Modern Operating Systems

Chapter 3 – Segmentation

Zhang Yang

Spring 2022

Content of the Lecture

- 3.7 Segmentation
  - Segmentation
  - Segmentation with Paging

Problem in One-Dimensional Address Space (1)

- The virtual memory solution so far is paging.
  - Paging is a “flat” memory model which means that the programmer sees addresses that start at address 0 to some maximum address.
  - For some applications, it may be useful to allow for entirely different address spaces within a single process...

Problem in One-Dimensional Address Space (2)

- Example
  - A compiler has 5 tables
  - Consider what happens if a program has a much larger than usual number of variables but a normal amount of everything else.

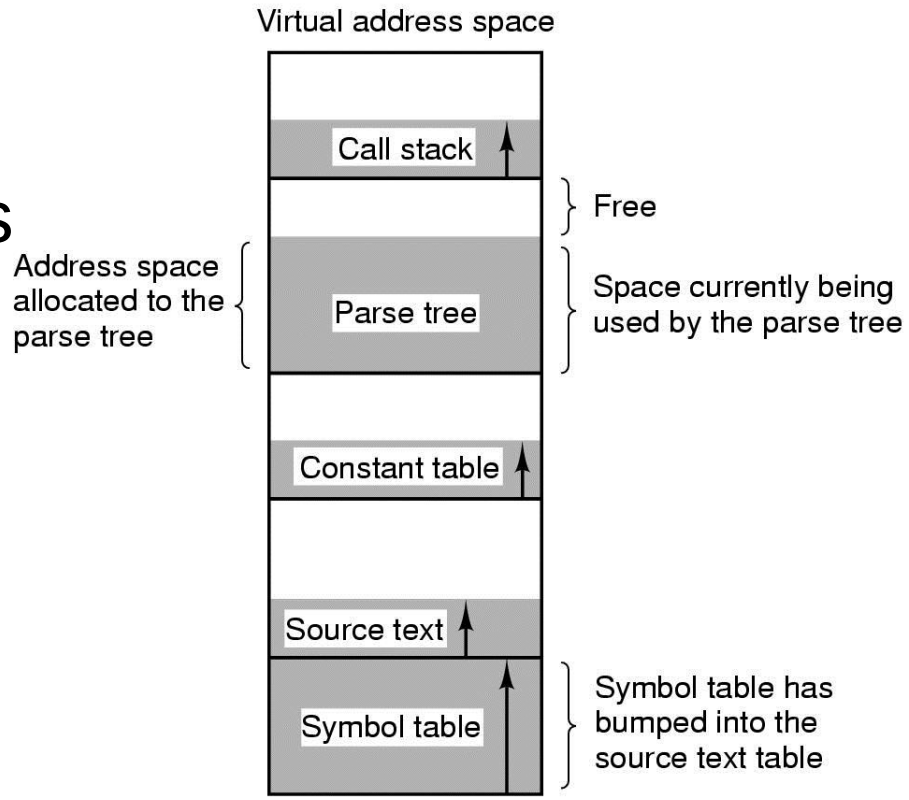


Figure 3-30 In a one-dimensional address space with growing tables, one table may bump into another.

Problem in One-Dimensional Address Space (3)

- Example (ctd.)
  - Solution: To provide the machine with many completely independent address spaces, called segments.

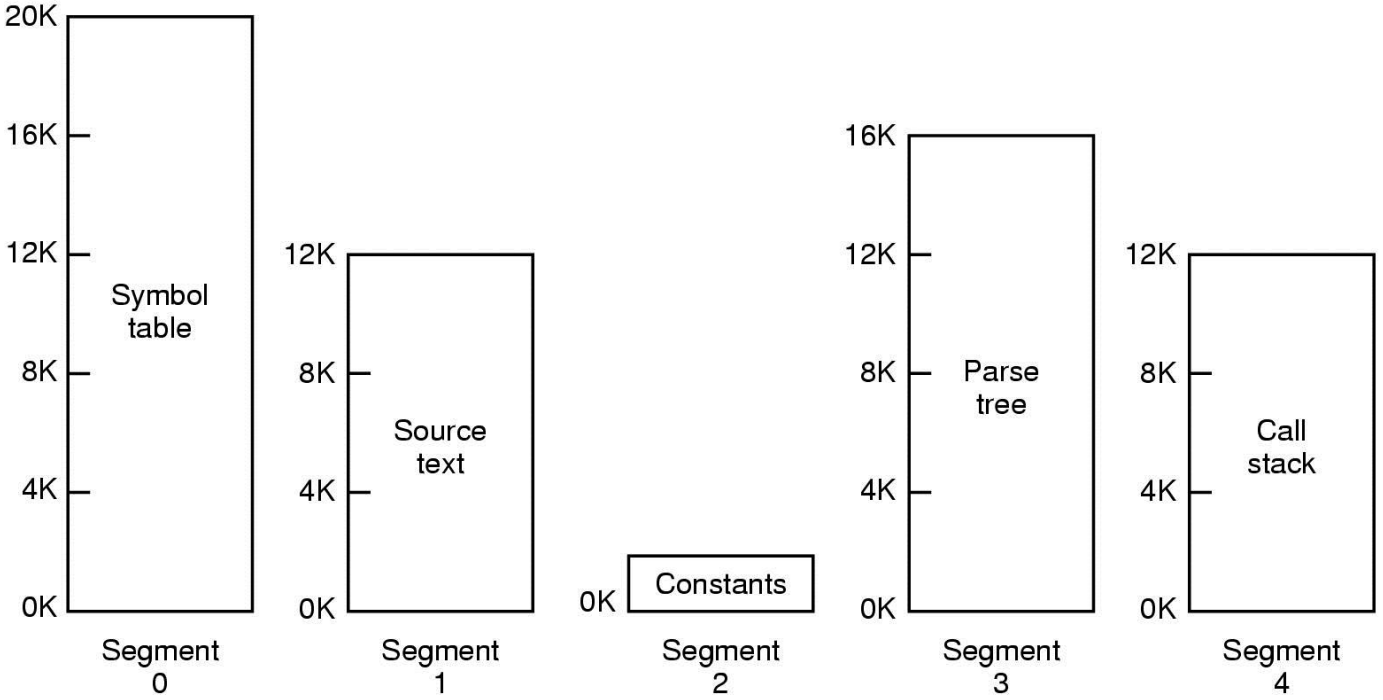


Figure 3-31 A segmented memory allows each table to grow or shrink, independently

Segmentation (1)

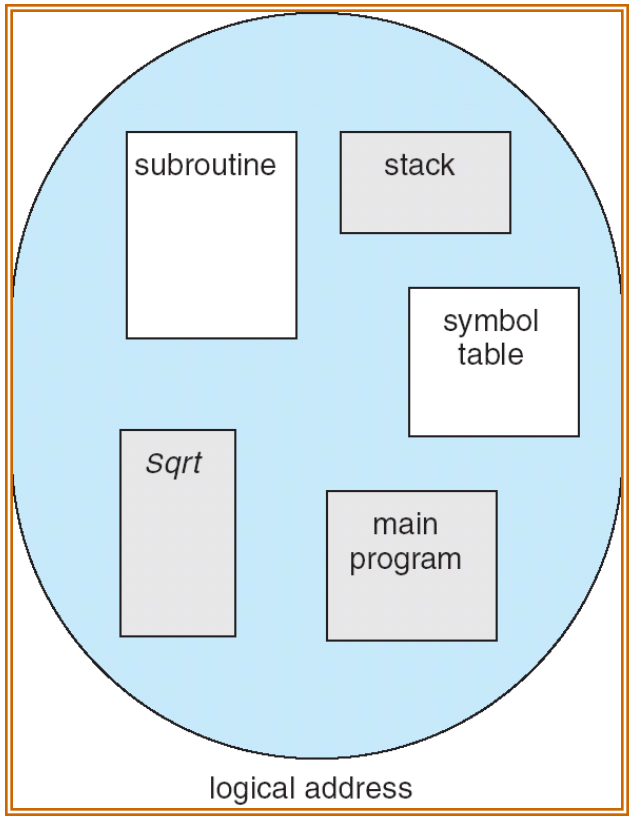
- Memory management scheme that supports user view of memory.
- The entire logical address space is considered as collection of segments with each segment having a number and a length.
- Segment
  - A self-contained unit, which consists of a linear sequence of address from 0 to some maximum.
  - A segment usually does not contain a mixture of different type.

Segmentation (2)

- Segment (ctd.)
  - The length of each segment can vary in size, from 0 to the maximum allowed.
  - Segment lengths may change during execution without affecting each other.
  - A segment is a logical entity, which the programmer is aware of and uses as a single logical entity.

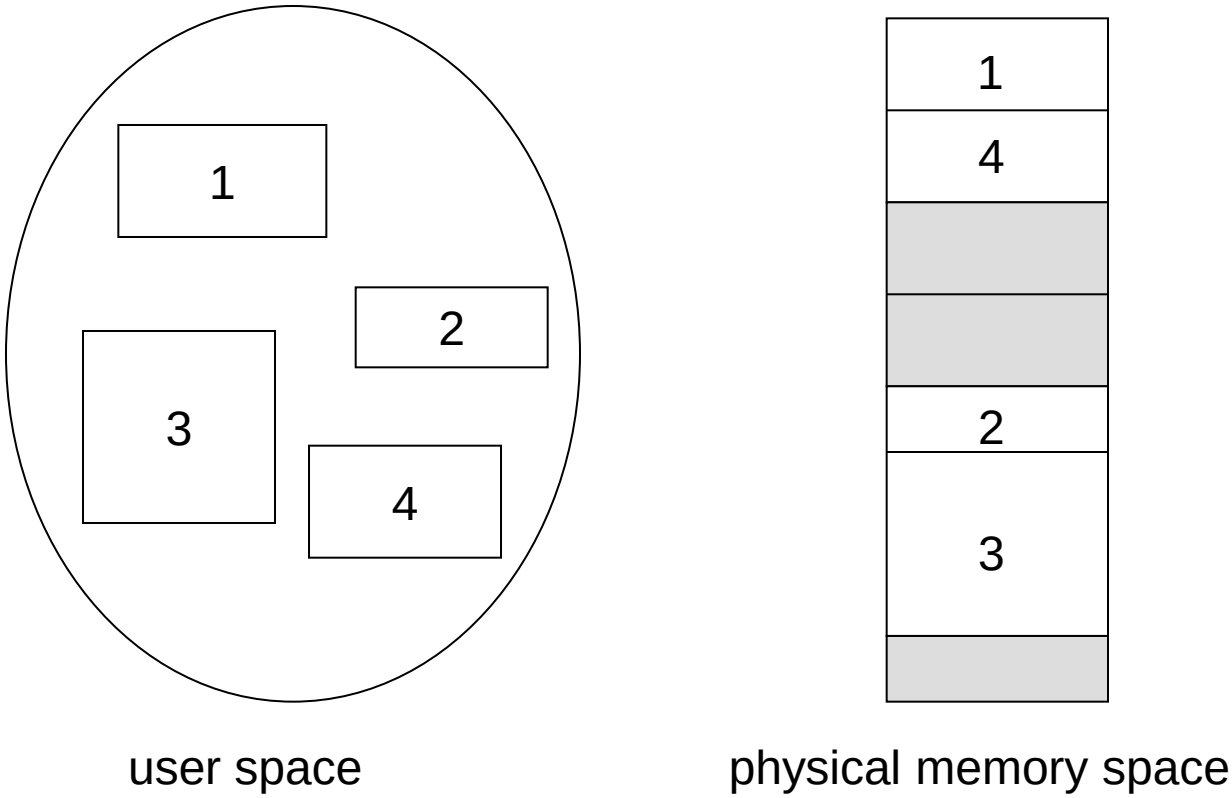
Segmentation (3)

- User's View of A Program
  - A program is a collection of segments. A segment is a logical unit such as:
    - Main program
    - Procedure
    - Function
    - Symbol table
    - Stack



Segmentation (4)

- Logical View of Segmentation



Address Translation Scheme (1)

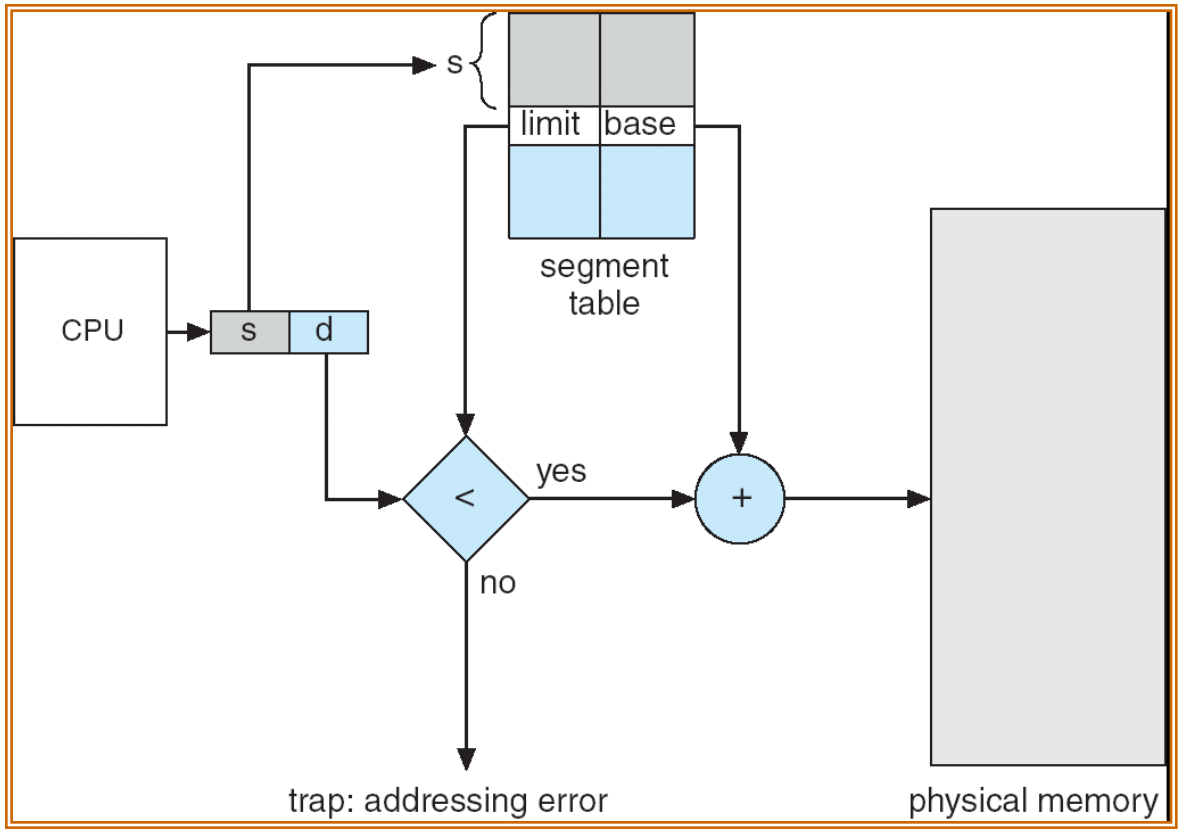
- Logical or Virtual Address
  - Virtual Segment Number (s)
  - Offset (d)
- Segment Table (maintained by OS)
  - Maps two-dimensional physical addresses.
  - The virtual segment number is used as an index to the segment table.
  - Each table entry has:
    - base – contains the starting physical address where the segments reside in memory
    - limit – specifies the length of the segment

Address Translation Scheme (2)

- Address Translation
  - When CPU generates a **logical address**, that address is sent to MMU. The MMU uses the **segment number** of logical address as an **index** to the **segment table**.
  - The **offset** is compared with the **segment limit** and if it is greater, invalid-address **error** is generated.
  - Otherwise, the **offset** is added to the **segment base** to form the **physical address**.

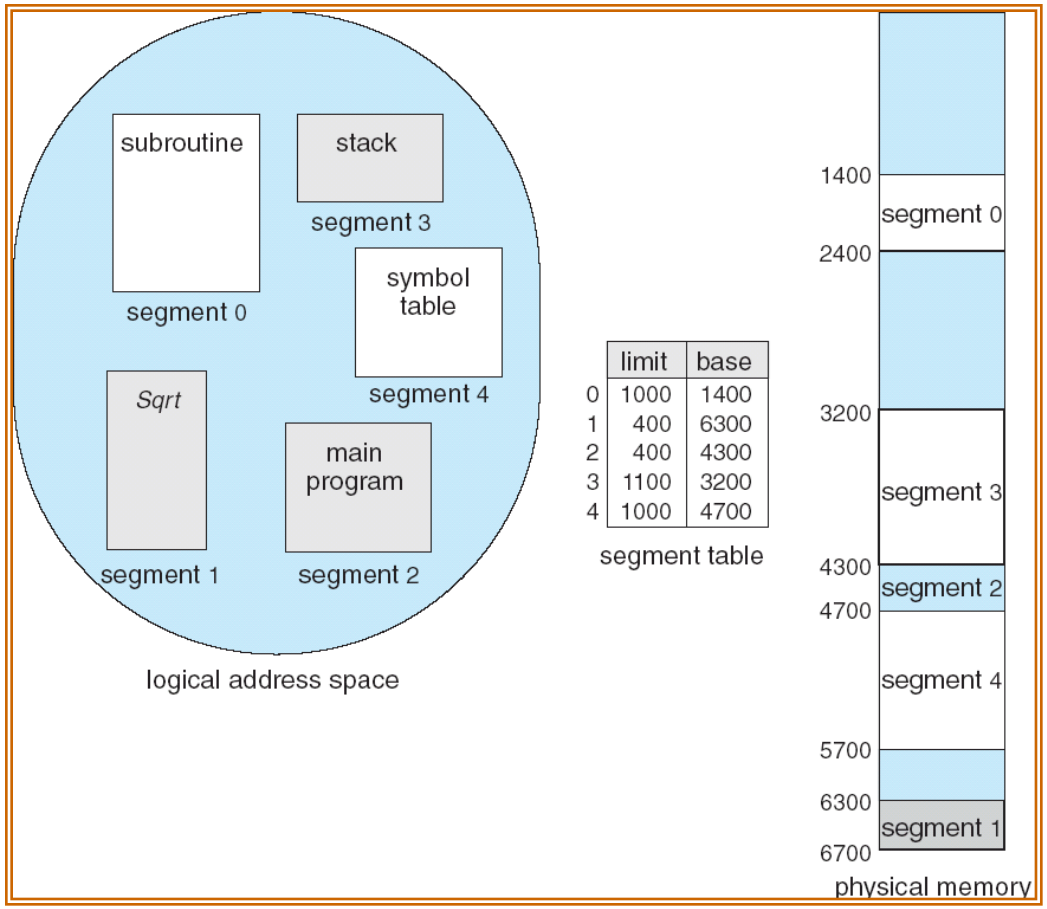
Address Translation Scheme (3)

- Address Translation (ctd.)
  - Figure



Address Translation Scheme (4)

- Example



Address Translation Scheme (5)

- Exercise
  - Using the following segment table, compute the physical address for the logical address consisting of segment and offset as follows:
    - Segment 2 and offset 247
    - Segment 4 and offset 439

	base	limit
0	5432	350
1	115	100
2	2200	780
3	4235	1100
4	1650	400

Segment Table

Address Translation Scheme (6)

- Exercise (ctd.)
  - Solution
    - Segment = 2, Offset = 247
    - From the segment table, limit of segment 2 = 780 and segment base = 2200
    - Since the offset is less than the limit,
    - Physical address = Offset + Segment base = 247 + 2200 = 2447

	base	limit
0	5432	350
1	115	100
2	2200	780
3	4235	1100
4	1650	400

Segment Table

Address Translation Scheme (7)

- Exercise (ctd.)
  - Solution
    - Segment = 4, Offset = 439
    - From the segment table, limit of segment 4 = 400 and segment base = 1650
    - Since the offset is greater than the limit, invalid-address error is generated.

	base	limit
0	5432	350
1	115	100
2	2200	780
3	4235	1100
4	1650	400

Segment Table

Advantages of Segmentation

- May be possible to shrink/grow segments.
- Each segment can be given its own protection information...this is a much easier protection scheme than trying to protect each page of memory.
- Linking programs is a trivial task.
- Code can be shared between processes easily. Just load the code segment once. Copies of the same program access the same segment.

Disadvantages of Segmentation

- Programmer must be aware of the memory model in use (at the assembly level, anyway).
- Just like with swapping systems, fragmentation can waste much memory.
- Segments may be too large to fit in physical memory.

Paging vs. Segmentation

Consideration	Paging	Segmentation
Need the programmer be aware that this technique is being used?	No	Yes
How many linear address spaces are there?	1	Many
Can the total address space exceed the size of physical memory?	Yes	Yes
Can procedures and data be distinguished and separately protected?	No	Yes
Can tables whose size fluctuates be accommodated easily?	No	Yes
Is sharing of procedures between users facilitated?	No	Yes
Why was this technique invented?	To get a large linear address space without having to buy more physical memory	To allow programs and data to be broken up into logically independent address spaces and to aid sharing and protection

Figure 3-32 Comparison of paging and segmentation

Implementation of Pure Segmentation

- Pure segmentation was once implemented by Burroughs in the B5500.

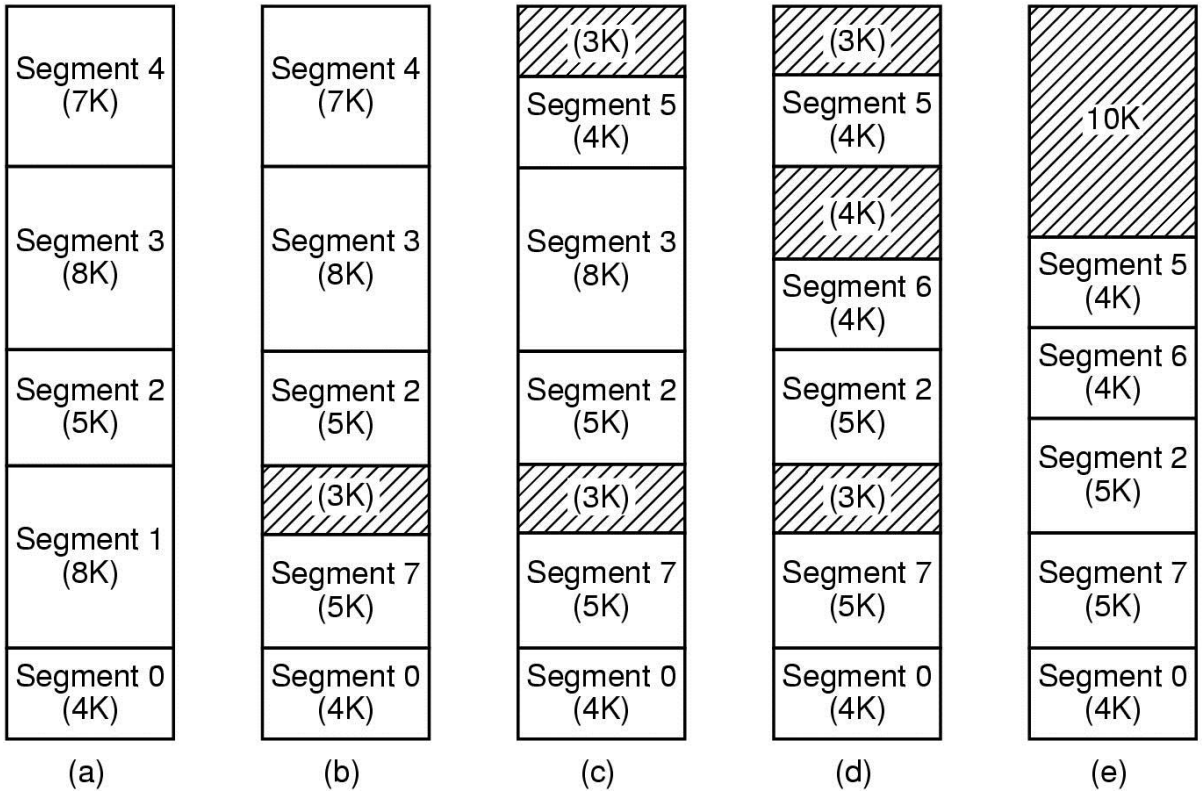
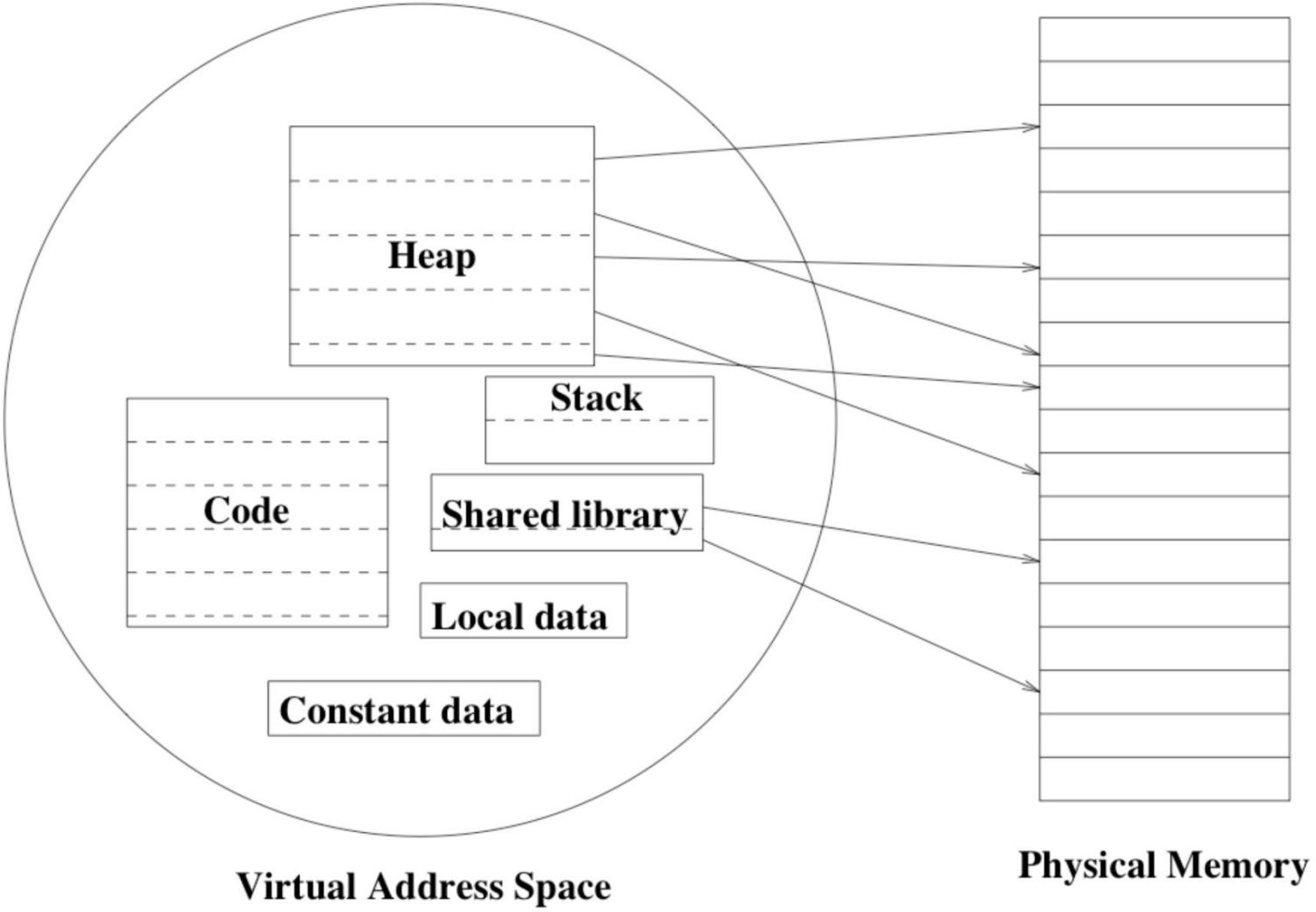


Figure 3-33 (a)-(d) Development of checkerboarding (e) Removal of the checkerboarding by compaction

Segmentation with Paging (1)

- Treat virtual address space as a collection of segments (logical units) of arbitrary sizes.
- Treat physical memory as a sequence of fixed size page frames.
- Segments are typically larger than page frames, ⇒ Map a logical segment onto multiple page frames by paging the segments.
- Physical memory contains only the demanded pages of a segment, not the full segment.

Segmentation with Paging (2)



Segmentation with Paging (3)

- Each process needs a segment table.
  - This table may be segmented and paged itself!
- Each segment is divided into a number of pages.
- To keep track of these pages, a page table is maintained for each segment.
  - Each entry in the segment table points to the page table for that segment.

Address Translation Scheme (1)

- The virtual address has three components

  - Segment Number
    - Used to index the segment table who's entry gives the starting address of the page table for that segment.
  - Page Number
    - Used to index that page table to obtain the corresponding frame number.
  - Offset
    - Used to locate the word within the frame.

Address Translation Scheme (2)

- Segment Table Entry

  - The Segment Base field is the physical address of the page table of that segment.
  - Protection and sharing info most naturally resides in segment table entry.
    - E.g., a read-only/read-write bit, a kernel/user bit...
- Page Table Entry

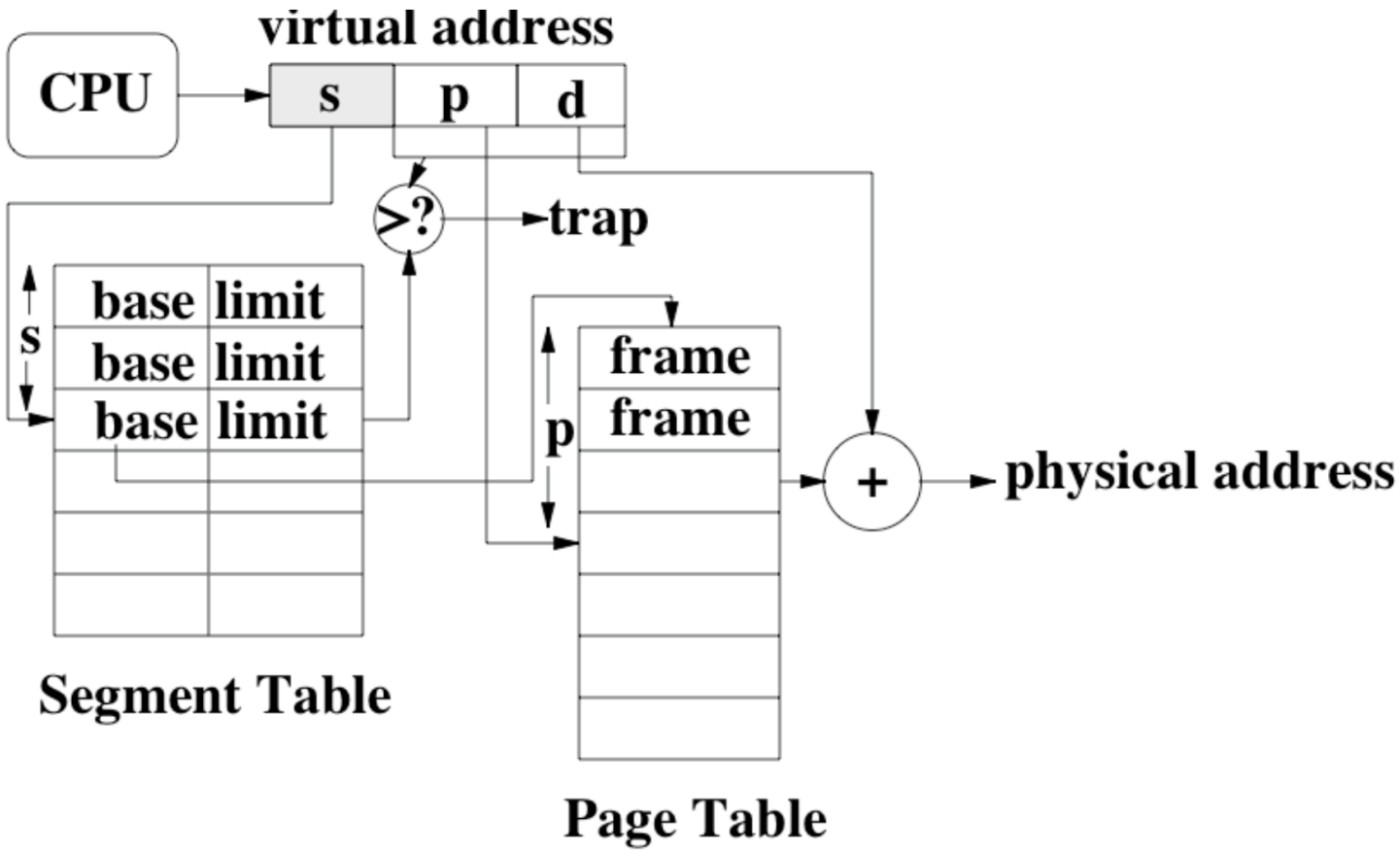
  - Present/modified bits are present only in page table entry.

Address Translation Scheme (3)

- Translation Scheme
  - The segment number indexes into the segment table which yields the base address of the page table for that segment.
  - Check the remainder of the address (page number and offset) against the limit of the segment.
  - Use the page number to index the page table. The entry is the frame. (The rest of this is just like paging.)
  - Add the frame and the offset to get the physical address.

Address Translation Scheme (4)

- Address Translation Figure



Exercise

- Given
  - A memory size of 256 addressable words,
  - A page table indexing 8 pages,
  - A page size of 32 words, and
  - 8 logical segments
- Question:
  - 1. How many bits is a physical address?
  - 2. How many bits is a virtual address?
  - 3. How many bits for the seg #, page #, offset?
  - 4. How many segment table entries do we need?
  - 5. How many page table entries do we need?

Segmentation with Paging: MULTICS (1)

- MULTICS Introduction
  - Multiplexed Information and Computing Service
  - Developed by MIT, GE and Bell Laboratories as the successors to MIT's CTSS (Compatible Time Sharing System).
  - One of the first operating systems to implement virtual memory.
  - A mainframe timesharing operating system begun in 1965 and used until 2000.
  - Sold by honeywell in Honeywell 6000 machines + descendents.
  - MULTICS source was open by MIT.

Segmentation with Paging: MULTICS (2)

- MULTICS Introduction (ctd.)
  - The Multics hardware (GE-645) was word addressable, with 36-bit words.
  - Each virtual address was 34-bits in length.
  - Per program: virtual memory of max. size 2^{18} = 256K segments (max. size 64K 36-bit word long)
    - Each segment of size up to 2^6=64 pages. Each page is of size 2^{10} (36-bit) words.
  - 16-word high speed TLB.

Segmentation with Paging: MULTICS (3)

- MULTICS Virtual Memory
  - Descriptor Segment: Segment table
    - Up to 256K segment descriptors.
  - Segment Descriptor: segment table entry

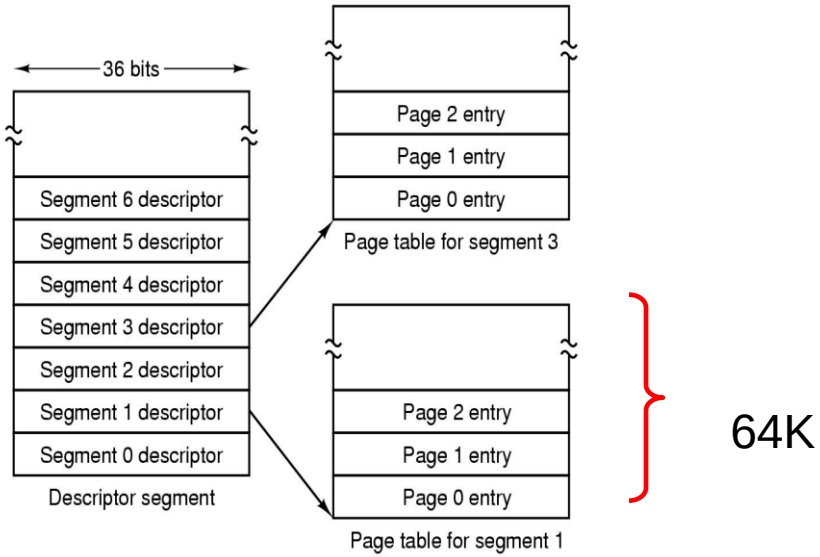


Figure 3-34 (a) Descriptor segment points to page tables

Segmentation with Paging: MULTICS (4)

- MULTICS Virtual Memory (ctd.)
  - Segment Descriptor Structure
    - Because physical addresses were 24-bit and pages were aligned on 64 byte boundaries, only 18 bits for main memory address.
    - The address of the segment in secondary memory was in another table used by the segment fault handler.

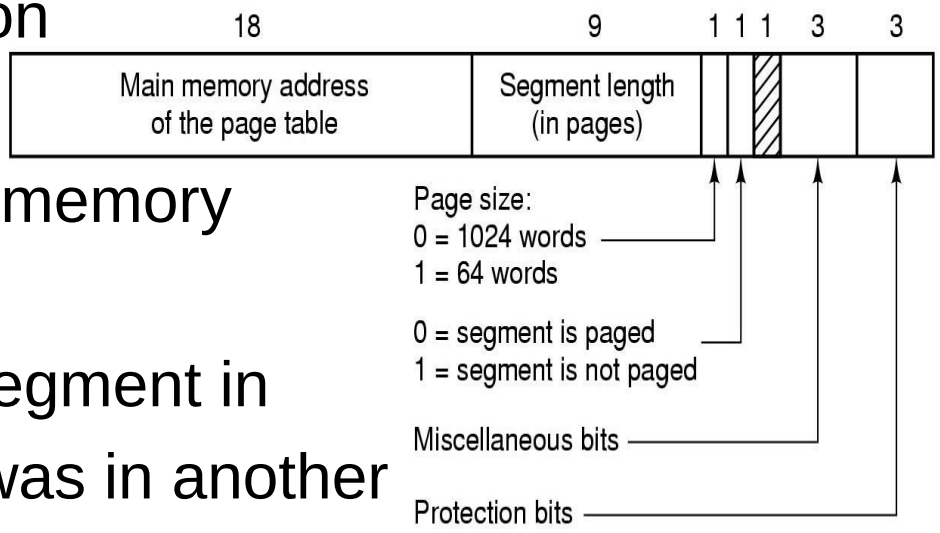


Figure 3-34 (b) Segment descriptor – numbers are field lengths

Segmentation with Paging: MULTICS (5)

- MULTICS Virtual Memory (ctd.)
  - Virtual Address Structure

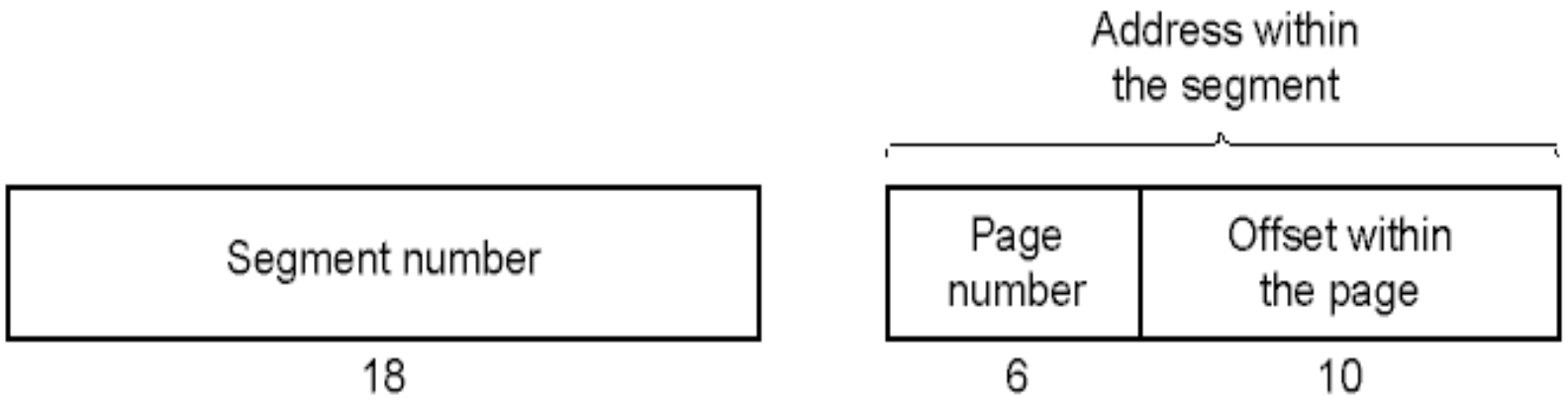


Figure 3-35 A 34-bit MULTICS virtual address

Segmentation with Paging: MULTICS (6)

- MULTICS Virtual Memory (ctd.)
  - Address Translation (Ignoring descriptor segment was paged)

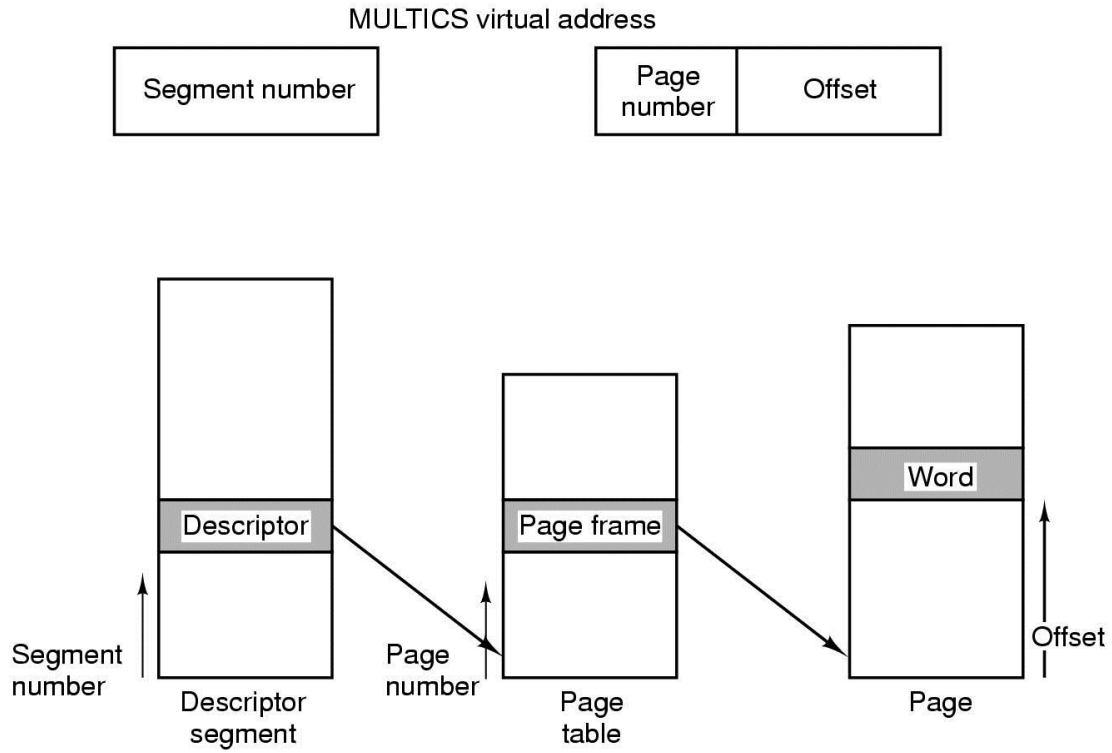


Figure 3-36 Conversion of a 2-part MULTICS address into a main memory address

Segmentation with Paging: MULTICS (7)

- MULTICS Virtual Memory (ctd.)
  - MULTICS TLB
    - The first system to have a TLB.

Comparison field: Segment number	Comparison field: Virtual page	Page frame	Protection	Age	Is this entry used?
4	1	7	Read/write	13	1
6	0	2	Read only	10	1
12	3	1	Read/write	2	1
					0
2	1	0	Execute only	7	1
2	2	12	Execute only	9	1

Figure 3-37 Simplified version of the MULTICS TLB.

Existence of 2 page sizes makes actual TLB more complicated

Segmentation with Paging: The Intel x86 (1)

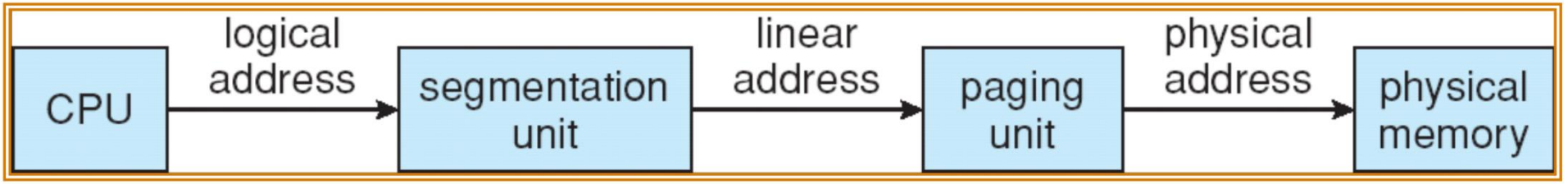
- The Intel x86
  - x86 is a family of backward-compatible instruction set architectures based on the Intel 8086 CPU and its Intel 8088 variant.
  - The 8086 was introduced in 1978 as a fully 16-bit extension of Intel's 8-bit-based 8080 microprocessor, with memory segmentation as a solution for addressing more memory than can be covered by a plain 16-bit address.
  - The term "x86" came into being because the names of several successors to Intel's 8086 processor end in "86", including the 80186, 80286, 80386 and 80486 processors.
  - 1978 (16-bit), 1985 (32-bit), 2003 (64-bit)
  - Support for segmentation removed in x86-64.

Segmentation with Paging: The Intel x86 (2)

- Each program could have up to 16K segments, each segment could be up to 1 billion(32-bit) words long.
- Each program has its own LDT.
- Local Descriptor Table (LDT)
  - Describes segments local to each program, including its code, data, stack, …
- A single GDT shared by all programs on the computer.
- GDT (Global Descriptor Table)
  - Describes system segments, including the OS itself.

Segmentation with Paging: The Intel x86 (3)

- x86 Address Translation
  - CPU generates logical address (segment selector, offset)
  - Gives it to segmentation unit, which produces linear addresses.
  - Linear address given to paging unit, which generates physical address in main memory.



Segmentation with Paging: The Intel x86 (4)

- x86 Segment Selector
  - Logical address: segment selector + offset
  - Segment selector stored in segment registers (or selector registers) (16-bit)
    - cs: code segment selector
    - ss: stack segment selector
    - ds: data segment selector
    - es, fs, gs: for OS to use freely
  - To access a segment, a x86 program first loads a selector for that segment into one of the machine's 6 segment register.

Segmentation with Paging: The Intel x86 (5)

- x86 Segment Selector (ctd.)
  - Consists three fields
    - Index Field: Specify LDT or GDT entry number. Theses tables are restricted to hold 8K segment descriptors.

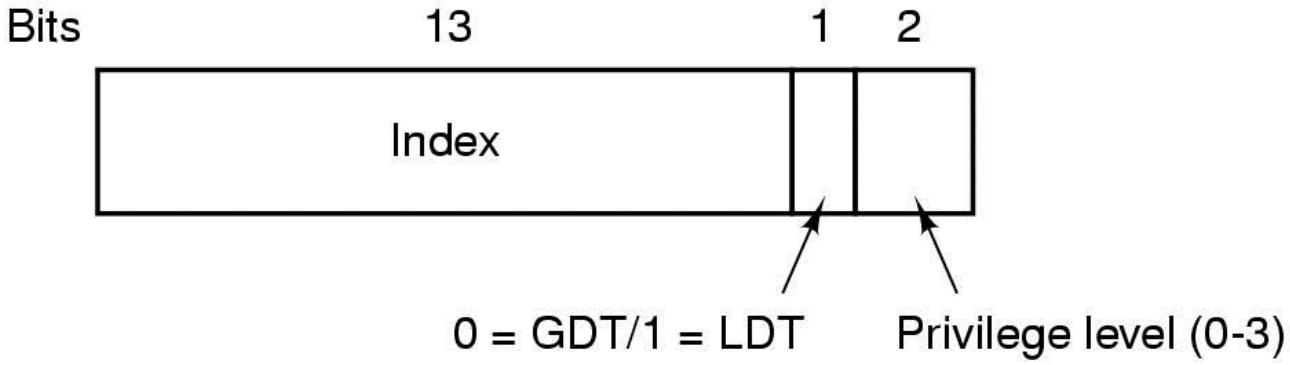


Figure 3-38 A x86 selector

Segmentation with Paging: The Intel x86 (6)

- Segment Descriptor
  - Actual attributes of a segment are contained in the segment descriptor.
  - Segment descriptors are 64 bits (8 bytes in size).
  - Reside in Global or Local descriptor tables of up to 8192 entries.
  - Entry 0 always represents an invalid segment.

Segmentation with Paging: The Intel x86 (7)

- Segment Descriptor (ctd.)
  - G Field: granularity flag. G=0, Limit: 1B～1MB. G=1, Limit: 4KB～4GB
  - Base Field: for compatibility with 286, broken into 3 pieces.

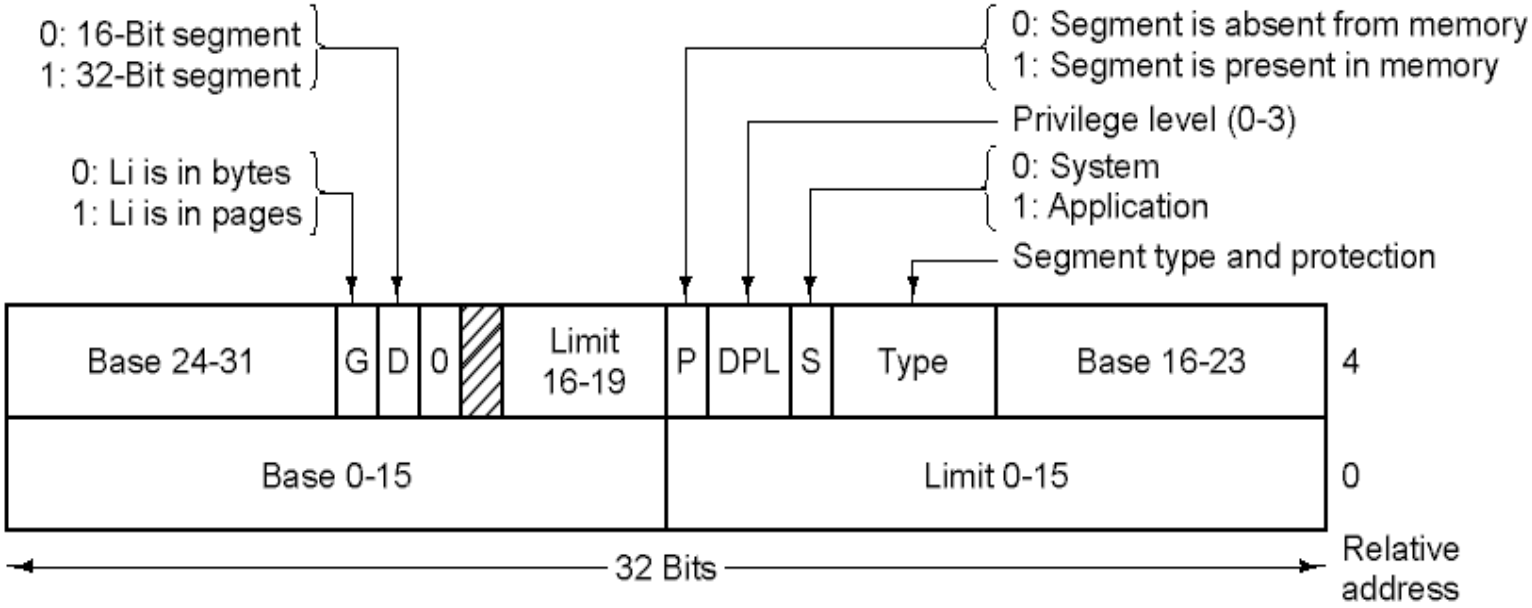


Figure 3-39 x86 code segment descriptor (Data segments differ slightly)

Segmentation with Paging: The Intel x86 (8)

- Translation from Logical Address to Linear Address

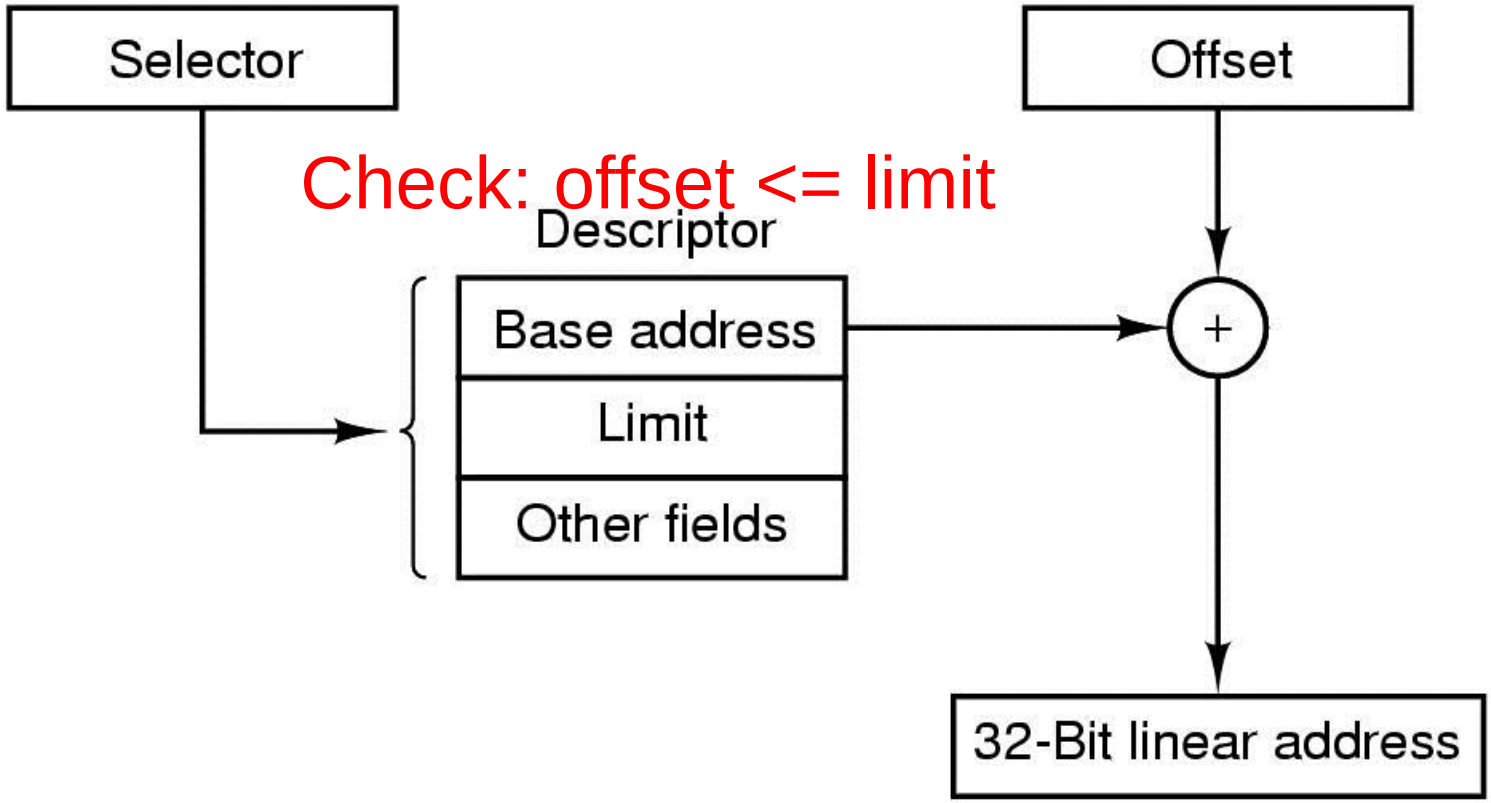


Figure 3-40 Conversion of a (selector, offset) pair to a linear address

Segmentation with Paging: The Intel x86 (8)

- Translation from Logical Address to Linear Address (ctd.)
  - If pure segmentation, the linear address is interpreted as the physical address.
  - If paging enabled, the linear address is interpreted as a virtual address.
    - Need mapping to physical address.

Segmentation with Paging: The Intel x86 (9)

- Mapping from Linear Address to Physical Address
  - 32 bit virtual address, 4KB page size
  - Two-level paging
  - Linear Address Structure

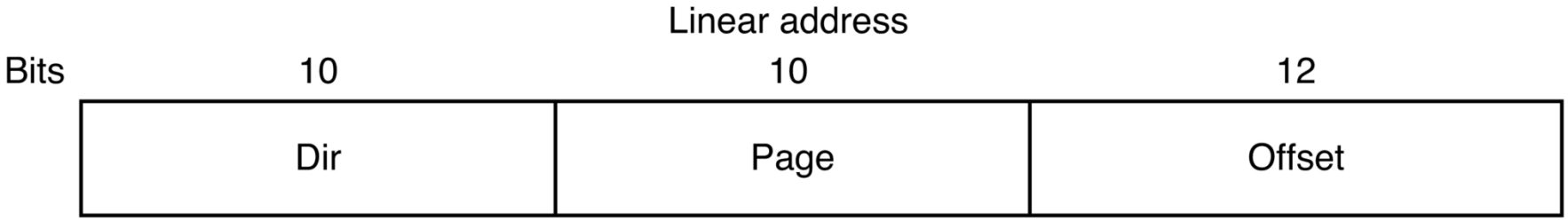


Figure 3-41 (a)

Segmentation with Paging: The Intel x86 (10)

- Mapping from Linear Address to Physical Address (ctd.)
  - Each running program has a page directory consisting of 1K 32-bit entries.
  - Each entry in this directory points to a table also containing 1K 32-bit entries.

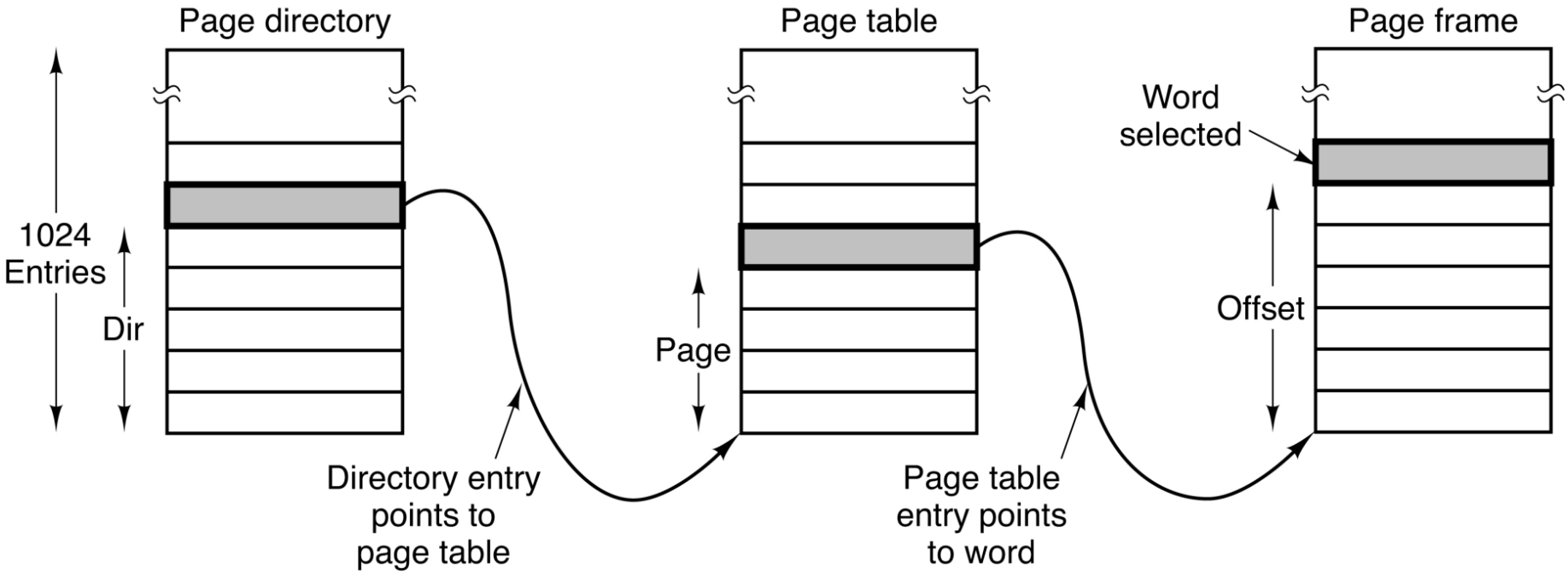
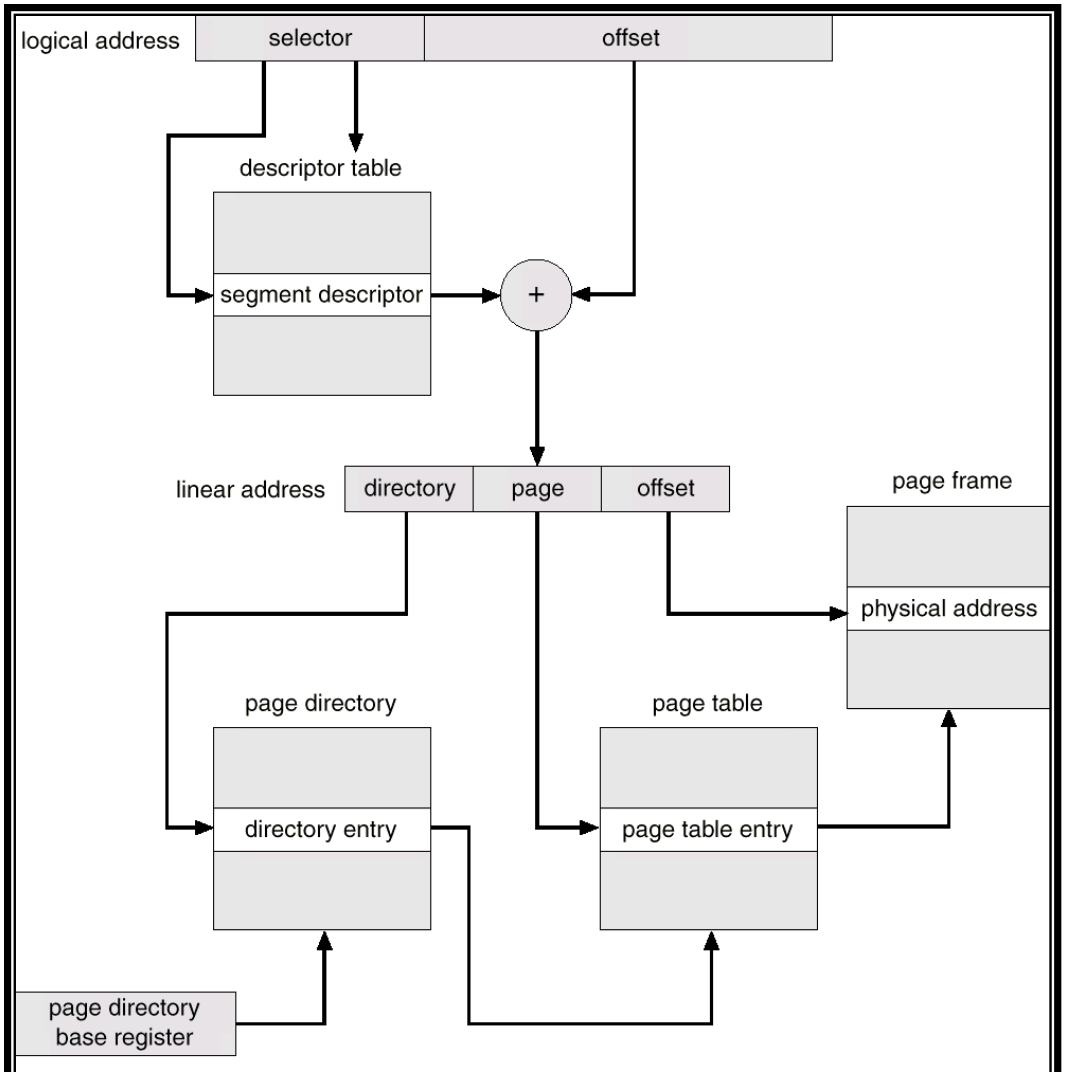


Figure 3-41 (b) Mapping of a linear address onto a physical address

Segmentation with Paging: The Intel x86 (11)

- From Logical Address to Physical Address



Homework

- P259: 45, 47