Addressing and Memory Organization of 8086 Microprocessor

1. Address Capability of 8086

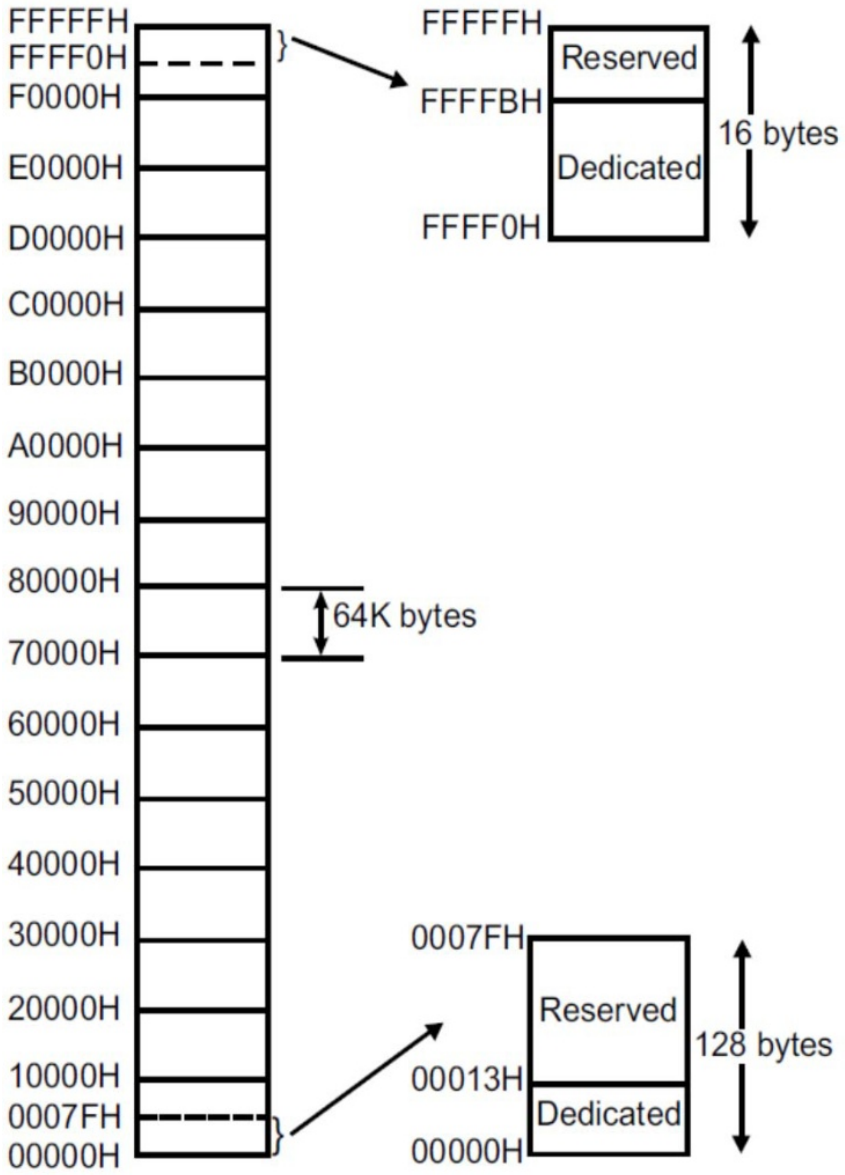


Figure 1: 8086 Memory Map (1 MB addressable space)

8086 Address Bus and Memory Space

- **Address Bus:** 20 bits wide, allowing $2^{20} = 1$ MB of addressable memory.
- **Memory Range:** 00000H to FFFFFH.
- **Memory Blocks:** 16 blocks, each $2^{16} = 64$ kB.

- **Segment Increments:** Most significant hex digit of segment address increases by 1 for each block.
- **Reserved Locations:** Allocated for future hardware and software needs.
- **Dedicated Locations:** Reserved for system interrupts and reset functions.

1.1 Reserved and Dedicated Memory Locations

Special Memory Areas in 8086

- Certain regions of the 1 MB address space are reserved for system-level operations such as interrupts, reset, and BIOS services. These are not available for general user programs.
- **Interrupt Vector Table (IVT):** Located at the beginning of memory from `00000H` to `003FFH` (1 KB).
- Contains 256 entries (one for each interrupt type).
- Each entry = 4 bytes (2 bytes for IP, 2 bytes for CS).
- Example: Type 0 interrupt (Divide Error) vector stored `00000H--00003H`.
- **BIOS Data Area (BDA):** Located in the range `00400H{004FFH` (256 bytes). Stores hardware-related data (I/O port addresses, system status, etc.).
- **System BIOS ROM Area:** Located near the top of memory, typically `F0000H{FFFFFH` (64 KB). Contains firmware routines for device initialization and low-level hardware access.
- **Reset Vector:** After hardware reset, the 8086 starts execution at physical address `FFFF0H` (16 bytes before the end of 1 MB). - Segment: `FFFFH`, Offset: `0000H`. - This ensures the CPU begins execution in the system BIOS.

2. Memory Segmentation of 8086

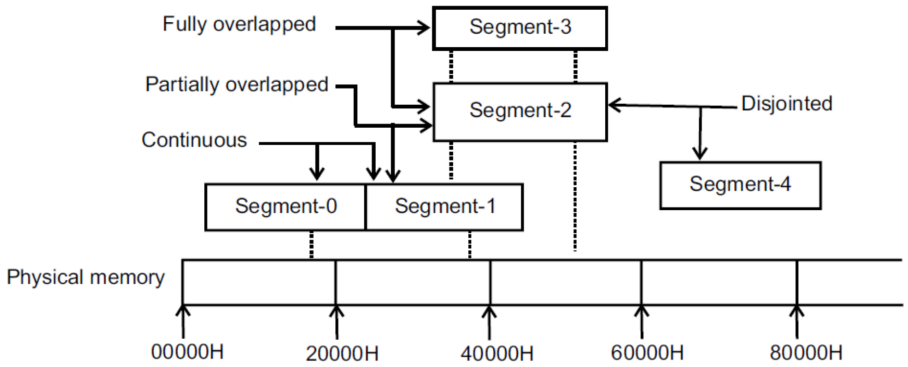


Figure 2: Memory Segmentation Types

Segmentation Overview

- 8086 memory can be organized in **segments** to simplify programming with 16-bit registers.
- Maximum memory = 1 MB (16 segments × 64 kB each).
- Types of segmentations:
  1. Continuous
  2. Partially overlapped
  3. Fully overlapped
  4. Disjointed
- At any instant, **4 segments can be active**: CS, DS, SS, ES.
- Segment registers hold the **starting address** of the segment.

Segment Overlaps in 8086

- In 8086, segments can overlap in memory because each segment starts at a multiple of 16 bytes (segment address × 16).
- This means the physical addresses of two segments may share some memory locations.
- Example:
  - Segment1: 1000H → physical range 10000H–1FFFFH
  - Segment2: 1001H → physical range 10010H–1FFFFH
  - Overlap occurs from 10010H onward.
- Advantages of overlapping segments:
  - Efficient memory utilization; multiple data/code sections can share memory.
  - Supports flexible access to large programs without needing 20-bit contiguous addresses.
- Programmer implication: Care must be taken to avoid unintended overwrites when accessing overlapping segments.

Memory Segment Sizes

- Each segment size = 2^{16} = 64 kB.
- Maximum active memory at a time = 64 kB × 4 = 256 kB.
- Program storage: 64 kB in CS
- Stack: 64 kB in SS

- Data storage: 128 kB in DS and ES

Clarification on "Maximum Active Memory"

- Physically, the 8086 can address the full **1 MB memory space**.
- What is meant by "256 kB active" is that only four segments (CS, DS, SS, ES) can be referenced directly at any moment using the 16-bit segment registers.
- Each segment = 64 kB → 4 segments × 64 kB = 256 kB "immediately accessible" via registers.
- To access other memory areas, the CPU can reload segment registers to point to a different segment.

Memory Segment	Segment Register	Content / Uses
Code Segment	CS	Instruction codes of the program
Data Segment	DS	Data, variables, and constants
Stack Segment	SS	Interrupt and subroutine return addresses
Extra Segment	ES	Destination of data for string instructions

2.1 Segment Base and Offset

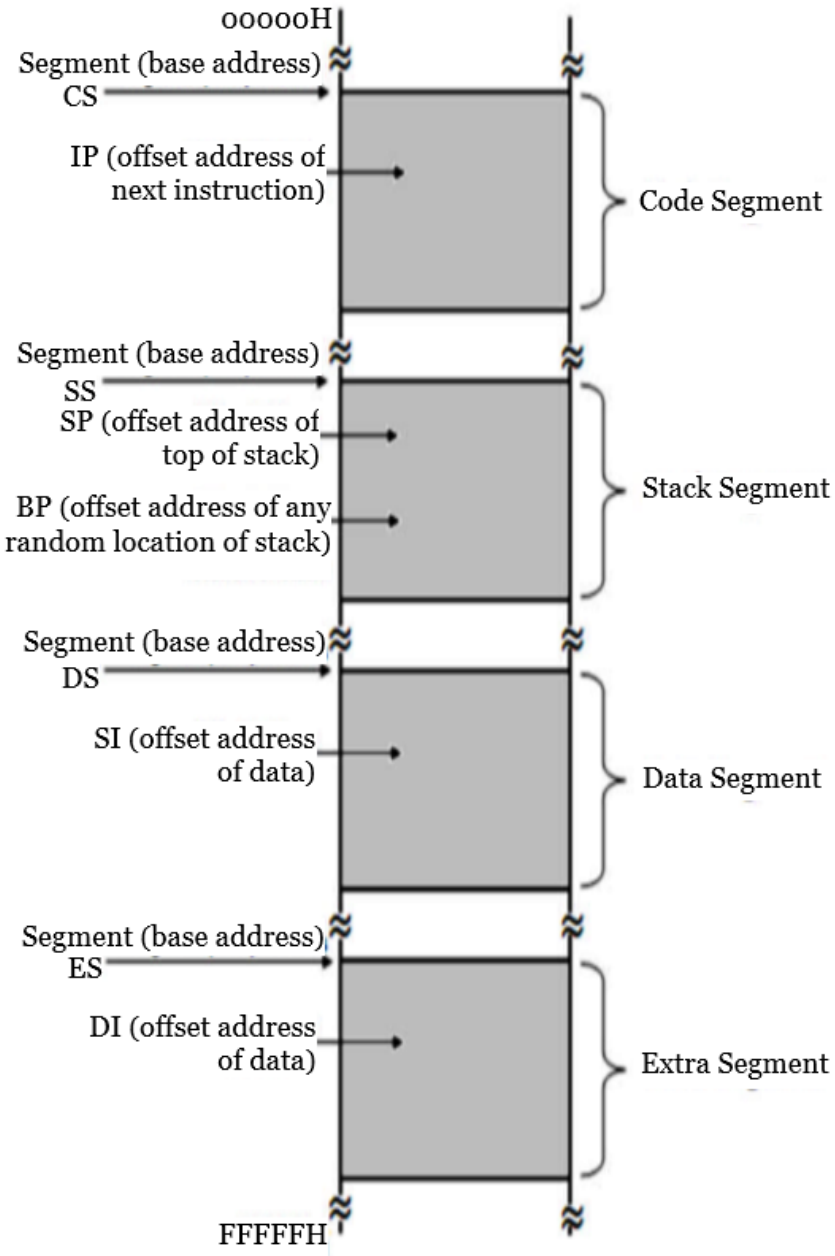


Figure 3: Illustration of Segment Base + Offset = Physical Address in 8086

Segment Base and Offset

- **Base Address of Segment:** Segment address $\times 10H$.
- Example: Segment addresses 0000H, 0001H, 0002H correspond to physical addresses 00000H, 00010H, 00020H, etc.
- **Offset Address:** Range = 0000H to FFFFH (16-bit).
- **Effective Address (EA):** Calculated as

$$\text{EA} = \text{Base Address} + \text{Offset}$$

- **Advantages:**
  - Enables 8086 to address more than 64 kB using 16-bit registers.
  - Supports multiple code, data, stack, and extra segments.
  - Facilitates time-shared multitasking: CPU can reload 4 segment registers to switch between programs.
  - Allows separation of program code and data.

3. Logical and Physical Addresses

Address Calculation

- **Segment Address:** A 16-bit value stored in the segment registers: `CS` (Code Segment), `DS` (Data Segment), `SS` (Stack Segment), and `ES` (Extra Segment). Each segment register points to the starting location (base) of a 64 KB memory segment.
- **Base Address:** The actual 20-bit starting address of a segment, obtained by multiplying the segment address by `10H` (i.e., shifting left by 4 bits).

$$\text{Base Address} = \text{Segment Address} \times 10H$$

- **Offset Address:** Also known as the *effective address*, it is a 16-bit value stored in general-purpose or pointer/index registers such as `IP`, `BP`, `SP`, `BX`, `SI`, or `DI`. The offset specifies the position of a particular byte/word within the segment.
- **Logical Address:** The programmer-visible address, represented in the form:

$$\text{Logical Address} = \text{Segment} : \text{Offset}$$

  Example: `DS:1234H`
- **Physical Address:** The actual 20-bit address in memory, calculated by combining the segment base address and the offset.

$$\text{Physical Address} = (\text{Segment Address} \times 10H) + \text{Offset Address}$$

  For example, if `DS = 1000H` and `Offset = 1234H`, then:

$$\text{Physical Address} = (1000H \times 10H) + 1234H = 10000H + 1234H = 11234H$$

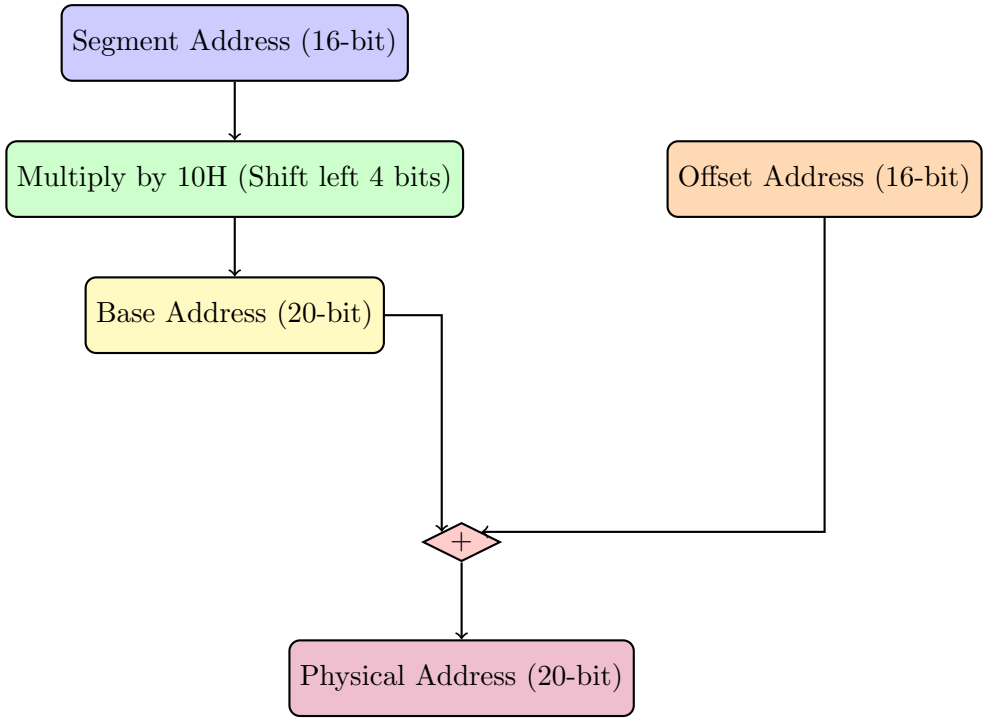


Figure 4: Conversion of Logical Address (Segment:Offset) into Physical Address

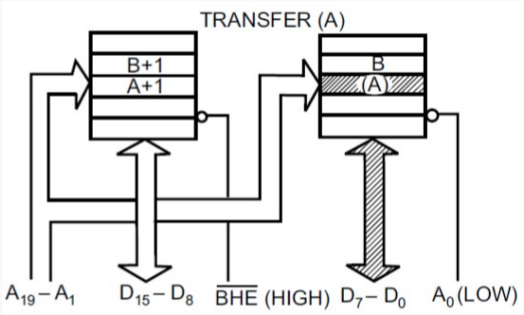
4. Memory Organization

8086 Memory Organization

- The 8086 has a **20-bit address bus** (`A0--A19`), allowing it to access up to **1 MB** of memory.
- Memory is divided into **two banks** for efficient 16-bit access:
  - **Low Bank (Even addresses):** Connected to data lines `D0--D7`.
  - **High Bank (Odd addresses):** Connected to data lines `D8--D15`.
- The CPU can access either an **8-bit byte** or a **16-bit word**:
  - **Byte access:** Only one bank is enabled.
  - **Word access:** Both banks are enabled simultaneously.
- **Bank Selection Mechanism:**
  - `A0` (Address Line 0) and `BHE` (Bus High Enable) control which bank(s) are active.
  - `CS` (Chip Select) from the memory chip must also be active to enable reading/writing.

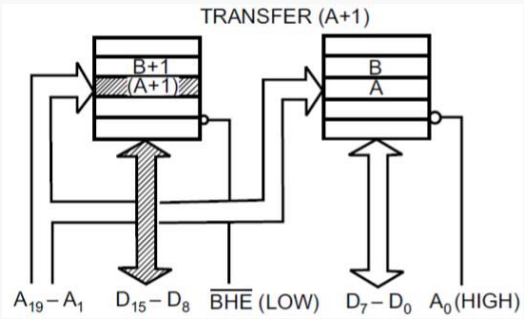
BHE	A0	Bank/Access Type
0	0	Both banks active ⇒ 16-bit word (even address)
0	1	High bank active ⇒ Odd-address byte
1	0	Low bank active ⇒ Even-address byte
1	1	No bank active

- **Why A0 is not provided externally:**
  - In 8086, the data bus is 16-bit (D0–D15).
  - Since the CPU fetches data in words, the least significant address bit (`A0`) does not need to be sent to external memory chips.
  - Instead, `A0` is internally used together with `BHE` to decide which bank(s) to activate.
- **Detailed Operation:**
  1. **Even-addressed byte:**
     - Condition: `A0 = 0, BHE = 1`
     - Only the **low bank** (D0–D7) is enabled.
     - The processor reads/writes a single byte from an even address.
     - Example: Accessing `10000H` → data goes through D0–D7.



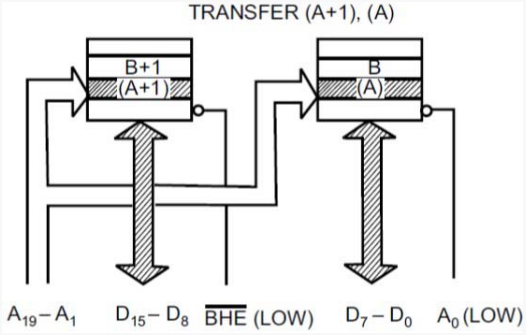
Even-addressed byte transfer using Low Bank (D0–D7)

  2. **Odd-addressed byte:**
     - Condition: `A0 = 1, BHE = 0`
     - Only the **high bank** (D8–D15) is enabled.
     - The processor reads/writes a single byte from an odd address.
     - Example: Accessing `10001H` → data goes through D8–D15.



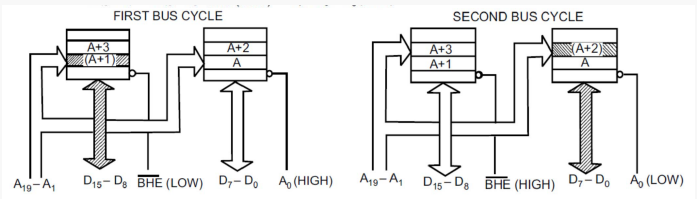
Odd-addressed byte transfer using High Bank (D8–D15)

3. **Even-addressed word (16-bit word at even boundary):**
   - Condition: `A0 = 0, BHE = 0`
   - Both banks are enabled: - Low bank (D0–D7) stores the low-order byte. - High bank (D8–D15) stores the high-order byte.
   - A single memory cycle is sufficient.
   - Example: Accessing word at `10000H` → - 10000H → low-order byte (D0–D7) - 10001H → high-order byte (D8–D15).



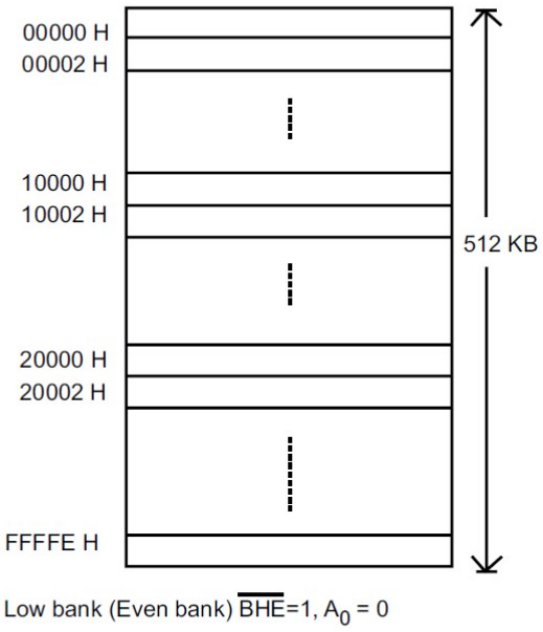
Even-addressed word transfer (D0–D15 active)

4. **Odd-addressed word (16-bit word at odd boundary):**
   - More complex, requires **two memory cycles**.
   - Why? Because the word straddles across banks and addresses.
   - Steps:
     (a) First cycle: `A0 = 1, BHE = 0` → fetches high byte from 10001H (via D8–D15).
     (b) Second cycle: `A0 = 0, BHE = 1` → fetches low byte from 10002H (via D0–D7).
   - Example: Accessing word starting at `10001H` → - 10001H (high byte, D8–D15) - 10002H (low byte, D0–D7).

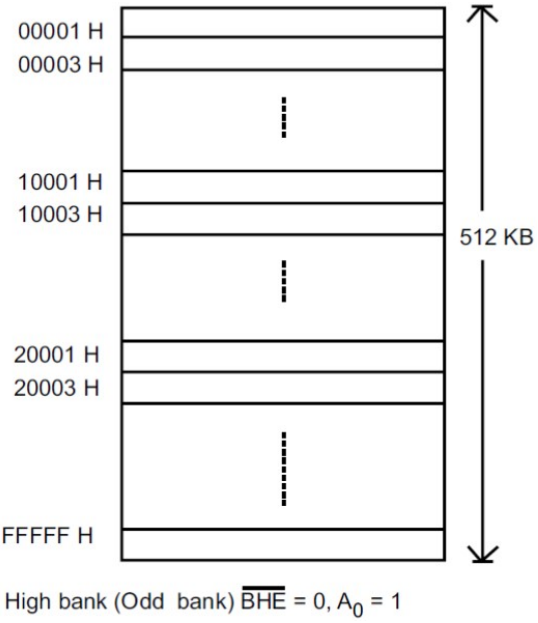


Odd-addressed word transfer (two cycles required)

- **Summary:**
  - Memory is organized in two 8-bit banks (low = even, high = odd).
  - `BHE` and `A0` signals control bank selection.
  - `CS` ensures only the selected chip responds.
  - `A1--A19` form the external address bus (since A0 is internally used).
  - This organization allows efficient word transfers on a 16-bit bus while keeping compatibility with byte-level addressing.



(a) Low Bank (Even addresses: D0–D7)



(b) High Bank (Odd addresses: D8–D15)

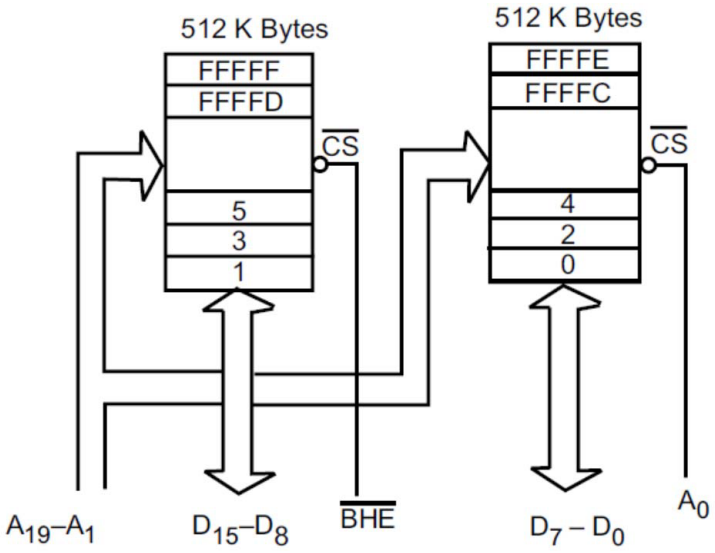


Figure 5: 8086 Memory Organization: (a) Low Bank, (b) High Bank, (c) Bank Selection and Access Operation

5. Summary

Key Takeaways

- The **8086 microprocessor** has a 20-bit address bus, enabling access to a total of **1 MB of memory space**.
- Memory is logically divided into **segments** of 64 KB each. At any time, up to four segments can be active through the segment registers: **CS (Code Seg-**

ment), **DS (Data Segment)**, **SS (Stack Segment)**, and **ES (Extra Segment)**.

- A memory location is specified using: **Logical Address** = `Segment:Offset` (16-bit each). **Physical Address** = `(Segment × 16) + Offset` (20-bit).
- Segmentation provides:
  - Flexible separation of code, data, and stack.
  - Efficient support for modular programming.
  - A mechanism for relocation and multitasking in larger programs.
- Memory is organized into **two banks** of 8 bits each (Low Bank = D0–D7, High Bank = D8–D15) to support 16-bit word transfers.
- The signals **A0** and **BHE (Bus High Enable)** control whether a byte or a word is accessed:
  - `A0 = 0, BHE = 0` → 16-bit word (both banks).
  - `A0 = 0, BHE = 1` → Low bank only (even address byte).
  - `A0 = 1, BHE = 0` → High bank only (odd address byte).
  - `A0 = 1, BHE = 1` → No valid access.
- For **word access at odd addresses**, the 8086 requires **two memory cycles**, as the word spans both banks.