it.[3] A process is only allowed to make a reference into a segment if the type of reference is allowed by the permissions, and if the offset within the segment is within the range specified by the length of the segment. Otherwise, a hardware exception such as a segmentation fault is raised.

Segments may also be used to implement virtual memory. In this case each segment has an associated flag indicating whether it is present in main memory or not. If a segment is accessed that is not present in main memory, an exception is raised, and the operating system will read the segment into memory from secondary storage.

Segmentation is one method of implementing memory protection.[5] Paging is another, and they can be combined. The size of a memory segment is generally not fixed and may be as small as a single byte.[6]

Segmentation has been implemented in several different ways on different hardware, with or without paging. Intel x86 memory segmentation does not fit either model and is discussed separately below, and also in greater detail in a separate article.

Segmentation without paging

Associated with each segment is information that indicates where the segment is located in memory— the *segment base*. When a program references a memory location the offset is added to the segment base to generate a physical memory address.

An implementation of virtual memory on a system using segmentation without paging requires that entire segments be swapped back and forth between main memory and secondary storage. When a segment is swapped in, the operating system has to allocate enough contiguous free memory to hold the entire segment. Often memory fragmentation results if there is not enough contiguous memory even though there may be enough in total.

Segmentation with paging

Instead of an actual memory location the segment information includes the address of a page table for the segment. When a program references a memory location the offset is translated to a memory address using the page table. A segment can be extended simply by allocating another memory page and adding it to the segment's page table.

An implementation of virtual memory on a system using segmentation with paging usually only moves individual pages back and forth between main memory and secondary storage, similar to a paged non-segmented system. Pages of the segment can be located anywhere in main memory and need not be contiguous. This usually results in a reduced amount of input/output between primary and secondary storage and reduced memory fragmentation.

History

The Burroughs Corporation B5000 computer was one of the first to implement segmentation, and "perhaps the first commercial computer to provide virtual memory"[7] based on segmentation. The later B6500 computer also implemented segmentation; a version of its architecture is still in use today on the Unisys ClearPath Libra servers.

The GE-645 computer, a modification of the GE-635 with segmentation and paging support added, was designed in 1964 to support Multics.

The Intel iAPX 432,[8] begun in 1975, attempted to implement a true segmented architecture with memory protection on a microprocessor.

The 960MX version of the Intel i960 processors supported load and store instructions with the source or destination being an "access descriptor" for an object, and an offset into the object, with the access descriptor being in a 32-bit register and with the offset computed from a base offset in the next register and from an additional offset and, optionally, an index register specified in the instruction. An access descriptor contains permission bits and a 26-bit object index; the object index is an index into a table of object descriptors, giving an object type, an object length, and a physical address for the object's data, a page table for the object, or the top-level page table for a two-level page table for the object, depending on the object type.[9]

Prime, Stratus, Apollo, IBM System/38, and IBM AS/400 computers use memory segmentation.

x86 architecture

The memory segmentation used by early x86 processors, beginning with the Intel 8086, does not provide any protection. Any program running on these processors can access any segment with no restrictions. A segment is only identified by its starting location; there is no length checking. The segment starting address granularity is 16 bytes and the offset is 16 bits, supporting segment sizes up to 64 KiB, so segments can (and often do) overlap and each physical address can be denoted by 4096 different segment–offset pairs (allowing for address offset wrap-around).

Segmentation in the Intel 80286 and later provides protection: with the introduction of the 80286, Intel retroactively named the sole operating mode of the previous x86 CPU models "real mode" and introduced a new "protected mode" with protection features. For backward compatibility, all x86 CPUs start in "real mode" with no memory protection, fixed 64 KiB segments, and only 20-bit (1024 KiB) addressing. An 80286 or later processor must be switched into another mode by software in order to use its full address space and advanced MMU features.

The Intel 80386 and later processors also support paging; in those processors, the segment table, rather than pointing to a page table for the segment, contains the segment address in *linear memory*. Addresses in linear memory are then mapped to physical addresses using a separate page table, if paging is enabled.

The x86-64 architecture does not use segmentation in long mode (64-bit mode).[10] In a x86-64 architecture it is considered legacy and most x86-64-based modern system software don't use memory segmentation. Instead they handle programs and their data by utilizing memory-paging which also serves as a way of memory protection. Though most x86-64 implementations still support it for backward compatibility reasons.[4] Four of the segment registers: CS, SS, DS, and ES are forced to 0, and the limit to 2^{64}. The segment registers FS and GS can still have a nonzero base address. This allows operating systems to use these segments for special purposes.

See also

- Memory management (operating systems)
- Virtual address space
- Virtual memory
- Data segment
- BSS Segment
- x86 memory segmentation
- Segmentation fault
- Flat memory model

References

1. Holt, A. W. (1961). "Program Organization and Record Keeping for Dynamic Storage Allocation". *Communications of the ACM*.
2. Englander, Irv (2003). *The architecture of computer hardware and systems software* (3rd ed.). Wiley. ISBN 0-471-07325-3.
3. Glaser, E. L.; Couleur, J. F.; Oliver, G. A. (1965). *System Design of a Computer for Time Sharing Applications* (https://multicians.org/fjcc2.html). 1965 Fall Joint Computer Conference.
4. "1.2 Memory Management". *AMD64 Technology AMD64 Architecture Programmer’s Manual Volume 2: System Programming* (https://www.amd.com/system/files/TechDocs/24594.pdf) (PDF). **2**. Advanced Micro Devices. 2018. p. 5.
5. Arpaci-Dusseau, Remzi H.; Arpaci-Dusseau, Andrea C. (2014). "Segmentation" (http://pages.cs.wisc.edu/~remzi/OSTEP/vm-segmentation.pdf) (PDF). *Operating Systems: Three Easy Pieces*. Arpaci-Dusseau Books.
6. *Intel® 64 and IA-32 Architectures Software Developer’s Manual Volume 3 (3A, 3B & 3C): System Programming Guide* (http://download.intel.com/products/processor/manual/325384.pdf) (PDF). Intel Corporation. 2012. pp. 3–13.
7. Mayer, Alastair J.W. "The Architecture of the Burroughs B5000 - 20 Years Later and Still Ahead of the Times?" (http://www.smecc.org/The%20Architecture%20%20of%20the%20Burroughs%20B-5000.htm). Retrieved March 15, 2012.
8. *Introduction to the IAPX 432 Architecture* (http://bitsavers.org/components/intel/iAPX_432/171821-001_Introduction_to_the_iAPX_432_Architecture_Aug81.pdf) (PDF). Intel Corporation. 1981. p. 78.
9. *BiiN CPU Architecture Reference Manual* (http://bitsavers.org/pdf/biin/BiiN_CPU_Architecture_Reference_Man_Jul88.pdf) (PDF). BiiN. July 1998.
10. *AMD64 Technology AMD64 Architecture Programmer’s Manual Volume 2: System Programming* (https://www.amd.com/system/files/TechDocs/24594.pdf) (PDF). **2**. Advanced Micro Devices. 2018.

External links

- IA-32 Intel Architecture Software Developer’s Manual Volume 3A: System Programming Guide. http://www.intel.com/products/processor/manuals/index.htm.
- Operating Systems: Internals and Design Principles by William Stallings. Publisher: Prentice Hall. ISBN 0-13-147954-7. ISBN 978-0-13-147954-8.