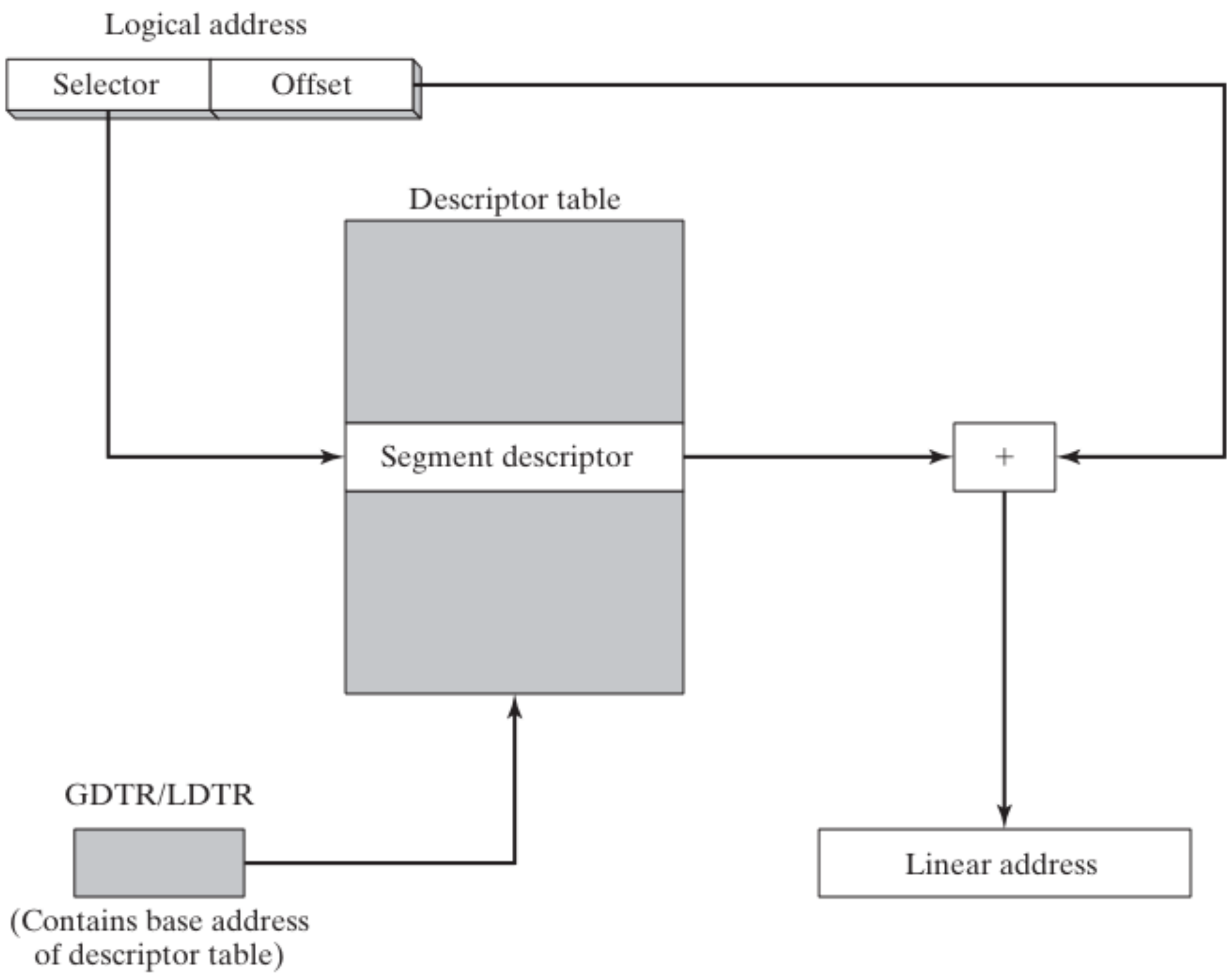
x86 Memory Management

Logical addresses and linear addresses are two different ways of addressing memory in an x86 processor.

A **logical address** is a combination of a segment selector and a 32-bit offset. The segment selector is a 16-bit value that identifies a segment descriptor, which in turn contains information about a memory segment. The offset is a 32-bit value that identifies a location within the segment.

A **linear address** is a 32-bit value that uniquely identifies a location in memory. It is calculated by adding the segment base address to the offset.

The x86 processor uses a two-step process to translate logical addresses to linear addresses:



The **segment selector** is used to index the segment descriptor table

(GDT or LDT) to obtain the segment descriptor.

The **segment base address** is added to the offset to produce the linear address. The following diagram shows the process of translating logical addresses to linear addresses:

```
Logical address = segment selector + offset
Segment base address = segment descriptor table[segment selector]
Linear address = segment base address + offset
```

Once the linear address has been calculated, the processor can use it to access memory directly.

Example

Suppose we have a program that has a variable at offset 200h in a segment with the segment selector value 0x1000. The segment descriptor table contains a segment descriptor for this segment with a base address of 0x100000.

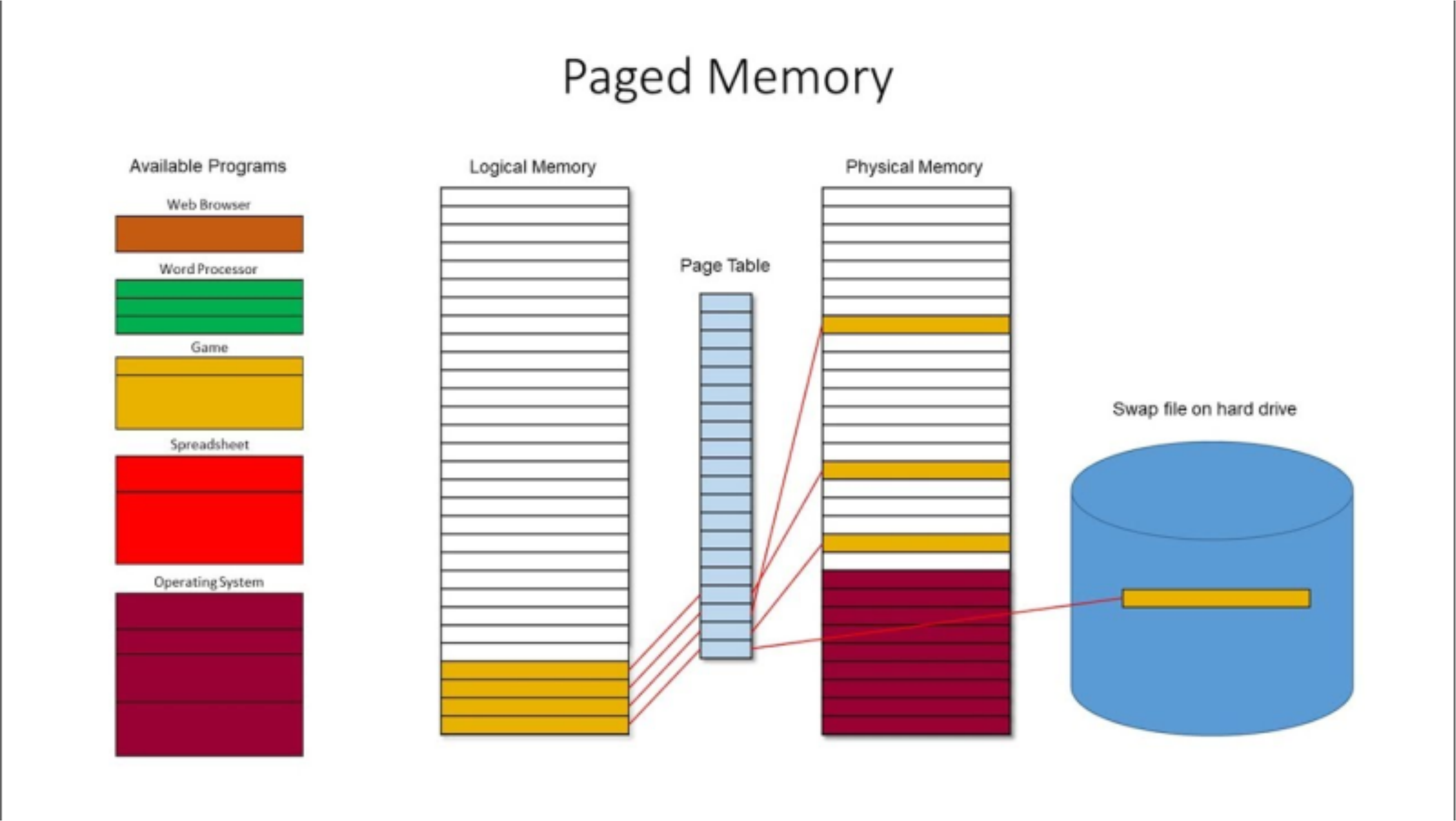
To access this variable, the processor would first calculate the linear address:

```
Linear address = segment base address + offset
Linear address = 0x100000 + 200h
Linear address = 0x100200
```

Once the linear address has been calculated, the processor can use it to access the variable at memory location 0x100200.

Paging

Paging is a memory management technique that allows the operating system to divide physical memory into pages.



Pages are typically 4KB in size, but can also be 2MB or larger.

When a program needs to access memory, the operating system converts the program's linear address to a physical address using a page table.

The **page table** is a data structure that maps linear addresses to physical addresses.

The following diagram shows the process of translating linear addresses to physical addresses using a page table:

```
Linear address = page table index + page offset
Physical address = page table[page table index] + page offset
```

The page table index is the upper 20 bits of the linear address. The page offset is the lower 12 bits of the linear address.

The operating system can use paging to implement a number of features, such as virtual memory and memory protection.

Conclusion

Logical addresses and linear addresses are two different ways of addressing memory in an x86 processor. Logical addresses are used by programs, while linear addresses are used by the processor.

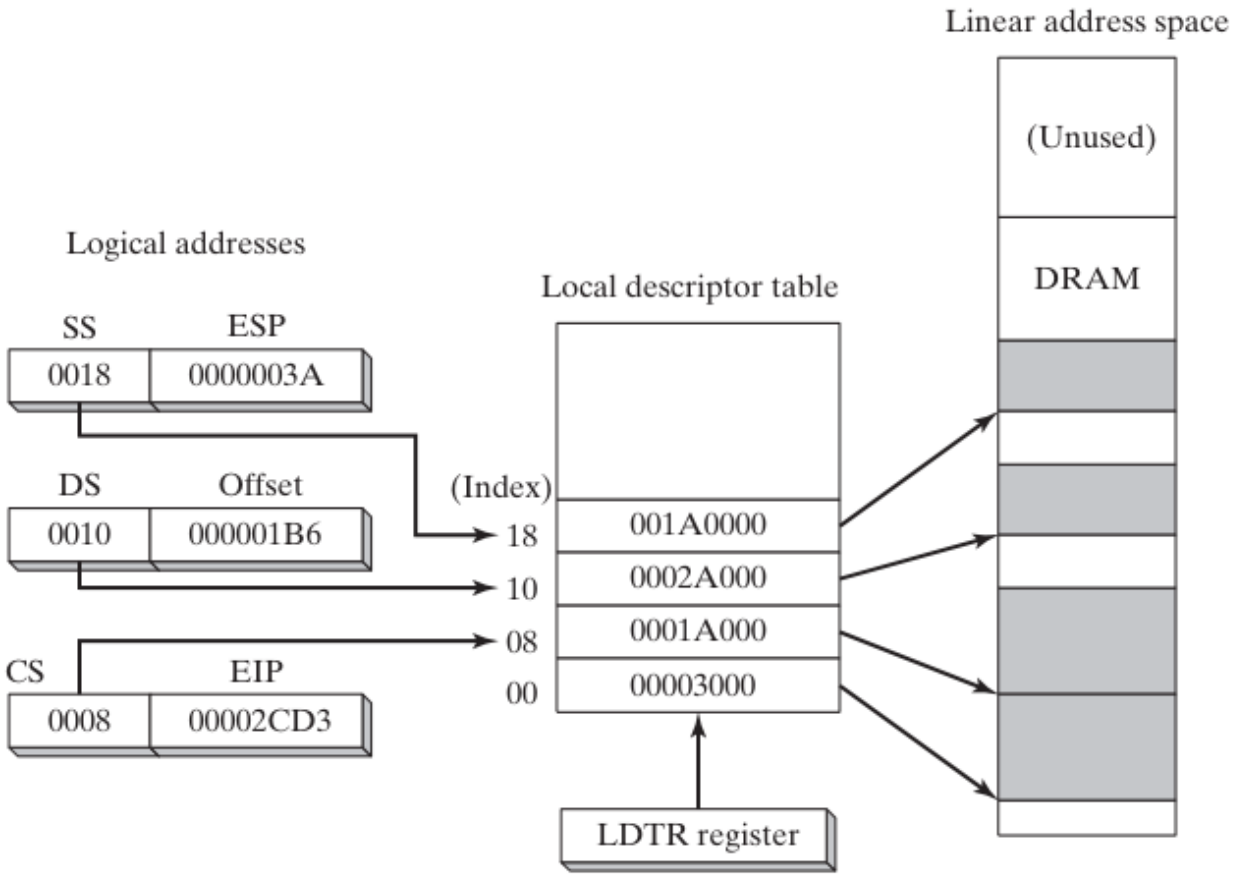
The processor translates logical addresses to linear addresses using segment descriptors. Linear addresses can then be translated to physical addresses using a page table.

=================================

Descriptor tables

=================================

Indexing into a local descriptor table.



Descriptor tables are data structures that contain information about memory segments. A segment is a variable-sized area of memory that is used by a program to store code or data.
There are two types of descriptor tables:

Global Descriptor Table (GDT): The GDT contains segment descriptors for all of the segments that are used by the system.

for all of the segments that are used by the system.

Local Descriptor Table (LDT): Each task or process has its own LDT, which contains segment descriptors for the segments that are used by that task or process.

Segment descriptors contain information about a segment, such as its base address, size, and access rights. The processor uses this information to translate logical addresses to linear addresses.

A logical address is a combination of a segment selector and a 32-bit offset. The segment selector is a 16-bit value that identifies a segment descriptor in the GDT or LDT. The offset is a 32-bit value that identifies a location within the segment.

The processor calculates the linear address by adding the segment base address to the offset. The linear address is then used to access memory directly.

Suppose we have a program that has a variable at offset 200h in a segment with the segment selector value 0x1000.

The GDT contains a segment descriptor for this segment with a base address of 0x100000.

To access this variable, the processor would first calculate the linear address:

```
Linear address = segment base address + offset
Linear address = 0x100000 + 200h
Linear address = 0x100200
```

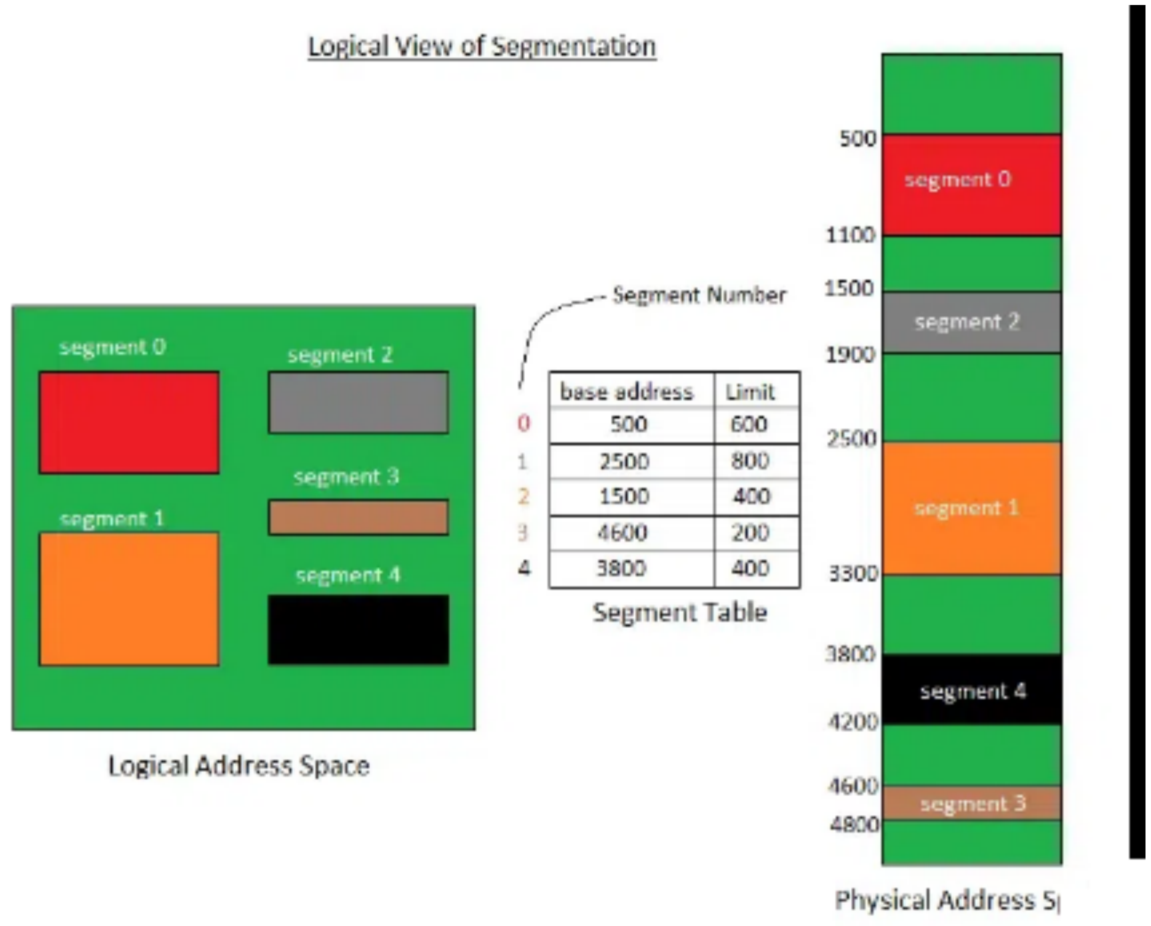
Once the linear address has been calculated, the processor can use it to access the variable at memory location 0x100200.

Segment descriptor details

In addition to the segment's base address, the segment descriptor contains the following information:

Segment limit: The segment limit specifies the maximum size of the segment. If a program tries to access a memory location outside of the segment limit, a processor fault is generated.

Segment type: The segment type specifies the type of data that is stored in the segment. For example, a code segment contains code, and a data segment contains data.



	base address	Limit
0	500	600
1	2500	800
2	1500	400
3	4600	200
4	3800	400

Access rights: The access rights specify which operations are allowed on the segment. For example, a read-only segment can only be read, and a write-only segment can only be written to. The processor uses this information to ensure that programs do not access memory in an unauthorized way.

Segment descriptors in x86 processors contain a number of fields that control how the segment is used, including:

Base address: The starting address of the segment in the linear address space.

Privilege level: The privilege level required to access the segment.

Segment type: The type of segment, such as code, data, or stack.

Segment present flag: Indicates whether the segment is present in memory.

Granularity flag: Determines whether the segment limit is interpreted in bytes or 4096-byte units.

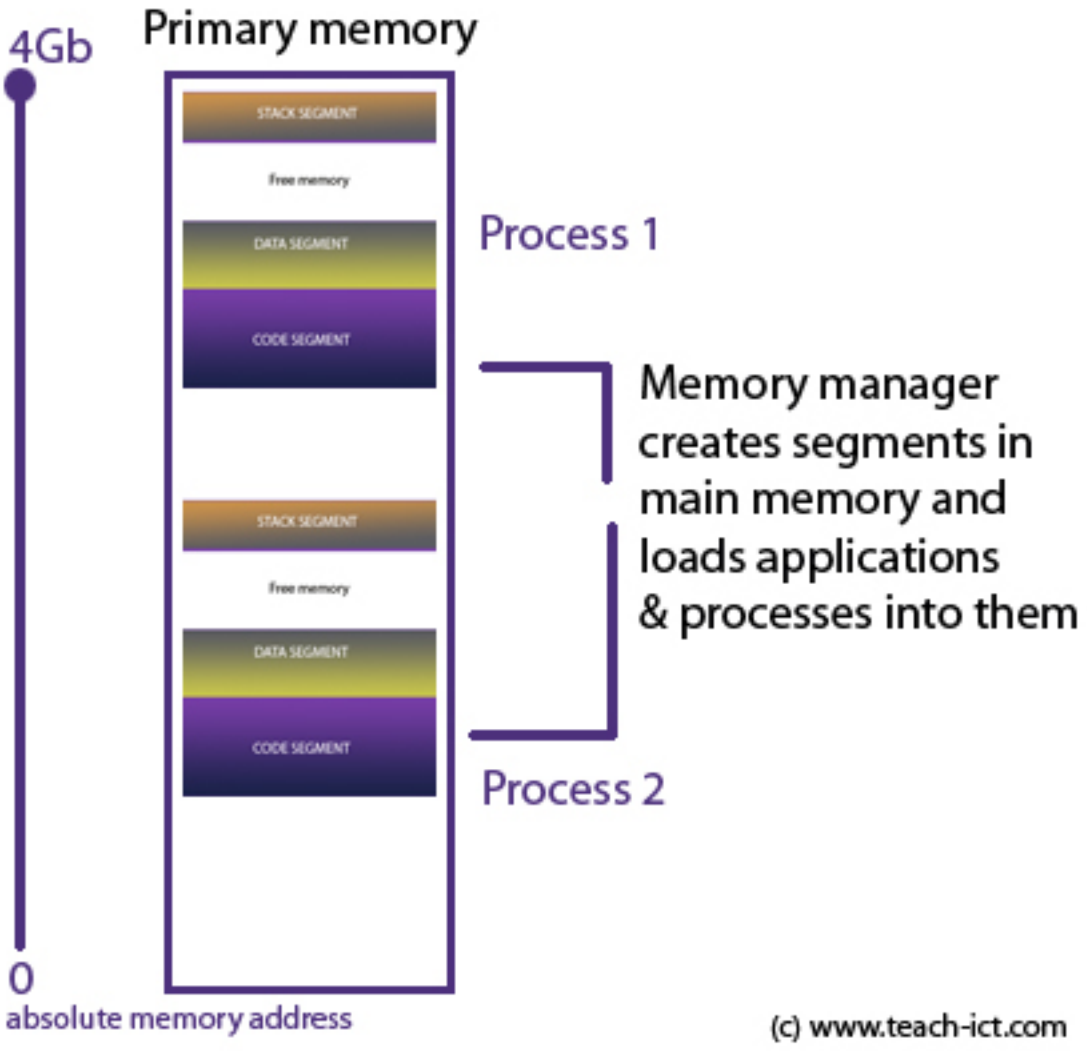
Segment limit: The maximum size of the segment.

The **protection level field** is used to protect operating system data from access by application programs.

Each segment can be assigned a **privilege level between 0 and 3**, where 0 is the most privileged and 3 is the least privileged.

If a program with a **higher privilege level** tries to access a segment with a lower privilege level, a processor fault is generated.

This prevents application programs from **accidentally or maliciously** modifying operating system data.

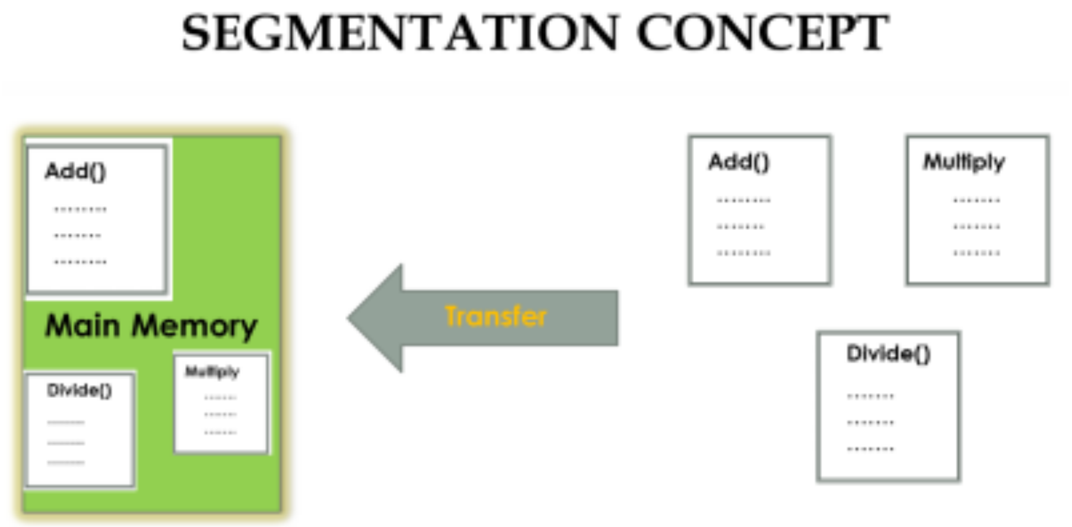


So,

The **segment type field** is used to specify the type of data that is stored in the segment and the type of access that is allowed. For example, a code segment can only be executed, and a data segment can only be read or written to.

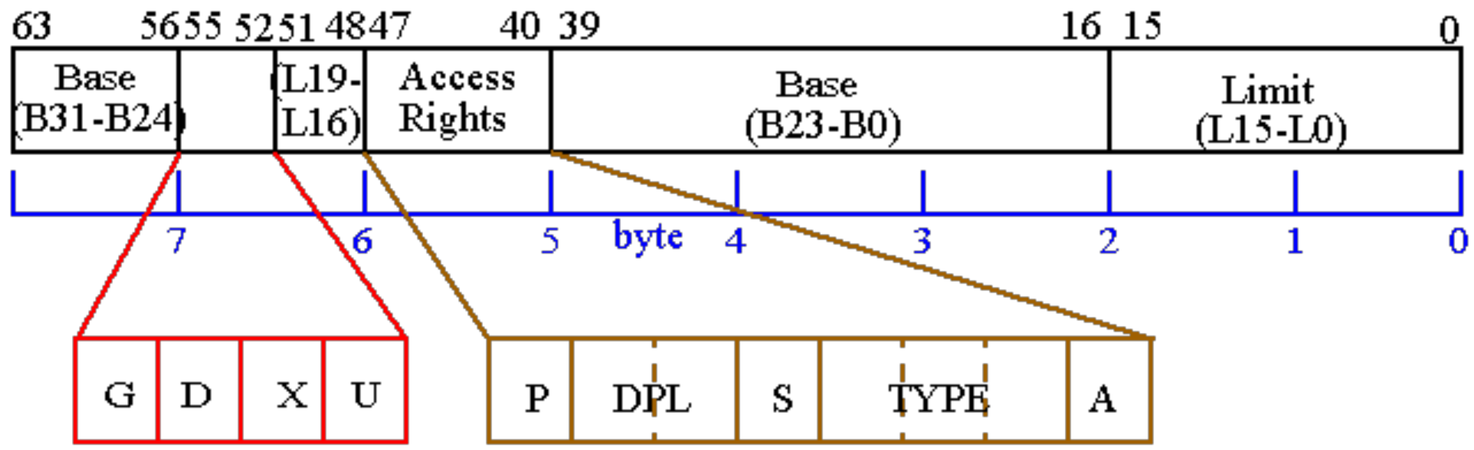
The **segment present flag** is used to indicate whether the segment is currently present in memory. If the flag is set, the segment is present in memory and can be accessed. If the flag is not set, the segment is not present in memory and cannot be accessed.

The **granularity flag** is used to determine how the segment limit field is interpreted. If the flag is set, the segment limit is interpreted in 4096-byte units. If the flag is not set, the segment limit is interpreted in bytes.

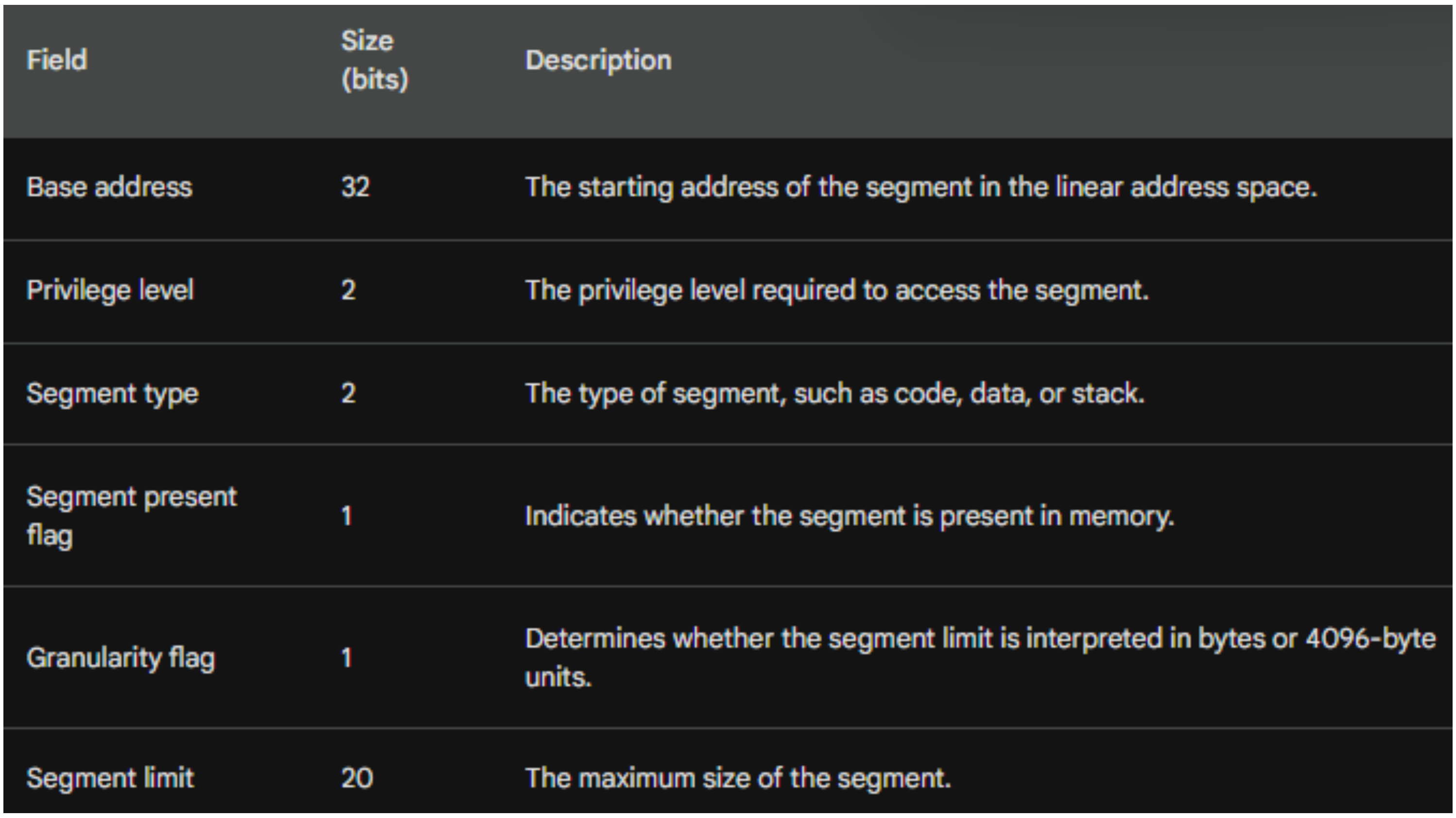


The **segment limit field** specifies the maximum size of the segment. If a program tries to access a memory location outside of the segment limit, a processor fault is generated.

Segment descriptors are an important part of memory management in x86 processors. They allow the processor to translate logical addresses to linear addresses and to ensure that programs do not access memory in an unauthorized way.



Described segment descriptor:



Field	Size (bits)	Description
Base address	32	The starting address of the segment in the linear address space.
Privilege level	2	The privilege level required to access the segment.
Segment type	2	The type of segment, such as code, data, or stack.
Segment present flag	1	Indicates whether the segment is present in memory.
Granularity flag	1	Determines whether the segment limit is interpreted in bytes or 4096-byte units.
Segment limit	20	The maximum size of the segment.

The segment descriptor image also shows the following:

The segment descriptor table (GDT) is located at address 0x00000000.
The segment selector is a 16-bit value that identifies a segment descriptor in the GDT.

The linear address is a 32-bit value that identifies a memory location in the linear address space.

The processor uses the segment selector to index the GDT to obtain the segment descriptor.

The segment descriptor is then used to calculate the linear address of the memory location.

==========================

Page translation:

===========================

Page translation is the process of converting a linear address to a physical address in an x86 processor when paging is enabled.

A linear address is a 32-bit value that uniquely identifies a location in memory. A physical address is also a 32-bit value, but it identifies a location in physical memory.

Paging allows the operating system to divide physical memory into pages, which are typically 4KB in size. The operating system then uses a page table to map virtual addresses to physical addresses.

The page table is a data structure that contains one entry for each page in the virtual address space.

Each entry contains the physical address of the page and its access rights.

When the processor needs to access a memory location, it first translates the linear address to a physical address using the page table.

The processor does this by looking up the page table entry for the linear address. The page table entry contains the physical address of the page and its access rights.

If the page table entry is valid, the processor uses the physical address to access the memory location.

If the page table entry is not valid, the processor generates a page fault.

Steps in page translation

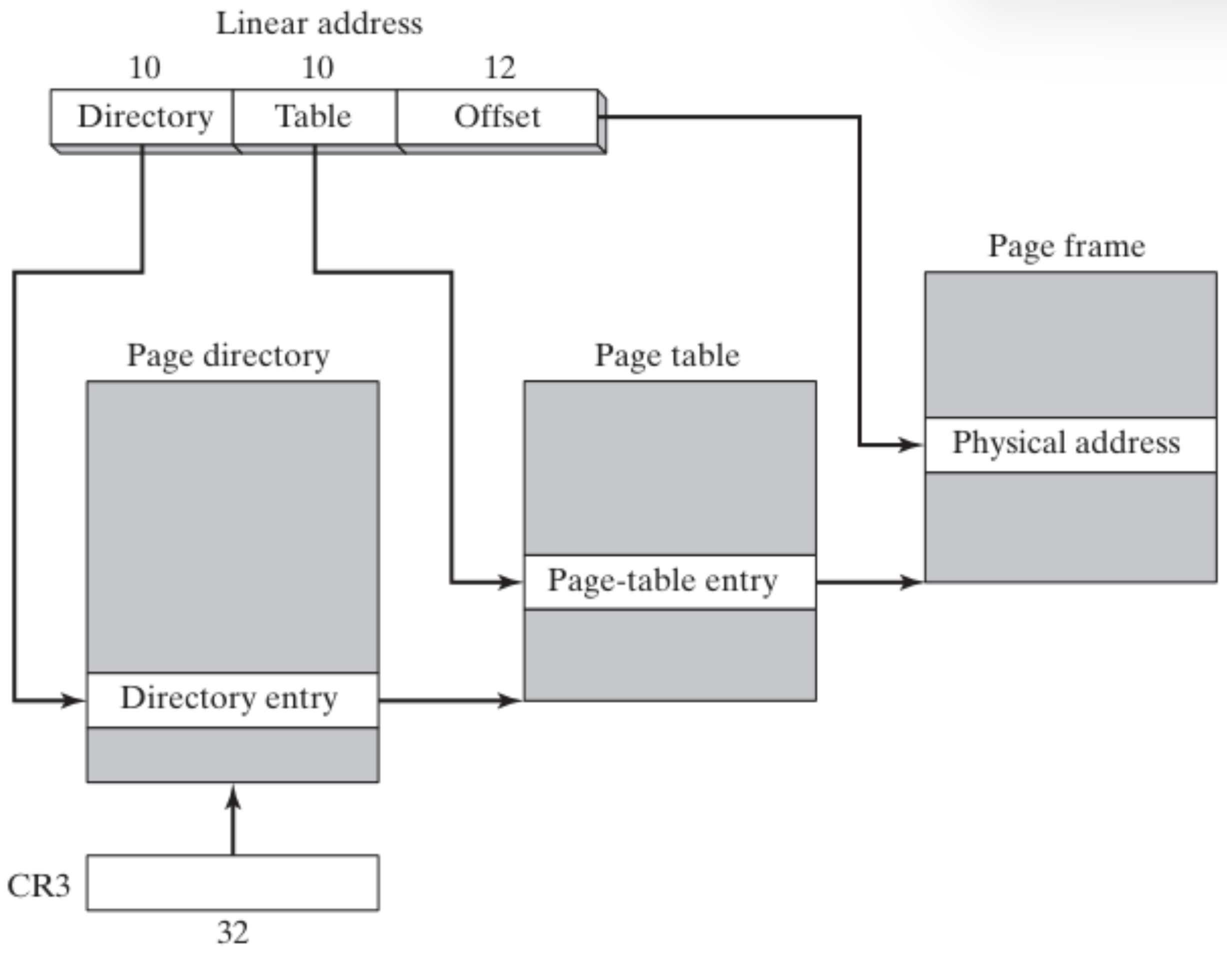
Let's describe the page table image first:

The linear address references a location in the linear address space. The 10-bit directory field in the linear address is an index to a page-directory entry. The page-directory entry contains the base address of a page table.

The 10-bit table field in the linear address is an index into the page table identified by the page-directory entry. The page-table entry at that position contains the base location of a page in physical memory.

The 12-bit offset field in the linear address is added to the base address of the page, generating the exact physical address of the operand.

Translating linear address to physical address.



The following steps are carried out by the processor when translating a linear address to a physical address:

The processor splits the linear address into three fields:

Directory field: The directory field is the upper 10 bits of the linear address.

Table field: The table field is the middle 10 bits of the linear address. Offset field: The offset field is the lower 12 bits of the linear address.

The processor uses the directory field to index the page directory. The page directory is a table of 1024 4-byte entries.

Each entry in the page directory points to a page table. The processor uses the table field to index the page table pointed to by the directory entry.

The page table is also a table of 1024 4-byte entries. Each entry in the page table points to a physical page frame.

The processor adds the offset field to the physical address of the page frame pointed to by the page table entry. This results in the physical address of the memory location. Example:

Suppose we have a linear address of 0x12345678. The **directory field** would be 0x1234, the **table field** would be 0x5678, and the **offset field** would be 0x123456.

The processor would first use the directory field to index the page directory. The page directory entry at index 0x1234 would contain the address of the page table.

The processor would then use the table field to index the page table. The page table entry at index 0x5678 would contain the physical address of the page frame.

Finally, the processor would add the offset field to the physical address of the page frame. This results in the physical address of the memory location, which is 0x12345678.

Conclusion

The operating system has the option of using a single page directory for all running programs and tasks, or one page directory per task, or a combination of the two.

Page translation is an important part of memory management in x86 processors. It allows the operating system to divide physical memory into pages and to map virtual addresses to physical addresses.

This allows the operating system to implement virtual memory and to protect memory from unauthorized access.

==================================

Windows Virtual Machine Manager

==================================

Windows Virtual Machine Manager (VMM) is the 32-bit protected mode operating system at the core of Windows. It is responsible for creating, running, monitoring, and terminating virtual machines. It also manages memory, processes, interrupts, and exceptions.

Microsoft® System Center Virtual Machine Manager™

VMM uses a single 32-bit flat model address space at privilege level 0. This means that all of the virtual machines, the VMM itself, and any virtual devices all share the same address space.

The VMM creates two global descriptor table (GDT) entries for each virtual machine, one for code and one for data. These segments are fixed at linear address 0.

VMM provides multithreaded, preemptive multitasking. This means that it can run multiple applications simultaneously by sharing CPU time between the virtual machines in which the applications run.

How VMM handles memory management

VMM uses a technique called paging to manage memory. Paging divides physical memory into pages, which are typically 4KB in size. VMM then uses a page table to map virtual addresses to physical addresses.

Each virtual machine has its own page table. The page table tells the processor which physical page contains a particular virtual address.

When a virtual machine tries to access a memory location, the processor first looks up the page table entry for that virtual address.

If the page table entry is valid, the processor uses it to access the physical memory location. If the page table entry is not valid, the processor generates a page fault.

Page faults are handled by the VMM. When a page fault occurs, the VMM checks to see if the virtual address is valid. If it is, the VMM loads the corresponding physical page into memory and updates the page table entry.

If the virtual address is not valid, the VMM generates an exception.

Benefits of using VMM

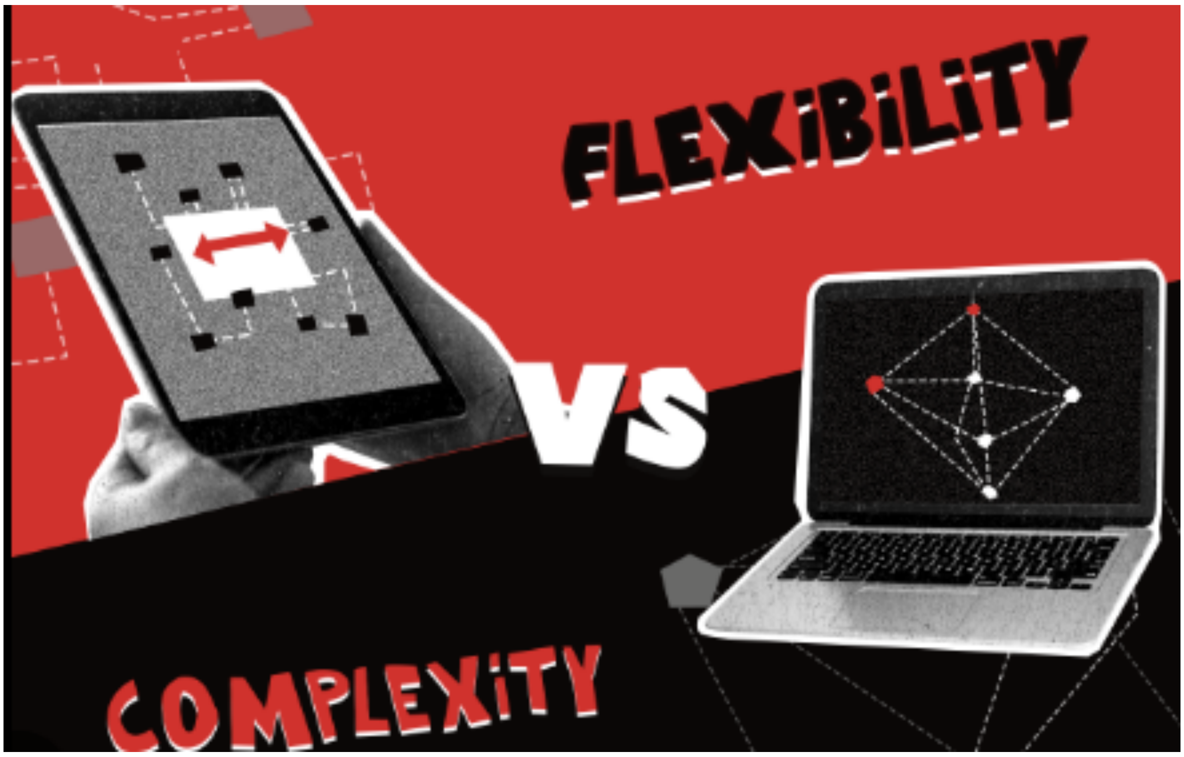
VMM offers a number of benefits, including:

Isolation: VMM isolates virtual machines from each other, so that a failure in one virtual machine does not affect other virtual machines.

Security: VMM can be used to implement security features such as access control and encryption.

Performance: VMM can improve the performance of applications by running them in separate virtual machines.

Flexibility: VMM can be used to create and manage different types of virtual machines, such as servers, desktops, and test environments.



Windows Virtual Machine Manager is a powerful tool that can be used to create, run, and manage virtual machines. VMM offers a number of benefits, including isolation, security, performance, and flexibility.