Operating Systems

Lecture 6.3

Non-Contiguous Memory Allocation Schemes - II

Instructor: Muhammad Arif Butt, PhD

Lecture Agenda

- Paged Segmentation
  - What is Paged Segmentation?
  - How Paged Segmentation work?
  - LA to PA translation in Paged Segmentation
  - Example Problems
- Address Translation in Intel family of Processors
  - Intel 8080
  - Intel 8086
  - Intel 80386
  - Intel/AMD x86-64

Paged Segmentation

Paged Segmentation Example

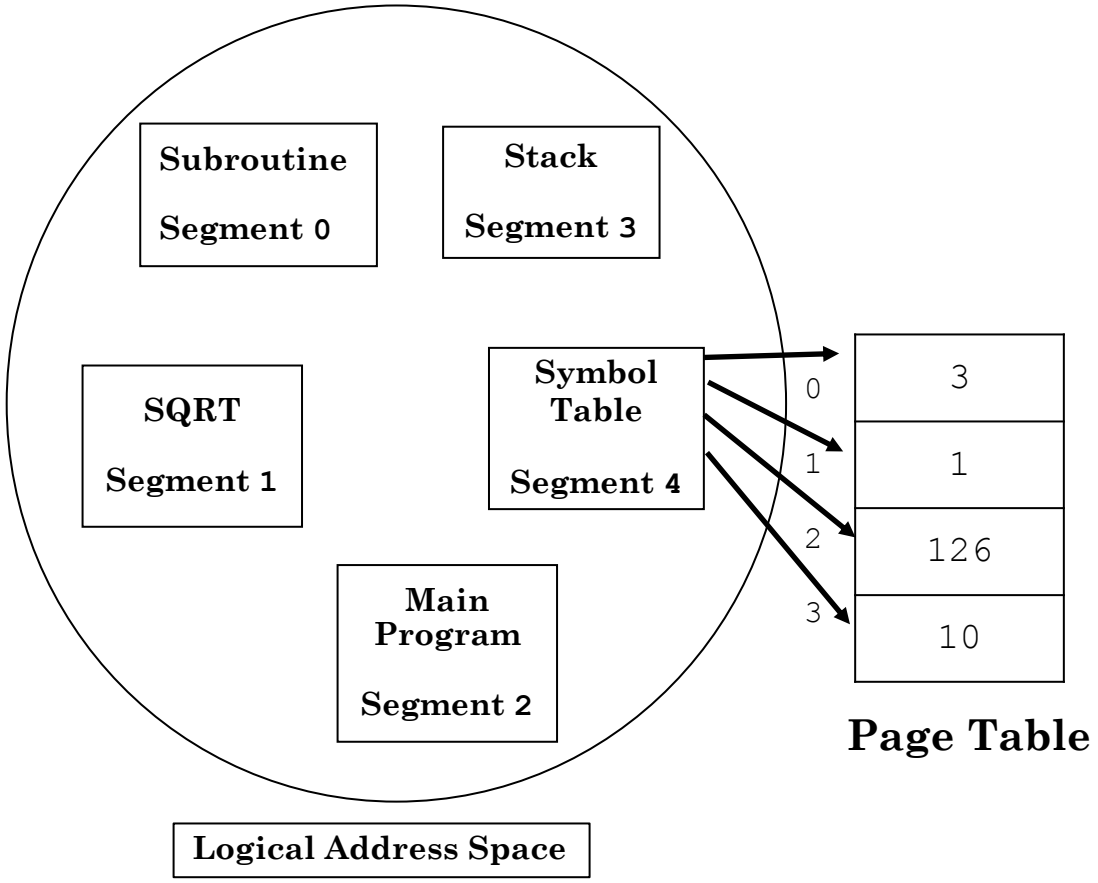
Problem: Large segment sizes increase the chance of External Fragmentation.

Solution:

- Each segment is further divided into pages.
- Each segment maintains its own Page Table.
- Segment Table → Points to the Page Table of the segment.
- Page Table → Maps segment pages to physical memory frames.

Advantages:

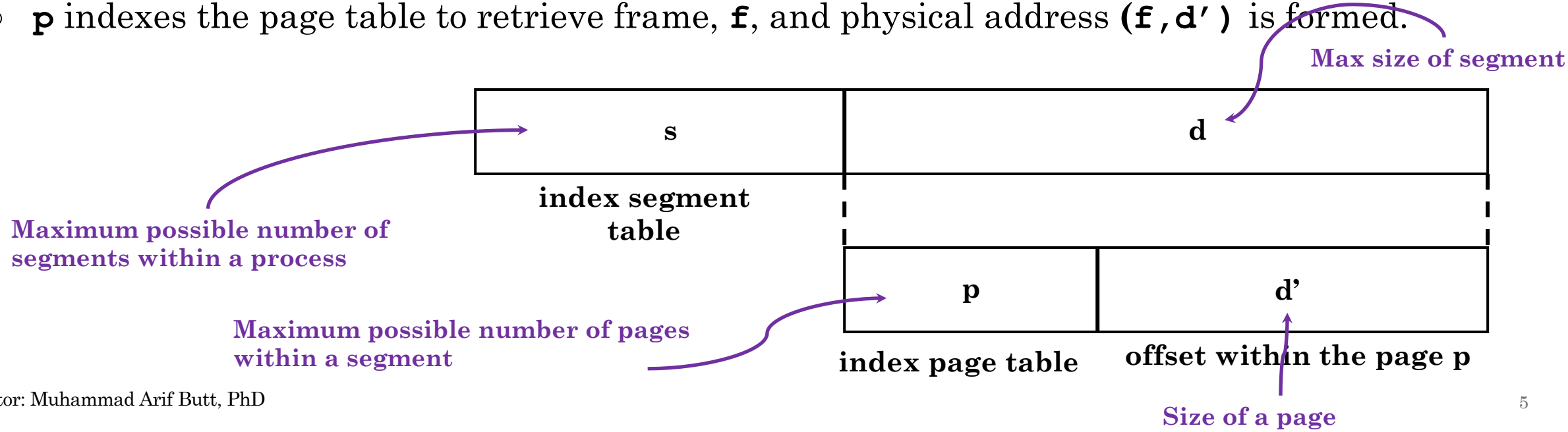
- Eliminates External Fragmentation.
- Provides benefits of both Segmentation (logical division, protection) and Paging (efficient use of memory).



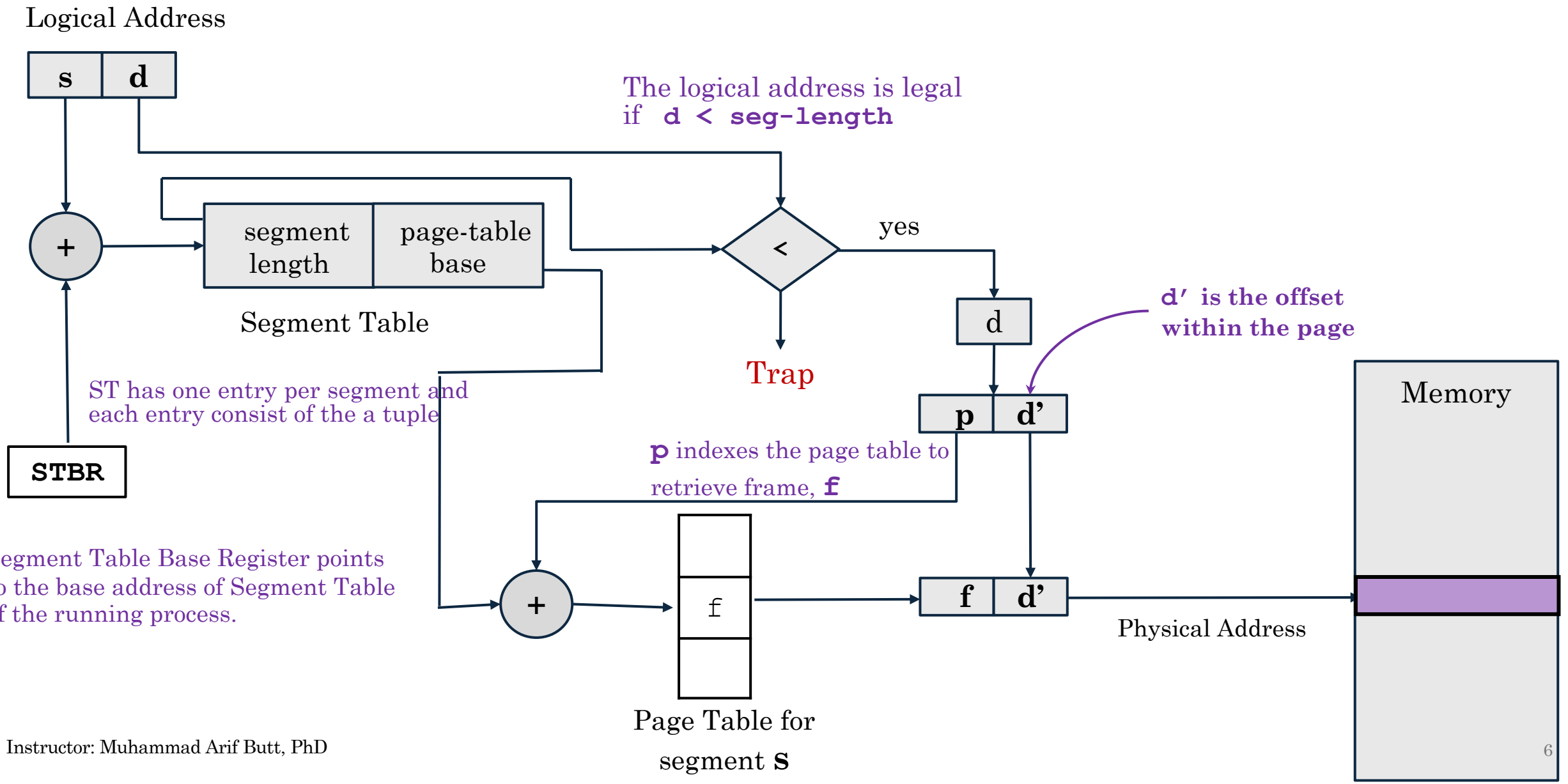
Here we have divided segment 4 into four pages and kept that information in page table.

Address Translation: GE-645

- The **GE 645** computer was designed in 1964 and implemented a ground breaking two-level address translation scheme that combined segmentation and paging.
- **MULTICS** (Multiplexed Information and Computing Service) was the first OS to implement paged segmentation.
- Logical address is still **<s,d>**, with **s** used to index the segment table.
- Segment offset, **d**, is partitioned into two parts: **p** and **d′**
  - **p** is used to index the page table associated with segment **s**
  - **d′** is used as offset within a page.
  - **p** indexes the page table to retrieve frame, **f**, and physical address **(f,d′)** is formed.



Address Translation: GE-645



Sample Problems

Problem 1 Consider MULTICS on a GE 645 processor, with logical address of 34 bits and page size of 1 KB. s is of 18 bits and d is of 16 bits. Answer the following questions:

- What is the largest segment size?
- What is the maximum number of segments per process?
- Give maximum number of pages per segment.
- Give the LA format including the no of bits for p and d'.

Sample Problems (Cont.)

Problem 2 Consider a process in MULTICS with its segment#15 having 5096 bytes. The page size is 1KB. The process generates a Logical Address of (15, 3921).

- Is it a legal address? If yes why?
- How many pages does the segment have?
- What page does the logical address refer to, and what is its offset?
- What is the P.A if page#3 (i.e. fourth page) is in frame 12?

Sample Problems (Cont.)

Problem 3 Consider the given segment table. How many page tables will be constructed for the process whose segments are shown in the segment table?

Segment #	Length	Base
0	100	12000
1	1200	12100
2	190	13300
3	444	15500
4	19308	18008
5	3400	5000

Problem 4 Consider the given segment table. If the system implements paged-segmentation with the page size of 2 KB, then compute the page number and offset for the logical address of (4,12765). Also compute the number of pages in segment 4 and the address of the 3rd entry in the page table (Assume PTES is 4 Bytes)

Problem 5 Consider the given segment table, if segment table base register (STBR) contains 36500 and segment table entry size (STES) is 64 bits then what will be size of segment table? Also compute the address of the last entry?

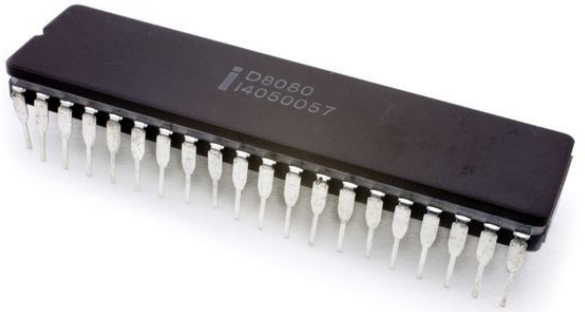
Sample Problems (Cont.)

Problem 6 Consider a logical address in paged segmentation `(16, 7865)`, where `16` is segment number and `7865` is offset. Suppose segment `16` has length of `15690` bytes. Page size is `1 KB`. Calculate following:

- Number of pages in segment number `16`
- In which page the above said offset will reside?
- How would you represent the above address in `<s, p, d'>` format?
- Let the page p is stored at frame `30`, what would be the physical address?

Address Translation

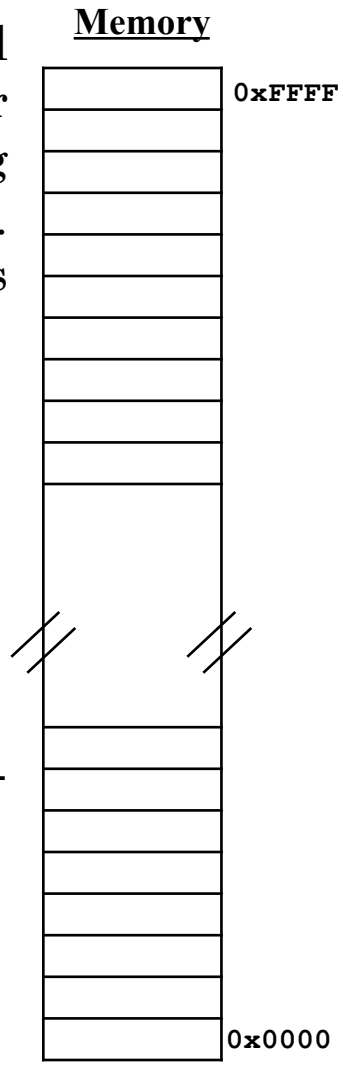
Intel 8080



Intel 8080 (Real Mode Only)

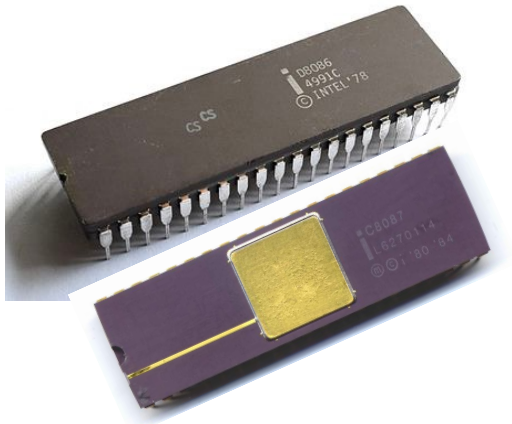
- In 1974, Intel introduced its 8-bit Intel-8080 CPU with an address bus of 16 bits. The Intel 8080 represents the pre-virtual memory era where address translation wasn't necessary or implemented. Every memory reference was a direct physical memory access, making programming simpler but limiting memory management capabilities and system protection. This direct approach was adequate for the single-user, single-tasking systems of the mid-1970s but became insufficient as computing needs evolved.
- The Intel 8080 uses a flat, linear addressing model where:
  - Logical addresses = Physical addresses
  - No virtual memory concept
  - No memory management unit (MMU)
  - Direct hardware memory access
- The addressing is simple, you put a 16-bit address on the address bus and you get back the 8-bit value that is stored at that address
- **Address Space Characteristics:**

```
Address Bus: 16-bit
Maximum Memory: 64KB (2^16 = 65,536 bytes)
Address Range: 0x0000 – 0xFFFF
Logical Address 0x1234 → Physical Address 0x1234 (direct mapping)
```



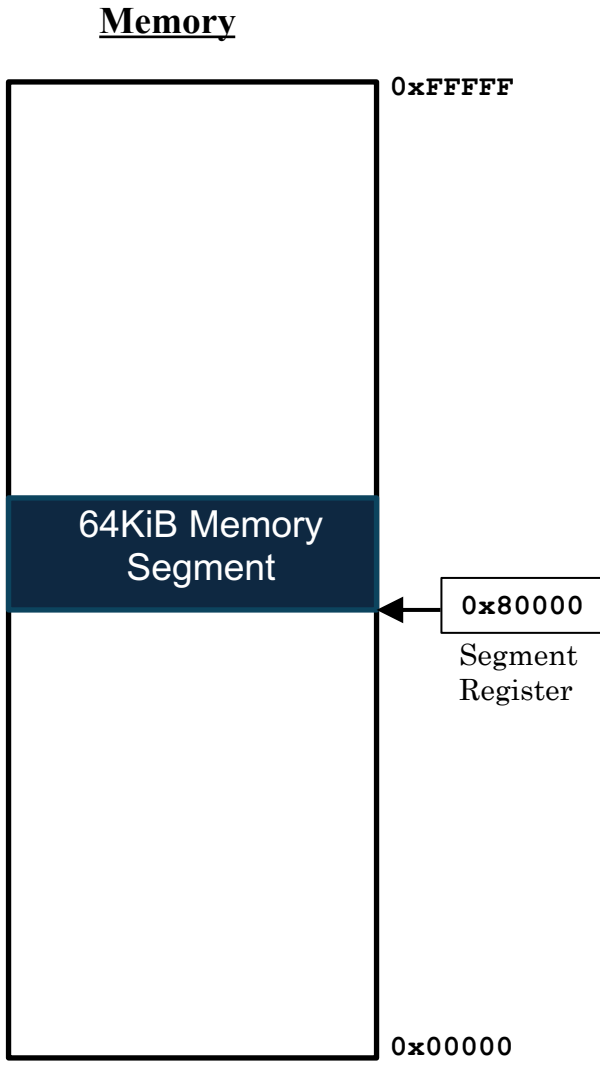
Address Translation

Intel 8086

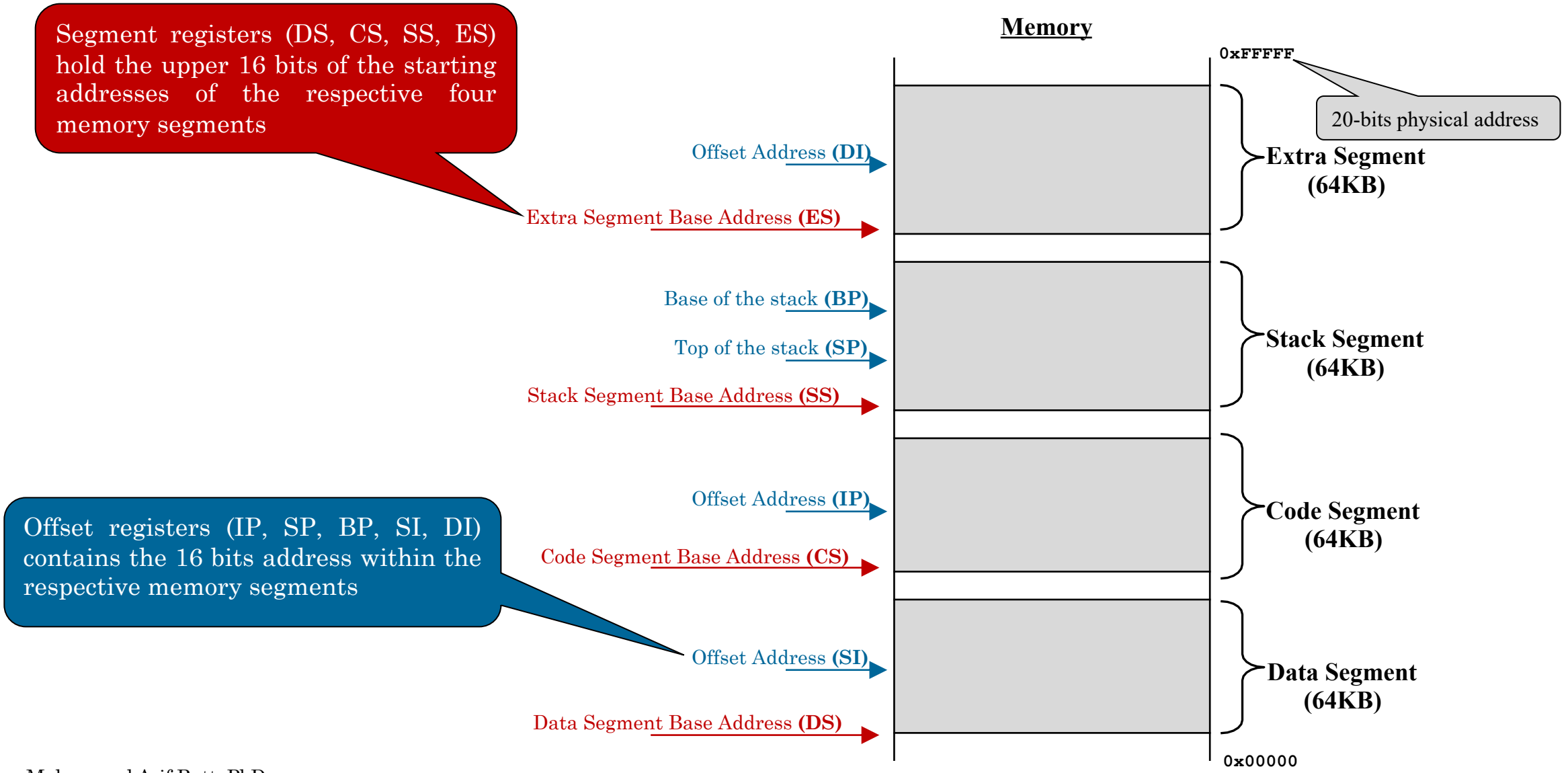


Memory Model of Intel 8086

- In 1978, Intel introduced its 16-bit Intel 8086 CPU, with an address bus of 20 bits, along with a separate 8087 Floating Point Unit (Math co-processor).
- Intel-8086 CPU has an addressable memory of 1 MiB, which is 16 times more than Intel 8080.
- Intel wanted to port all assembly programs running on 8080 to run on 8086 as well. To make this porting possible, the designers of 8086 divided its memory in 64 KiB segments, so that a 8080 program could be loaded into a 64 KiB memory segment and can execute successfully.
- Intel 8086 memory model is known as **Segmented memory model**, which divides the memory into groups of independent segments referenced by pointers located in special CPU registers called segment registers. Code, data and stack can appear as three distinct units in memory.
- **Characteristics:**
  - Maximum addressable memory: 1MB (20-bit addresses).
  - No memory protection.
  - Direct hardware mapping.
  - No virtual memory support.



Segmented Memory of Intel 8086



Address Translation: Intel 8086

- In Intel 8086, Logical addresses ≠ Physical addresses (translation required)
- The architecture use pure segmentation, no paging.
- **Translation Process:** You provide a 16-bit segment selector and 16-bit offset (forming a logical address), which the CPU translates into a 20-bit physical address using the following formula, allowing addressing up to 1MiB of physical memory despite using 16-bit registers.

```
Logical Address: [Segment:Offset] (16-bit each)

Physical Address = (Segment × 16) + Offset
```

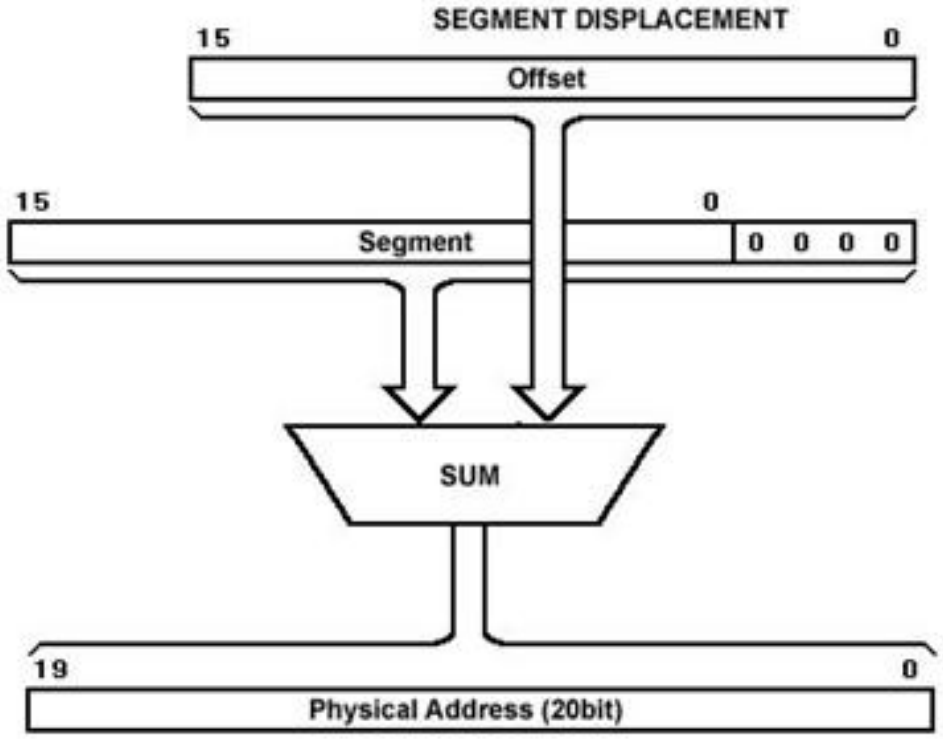
- **Example:** Suppose the CS register contains `0x3F2A`. Translate the 16-bit logical address `0x1B08` to the corresponding 20-bit physical address for Intel 8086 processor.

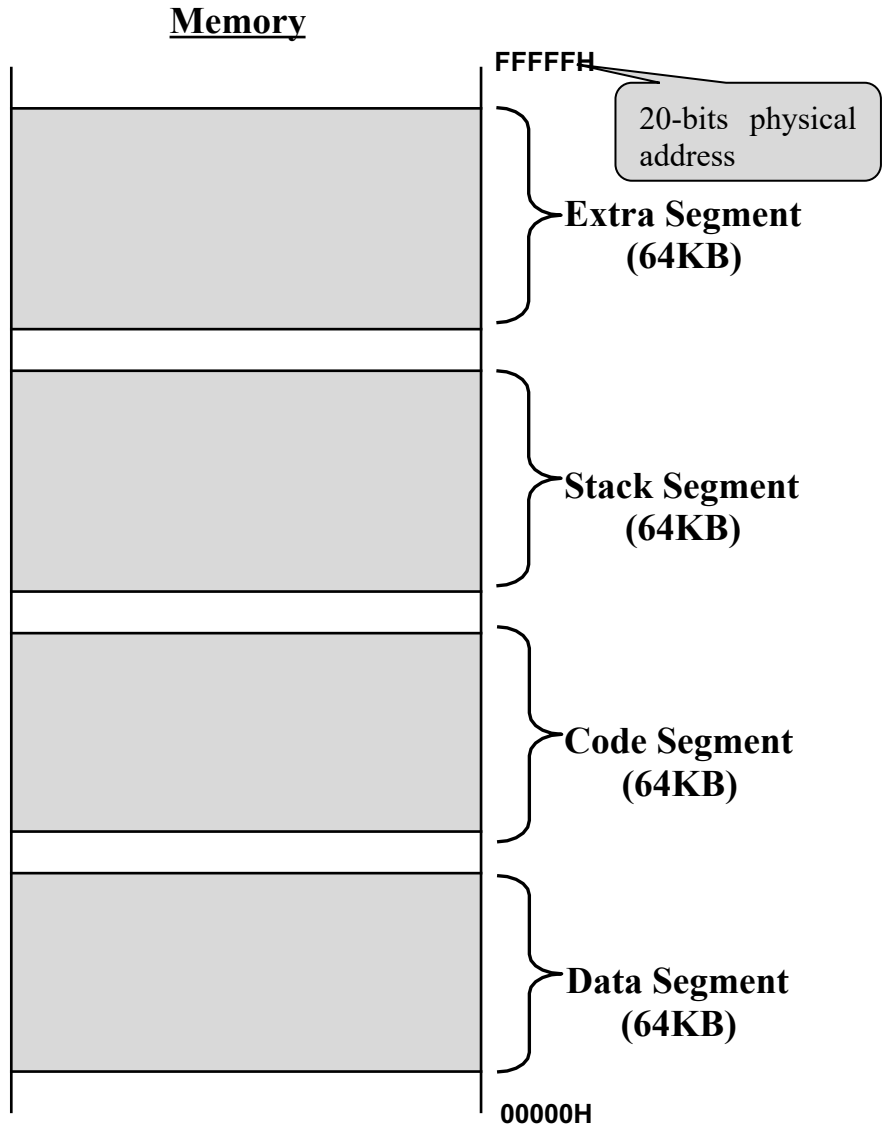
```
CS = 0x3F2A
IP = 0x1B08
CS:IP = 3F2A:1B08
P.A = CS * 10₁₆ + IP
P.A = 3F2A * 10 + 1B08
P.A = 3F2A0 + 1B08 = 40DA8
```

Address Translation Intel 8086 (cont...)

Example:

```
CS = 0x3F2A
IP = 0x1B08
CS:IP = 3F2A:1B08
P.A = CS * 10₁₆ + IP
P.A = 3F2A * 10 + 1B08
P.A = 3F2A0 + 1B08 = 40DA8
```





Address Translation

Intel 80386

Memory Model of Intel 80386

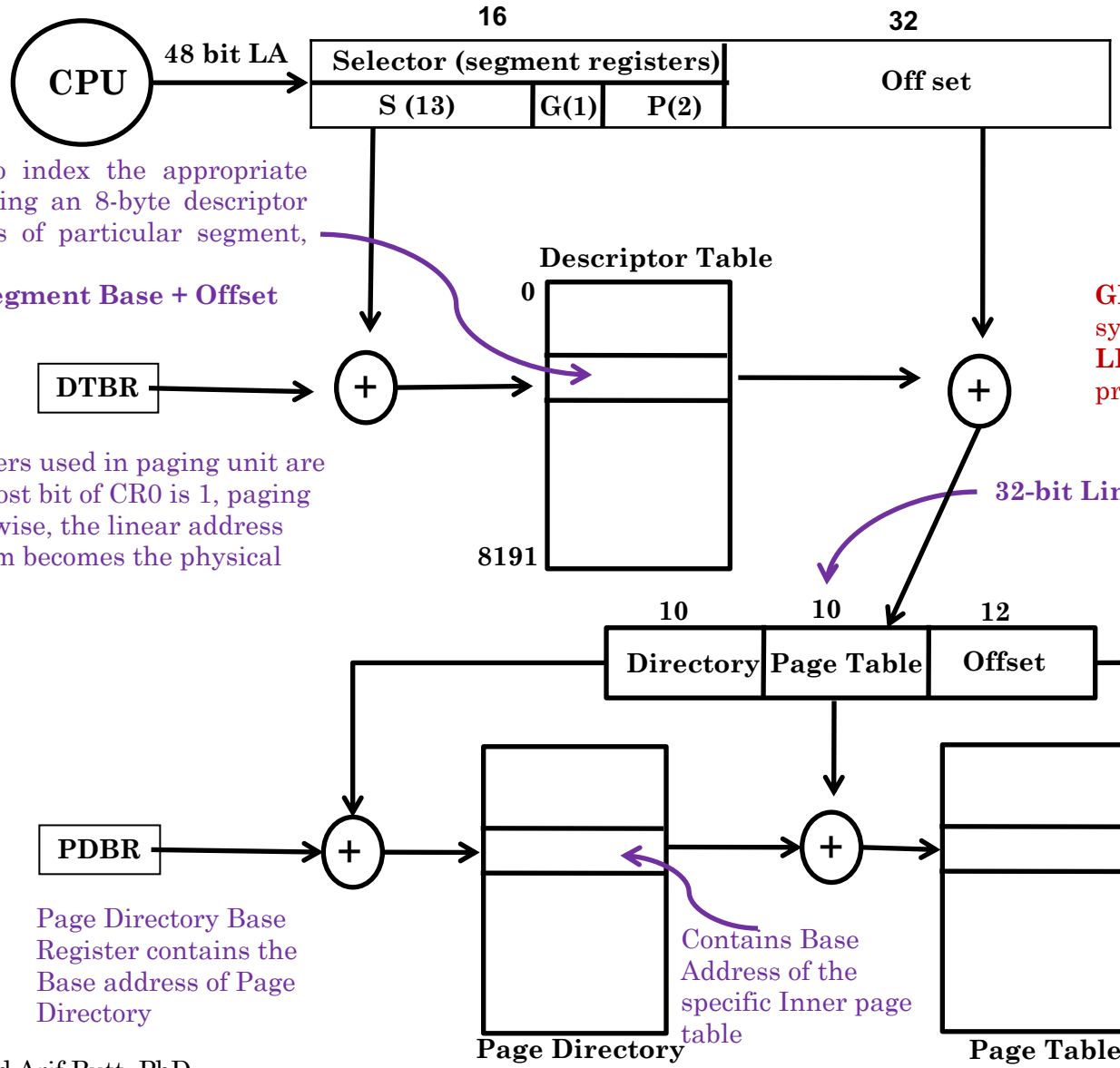
In 1985, Intel introduced its 32-bit Intel 80386 CPU, with 32-bit address bus that could address 4 GiB of memory. Other than the real mode (for compatibility), it also support protected mode.

Real Mode: In real mode it uses simple segment:offset addressing (8086-compatible) without protection. Maximum addressable memory is 1MiB, and the memory allocation appear contiguous to applications.

Protected Mode (80386 Enhanced): This new mode of 80386 allows access to data and programs located above the first 1 MiB of memory (extended memory), as well as within the first 1 MiB of memory. The segment registers are now considered part of the operating system, you can neither read nor change them directly. They point to OS data structures that contain information to access a location. It revolutionized memory management and has a support of following memory managemtn modes:

- Pure Segmentation: Uses only segment descriptors without paging.
- Pure Paging: Flat memory model with paging (segments cover entire 4GB space)
- **Paged Segmentation:** Full two-stage translation: 48-bit LA to 32-bit PA (segment→linear→physical)
- Backward Compatibility: Maintains 80286 and 8086 compatibility modes

Address Translation: Intel 80386



The selector is used to index the appropriate descriptor table, retrieving an 8-byte descriptor containing base address of particular segment, limit, and access rights.

Linear Address = Segment Base + Offset

Program invisible registers used in paging unit are CR0 to CR3. If the leftmost bit of CR0 is 1, paging mechanism works otherwise, the linear address generated by the program becomes the physical address

Page Directory Base Register contains the Base address of Page Directory

Contains Base Address of the specific Inner page table

Inner page table contains the Frame # which is appended with offset to get the P.A

Structure of a Selector stored in Segment Registers:

S(13) used to index into a Descriptor Table

G(1) specifies Table Indicator: 0 means GDT and 1 means LDT

P(2) define the privilege level for access or rings of protection

- 00 – Private OS functions
- 01 – OSS services
- 10 – device drivers
- 11 – Application programs

GDT (Global Descriptor Table) is one per system, that describes system-wide segments (e.g., OS kernel, shared libraries, device drivers).

LDT (Local Descriptor Table) is one per process, that describes process-private segments (code, data, stack).

Address Translation

Intel/AMD x86-64

Memory Model of Intel/AMD x86-64

In 2003, AMD introduced the x86-64 architecture (later adopted by Intel as Intel 64), featuring 64-bit processing capabilities with a *64-bit address bus that could theoretically address 16 EiB of memory*. Intel Core i7 supports a 48-bit (256 TiB) virtual address space, and 52-bit (4 PiB) physical address space. You can check your machine's virtual & physical address sizes in the `/proc/cpuinfo` file.

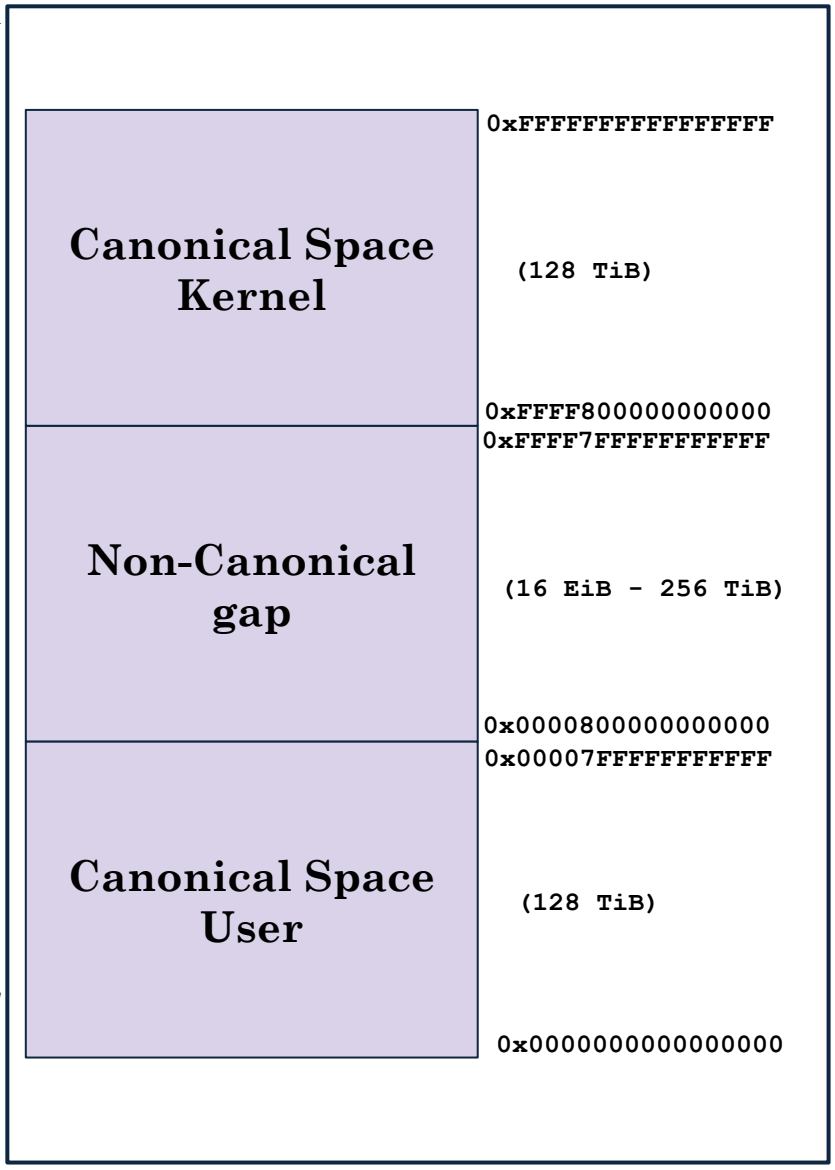
Real Mode: Maintains complete 8086 compatibility using simple segment:offset addressing without protection. Maximum addressable memory remains 1MiB, and memory allocation appears contiguous to applications. This mode is primarily used during system boot, before transitioning to other modes.

Protected Mode (Legacy 32-bit): Fully compatible with 80386 protected mode, supporting all previous memory management models including segmentation and paging. This mode allows 32-bit applications to run unchanged on 64-bit processors while maintaining the 4GB address space limitation.

Long Mode (x86-64 Enhanced): Long mode is the mode where a 64-bit OS can access 64-bit instructions and registers. 64-bit programs are run in a sub-mode called 64-bit mode, while 32-bit programs and 16-bit protected mode programs are executed in a sub-mode called compatibility mode.

Logical Address Space of Intel/AMD x86-64

- The x86-64 CPU chips that you can buy today implement 48 bit logical address for virtual memory (as shown), and 40 bits for physical memory.
- To keep things consistent, the CPU enforces canonical addresses:
  - Bit 47 decides the sign.
  - Bits 63:48 must be the same as bit 47 (→ sign extension).
  - If this rule is broken → address is non-canonical (invalid).
- This splits the logical address space into three regions:
  - **User space** starts from `0x0000000000000000` to `0x00007FFFFFFFFFFF`, providing 128 TiB of addressable memory. This region is accessible to user-mode processes and is where the code, data, heap, and stack reside.
  - **Kernel space** starts from `0xFFFF800000000000` to `0xFFFFFFFFFFFFFFFF`, spanning 128 TiB. It is exclusively reserved for the operating system kernel, its subsystems, device drivers, and essential kernel data structures. Only code running in privileged mode (ring 0) can access this region. Attempts to access it from user mode result in a protection fault, ensuring kernel integrity and isolation.
  - **The non-canonical space**, ranging from `0x0008000000000000` to `0xFFFF7FFFFFFFFFFF`, lies between the user and kernel address ranges (16 EiB minus 256 TiB).
- The non-canonical gap is reserved so that when features like 5-level paging (57-bit virtual addresses) are introduced in newer Intel/AMD CPUs, portions of that gap can be reclaimed and made usable. You can check your machine's virtual & physical address sizes in the `/proc/cpuinfo` file:



Logical Address of Intel/AMD x86-64

By default the x86-64 uses 48-bits for virtual address and uses a four-level paging structure, commonly called Page Map Levels (**PML4→PML3→PML2→PML1)**

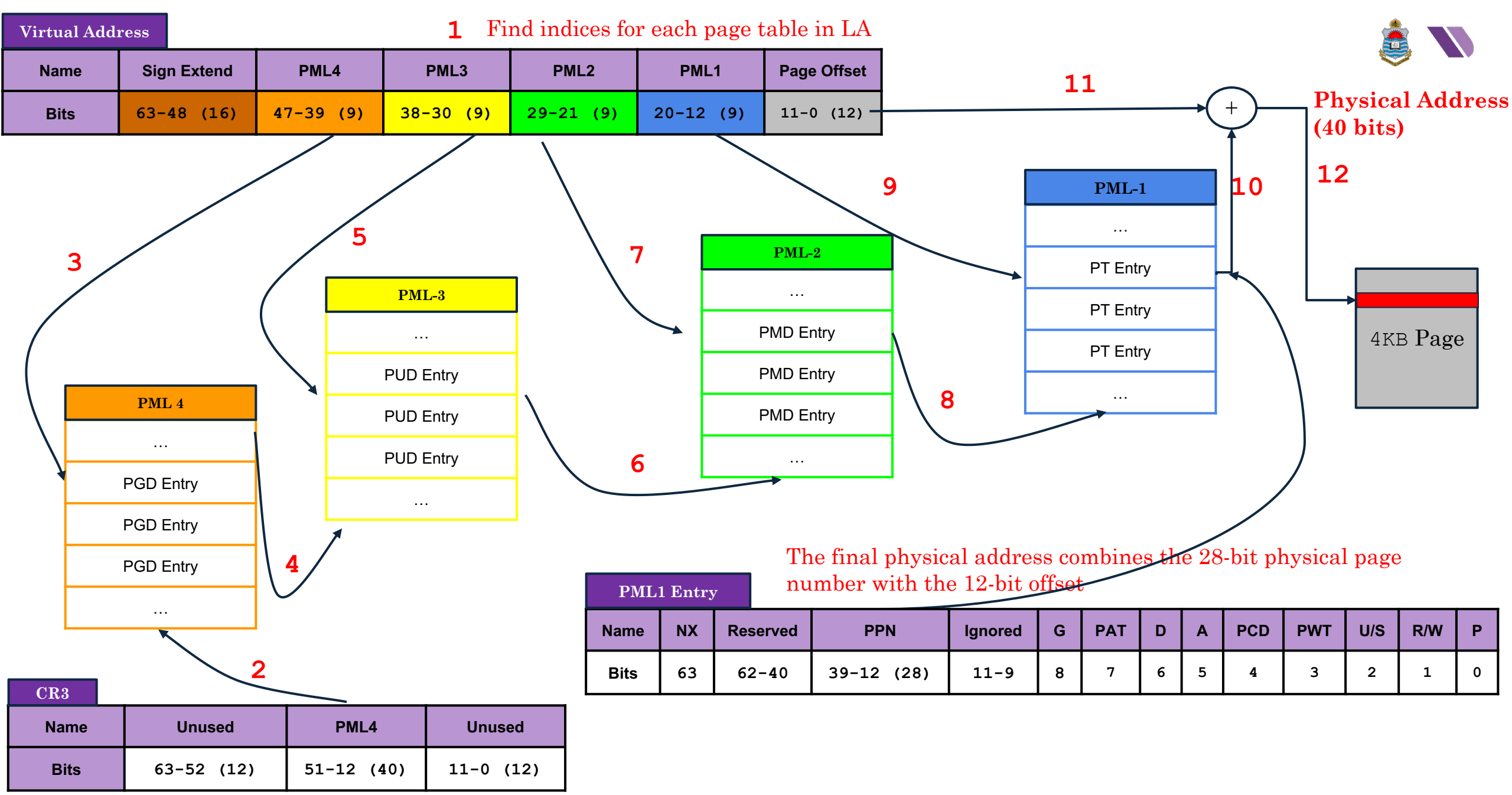
Virtual Address

Name	Sign Extend	PML4	PML3	PML2	PML1	Page Offset
Bits	63-48 (16)	47-39 (9)	38-30 (9)	29-21 (9)	20-12 (9)	11-0 (12)

- CR3 register contains the physical address of PML4 table plus control bits for access permissions and flags.
- PML4 entry contains the physical address of a PML3 table plus control bits for access permissions and flags.
- PML3 entry contains the physical address of a PML2 table plus control bits for access permissions and flags.
- PML2 entry contains the physical address of a PML1 table plus control bits for access permissions and flags.
- PML1 entry contains the PA of the actual 4KB page frame plus control bits for access permissions and flags.

Page Table Characteristics:

- Each of the four page table levels has a size of 4 KiB.
- Each page table contains exactly 512 entries.
- Each entry is 8 bytes in size.
- Each entry either points to the next-level page table OR contains the final physical address that the virtual address resolves to.


Virtual Address
1 Find indices for each page table in LA
Name
Sign Extend
PML4
PML3
PML2
PML1
Page Offset
Bits
63-48 (16)
47-39 (9)
38-30 (9)
29-21 (9)
20-12 (9)
11-0 (12)
11
+
Physical Address (40 bits)
12
9
10
PML-1
…
PT Entry
PT Entry
PT Entry
…
3
5
7
PML-3
…
PUD Entry
PUD Entry
PUD Entry
…
PML-2
…
PMD Entry
PMD Entry
PMD Entry
…
8
6
PML 4
…
PGD Entry
PGD Entry
PGD Entry
…
4
4KB Page
2
The final physical address combines the 28-bit physical page number with the 12-bit offset
PML1 Entry
Name
NX
Reserved
PPN
Ignored
G
PAT
D
A
PCD
PWT
U/S
R/W
P
Bits
63
62-40
39-12 (28)
11-9
8
7
6
5
4
3
2
1
0
CR3
Name
Unused
PML4
Unused
Bits
63-52 (12)
51-12 (40)
11-0 (12)

Sample Problem

Consider x86-64 processor using 4-level paging. You need to translate the given 48 bit virtual address `0x1A2B3C4D5678` to a 40-bit physical address. Perform the translation process using following information:

- **CR3 Register:** Contains `0x8000A000` (physical address of PML4 table)
- **PML4 Entry at index:** Contains `0x8000B000000000FF`
- **PML3 Entry at index:** Contains `0x8000C000000000FF`
- **PML2 Entry at index:** Contains `0x8000D000000000FF`
- **PML1 Entry at index:** Contains `0x12345678000001FF`

Hint:

- Step1: Break down the virtual address into its components
- Step 2: Calculate the base address, entry address, entry value and next level address of all four PTs
- Step 3: Extract the physical page number from the PML1 entry.
- Step 4: Generate the final 40-bit physical address

Example

Step 1: Break Down the Virtual Address: `0x1A2B3C4D5678`

- Bits 47-39 (PML4 Index): `0x34` (52 decimal)
- Bits 38-30 (PML3 Index): `0x56` (86 decimal)
- Bits 29-21 (PML2 Index): `0xF1` (241 decimal)
- Bits 20-12 (PML1 Index): `0x6A` (106 decimal)
- Bits 11-0 (Page Offset): `0x678` (1656 decimal)

Example

- **Step 2:**
- PML4 Table Address:
  - Starting Address: CR3 = 0x8000A000
  - Entry Address: 0x8000A000 + (52 × 8) = 0x8000A000 + 0x1A0 = 0x8000A1A0
  - Entry Value: 0x8000B000000000FF
  - Next Level Address: 0x8000B000000000FF & 0x000FFFFFFFFFF000 = 0x8000B000
- PML3 Table Address:
  - Starting Address: 0x8000B000
  - Entry Address: 0x8000B000 + (86 × 8) = 0x8000B000 + 0x2B0 = 0x8000B2B0
  - Entry Value: 0x8000C000000000FF
  - Next Level Address: 0x8000C000000000FF & 0x000FFFFFFFFFF000 = 0x8000C000
- PML2 Table Address:
  - Starting Address: 0x8000C000
  - Entry Address: 0x8000C000 + (241 × 8) = 0x8000C000 + 0x788 = 0x8000C788
  - Entry Value: 0x8000D000000000FF
  - Next Level Address: 0x8000D000000000FF & 0x000FFFFFFFFFF000 = 0x8000D000
- PML1 Table Address:
  - Starting Address: 0x8000D000
  - Entry Address: 0x8000D000 + (106 × 8) = 0x8000D000 + 0x350 = 0x8000D350
  - Entry Value: 0x12345678000001FF

Example

Step 3: Extract Physical Page Number from PML1 Entry

- PML1 Entry: 0x12345678000001FF
- Extract bits 39-12 (Physical Page Number):
  - PML1 Entry in binary: 0001 0010 0011 0100 0101 0110 0111 1000 0000 0000 0000 0000 0000 0001 1111 1111
  - Bits 39-12: 000100100011010001010110011 1 (28 bits)
  - Physical Page Number: 0x1234567

Step 4: Generate Final Physical Address

- Formula: Physical Address = (Physical Page Number << 12) | Page Offset
- Calculation:
  - Physical Page Number: 0x1234567
  - Page Offset: 0x678
  - Physical Address: (0x1234567 << 12) | 0x678
  - Physical Address: 0x1234567000 | 0x678
  - Final Physical Address: 0x1234567678

Memory Management Related Info in /proc/ File System


System-wide Memory Information

- /proc/meminfo → Overall memory statistics (total, free, available, buffers, cached, swap, etc.).
- /proc/vmstat → Detailed virtual memory statistics (page ins/outs, faults, swaps, NUMA stats).
- /proc/zoneinfo → Info about memory zones (DMA, Normal, HighMem).
- /proc/pagetypeinfo → Breakdown of free pages by migration type and order.
- /proc/buddyinfo → Buddy allocator information (free blocks of various sizes).
- /proc/kpagecount → For each physical page frame, how many times it is mapped.
- /proc/kpageflags → Flags describing state of each physical page.
- /proc/kpagecgroup → Memory cgroup ownership of pages.
- /proc/sys/vm/ → Tunable virtual memory parameters (e.g., swappiness, overcommit_memory, dirty_ratio).

Per-Process Virtual Memory Information (for each process /proc/[pid]/)

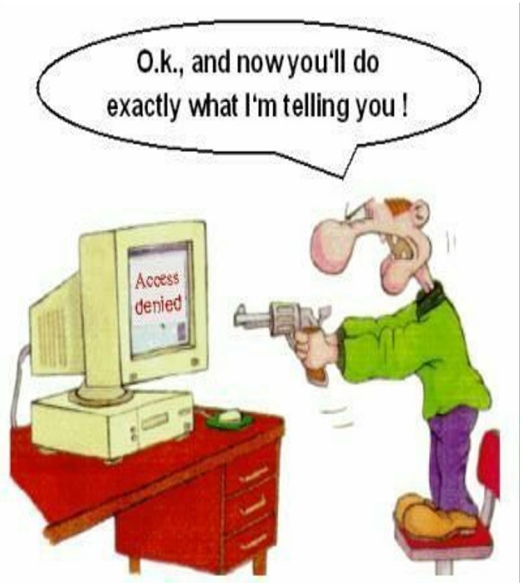
- /proc/[pid]/maps → Memory regions mapped by the process (heap, stack, shared libs, mmap).
- /proc/[pid]/smaps → Detailed memory usage per mapping (RSS, PSS, shared/private clean/dirty).
- /proc/[pid]/smaps_rollup → Summary of smaps (aggregated stats).
- /proc/[pid]/status → High-level memory usage of process (VmSize, VmRSS, VmData, VmStack).
- /proc/[pid]/statm → Memory usage in pages (size, resident, shared, text, data).
- /proc/[pid]/numa_maps → NUMA node allocation of a process's memory.
- /proc/[pid]/pagemap → Mapping of virtual pages to physical frames.

Swap and Huge Pages

- /proc/swaps → Active swap areas and their usage.
- /proc/sys/vm/nr_hugepages → Number of huge pages configured.
- /proc/meminfo → (also reports huge page stats like HugePages_Total, HugePages_Free).

To do

- Carefully review all concepts discussed in class and go through the slides to build a clear understanding of non-contiguous memory allocation schemes.
- Understand in detail the working of address translation from LA to PA in a paged segmentation architecture.
- Practice address translation problems in paged segmentation.
- Draw memory layout diagrams for address translation of Intel 8080, 8086, 80386 and x86-64 architectures.
- Prepare a list of possible exam-style short/long questions from today's content and try answering them without notes.
- Form small groups to discuss and cross-check answers for numerical and conceptual questions.



Coming to office hours does NOT mean that you are academically weak!