8086 minimum mode

- 8086 works in Minimum Mode, when MN/ ¯MX = 1.
- Minimum Mode, 8086 is the only processor in the system. The Minimum Mode circuit of 8086 is as shown below:
- Clock is provided by the 8284 clock generator, it provides CLK, RESET and READY input to 8086.
- Address from the address bus is latched into 8282 8-bit latch. Three such latches are needed, as address bus is 20-bit. The ALE of 8086 is connected to STB of the latch. The ALE for this latch is given by 8086 itself.
- The data bus is driven through 8286 8-bit trans-receiver. Two such trans-receivers are needed, as the data bus is 16-bit. The trans-receivers are enabled through the DEN signal, while the direction of data is controlled by the DT/ ¯R signal. ¯DEN is connected to ¯OE and DT/ ¯R is connected to T. Both ¯DEN and DT/ ¯R are given by 8086 itself.

$\overline{DEN}$	DT/ $\overline{R}$	Action
1	X	Transreceiver is disabled
0	0	Receive data
0	1	Transmit data

- Control signals for all operations are generated by decoding M/¯IO , ¯RD and ¯WR signals.

M/ $\overline{IO}$	$\overline{RD}$	$\overline{WR}$	Action
1	0	1	Memory Read
1	1	0	Memory Write
0	0	1	I/O Read
0	1	0	I/O Write

- M/¯IO , ¯RD and ¯WR are decoded by a 3:8 decoder like IC 74138. Bus Request (DMA) is done using the HOLD and HLDA signals.
- ¯INTA is given by 8086, in response to an interrupt on INTR line.

Maximum mode 8086

- These max.mode pins can be described as follows:

1. $\overline{QS_0}$ $\overline{Qs_1}$ **:** These are Q status pin, which are monitored by 8087 NDP to synchronize with 8086 microprocessor.

2. $\overline{S0}$, $\overline{S_1}$, $\overline{S_2}$ **:** These are status pins initiated with binary coded information about type of access, the up will be performing with memory or IO devices.

3. $\overline{TEST}$ pin **:** It is used in conjunction with WAIT instruction in multiprocessor environment.

4.$\overline{LOCK}$ pin **:** This output pin of up activated while up is executing an instruction with $\overline{LOCK}$ prefix.

5. $\overline{RQ}/\overline{GT}$ **:** There are request/grant pins used by 8087 NDP or 8089 IOP to request for system bus control in multiprocessor environment.

A typical 8086 maximum mode s/m diagram will be as follow:

- In this mode latches are used to demultiplex ADo-A19/S6 of up to obtain address lines A0-A19.
- Transceivers are used to provides two sets of data buses D0 - D7 and D8 - D15.
- The 8284 clock generator is used to provide CLK, RESET AND READY signal for the up as well as for 8087 NDP.
- The 8288 bus controller is responsible for status decoding (on S0 S1 S2) to generate.

Instruction Set
https://www.ques10.com/p/41337/classify-and-explain-8086-instruction-set-1/

PipeLining

☞ (4) 6-Byte Pre-Fetch Queue

- 8086 implements 2-stage pipelining.
- **Fetching the next instruction while executing the current instruction is called pipelining.**
- The instructions that are pre-fetched are stored in the 6-byte prefetch queue.
- It works in first in first out manner which means the order in which instructions are fetched by the BIU is the same order in which they are removed by the EU for decoding and execution.
- **BIU fetches** the next **"six instruction-bytes"** from the Code Segment and stores it into the queue.
- Execution Unit (EU) removes instructions from the queue and executes them.
- As EU removes instructions from the bottom end of the queue, there is empty space created at the top of the queue. **The queue is refilled when atleast two bytes are empty as 8086 has a 16-bit data bus.**

☞ Advantages of pipelining

- The obvious advantage of Pipelining is that it **increases** the **efficiency** of µP.
- Time is no longer spent on fetching as it happens alongside execution.
- The following graph shows the positive effect of pipelining on a processors performance.
- Consider a program of 5 instructions stored in the memory.
- The following is the behavior of a non pipelined processor like 8085 as compared to a pipelined processor like 8086. You can clearly see the speed advantage achieved by 8086 due to pipelining.

☞ Drawbacks of pipelining

- Pipelining assumes the program will always be executed in a sequential manner.
- Hence it **fails when a branch** occurs, as the pre-fetched instructions are no longer useful. As soon as 8086 detects a branch operation, it clears/discards the entire queue.
- Now, the next six bytes from the new location (branch address) are fetched and stored in the queue and Pipelining continues.
- In advanced processors like Pentium which have very deep and superscalar pipelines, this drawback is almost neutralized due to a highly efficient technique called **Branch Prediction.**

Reset

1.7.2 RESET

Q. Write short note on generation of RESET Signals in 8086 based system (Dec. 15, 5 Marks)

Q. Explain power on reset circuit used in 8086 system. (Dec. 17, 5 Marks)

- This is the reset input signal used to "reset" the µP and hence the whole computer.
- On reset : CS becomes FFFFH and IP becomes 0000H.
- This automatically means the Physical address becomes FFFF0H (CS × 10H + IP)
- This address "FFFF0H" is called the reset vector address of 8086.
- It means whenever 8086 is reset it will always go to this address to fetch the first instruction.
- The program stored at this address is the first program to be executed by the µP on reset.
- It is called the "monitor program" also called the BIOS program, used to "boot up" the system.
- The reset signal is provided by the 8284 clock generator.
- Generation of reset signal is done using a **power on reset circuit.**
- The purpose of this circuit is to activate the reset signal as soon as we supply power to 8086.
- This is done so that CS & IP acquire the above values and the system can initialize the monitor program (BIOS) as soon as 8086 is powered on. This is also called COLD starting the system.

Pins:

1.7.4 $\overline{\text{TEST}}$

- It is an active low **input** line dedicated for 8087 Co-processor.
- **$\overline{\text{TEST}}$ can only be connected with 8087 in Max mode.**
- Otherwise this signal is of no consequence.
- **$\overline{\text{TEST}}$** is used by 8086 to check whether 8087 is busy or not.
- BUSY pin of 8087 is connected to **$\overline{\text{TEST}}$** of 8086.
- µP checks **$\overline{\text{TEST}}$** when we write WAIT instruction. WAIT is written just before an 8087 instruction.
- If **$\overline{\text{TEST}}$** is "0" it means 8087 is not busy. In this case µP will simply continue.
- If **$\overline{\text{TEST}}$** is "1" it means 8087 is still busy completing the previous instruction.
- In such a case µP must not go ahead with the following instruction as the forthcoming instruction is again if 8087 and needs 8087 to be free for its execution. Hence µP will enter wait state. This allows 8087 extra time to complete its operation.
- Once 8087 finishes its execution, it makes BUSY signal "0".This makes **$\overline{\text{TEST}}$** signal "0".
- Now µP comes out of wait state and will proceed ahead with the program.

1.7.5 MN/ $\overline{MX}$

- This is an **input** signal to 8086.
- It is used to inform the µP whether it is working in minimum mode or maximum mode.
- If **MN/ $\overline{MX}$** is "1" that means the µP is working in minimum mode.
- In minimum mode there is only on processor that is 8086.
- If **MN/ $\overline{MX}$** is "0" that means the µP is working in maximum mode.
- In maximum mode there can be multiple processors alongside 8086 in the same circuit.
- We could connect an 8087, which could work as the numeric processor for powerful arithmetic operations such as trigonometry, log etc..
- We could also connect an 8089 which works as a dedicated I/O processor used to perform sophisticated I/O data transfers.
- The behavior of a lot of pins of µP changes when it switches from min mode to max mode.
- Hence on reset µP immediately checks this signal to determine how the pins and the µP itself must function.

1.7.6 NMI

- It is one of the two **hardware interrupt** input lines to 8086.
- The other one being INTR.
- NMI is the **higher priority** interrupt.
- It means if NMI and INTR occur together, NMI will be serviced first.
- NMI is **Non Maskable**, hence the name.
- It means NMI cannot be disabled by the CLI instruction. CLI stands for Clear Interrupt Flag.
- NMI is a **Vectored interrupt**.
- It means NMI has a fixed vector number in the IVT. Its vector number is 2.
- On receiving an interrupt on NMI line, the µP executes **INT 2** i.e. and takes control to location $2 \times 4 = 00008H$ in the Interrupt Vector Table (IVT), to get the value for CS ad IP.
- NMI is **edge-triggered**.
- It means it is recognized on the rising edge (when it goes from low to high) by the µP.
- NMI is typically used for hardware failures which require immediate attention such as failure, overheating etc.

HLDA:

1.7.13 HOLD — $\overline{RQ_0}$ / $\overline{GT_0}$

- In **Minimum** Mode this line carries the **HOLD** input signal.
- The **DMA Controller issues** the HOLD signal to request for the system bus.
- In response 8086 completes the current bus cycle and releases control of the system bus.
- µPin forms DMAC that the bus has been released by giving HLDA signal.

ALE

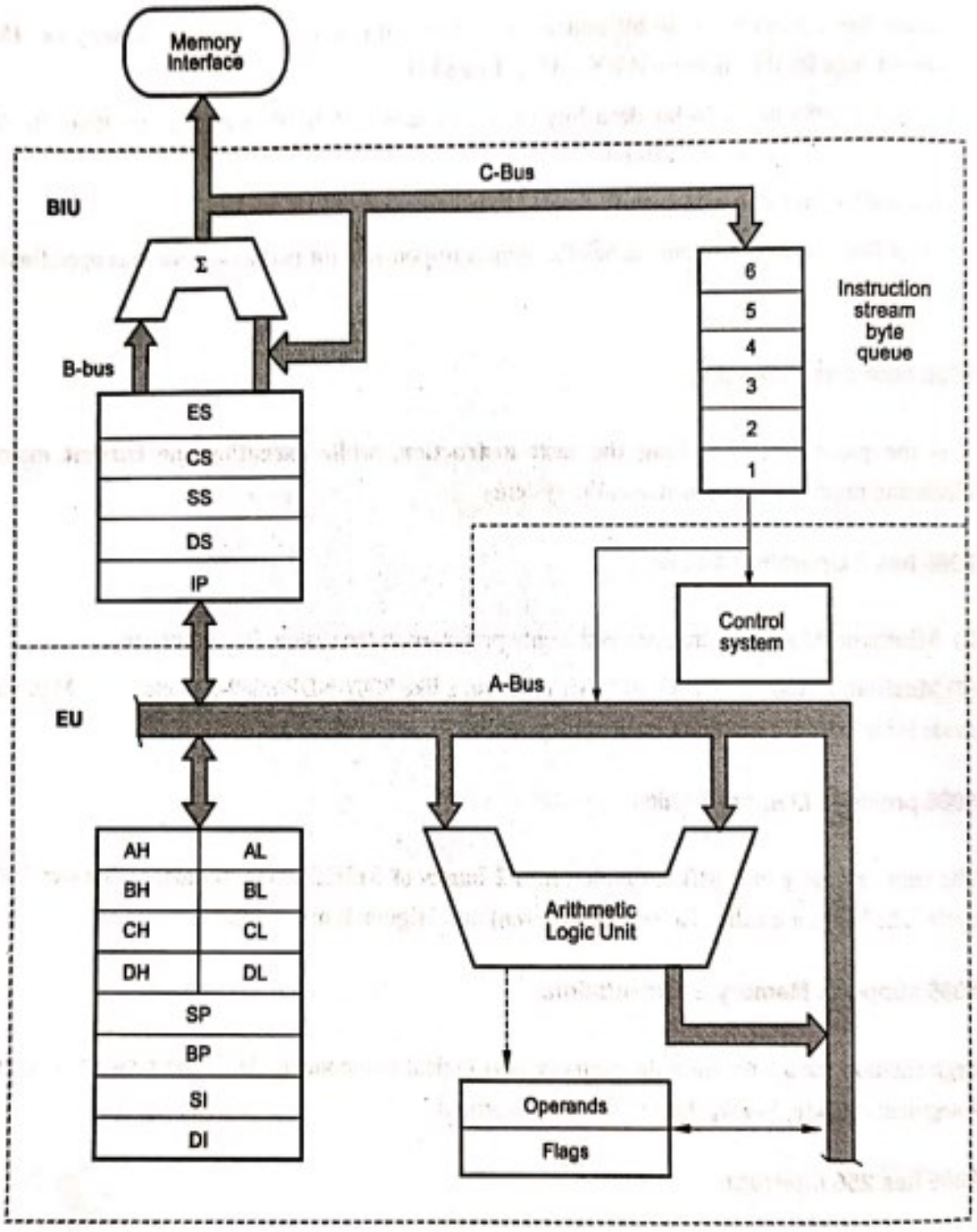
1.7.19 ALE – QS_0

- In **Minimum** Mode it carries the **ALE** signal, which is used to latch the address.

S0,s1,s2

In Maximum mode: $\overline{S_2}$, $\overline{S_1}$ and $\overline{S_0}$ are decoded by the 8288 bus controller to decide which operation has to be performed and hence which control signal has to be generated

$\overline{S_2}$	$\overline{S_1}$	$\overline{S_0}$	Processor State	8288 Active Output
0	0	0	Int. Acknowledge	$\overline{INTA}$
0	0	1	Read I/O Port	$\overline{IORC}$
0	1	0	Write I/O Port	$\overline{IOWC}$ and $\overline{AIOWC}$
0	1	1	Halt	None
1	0	0	Instruction Fetch	$\overline{MRDC}$
1	0	1	Memory Read	$\overline{MRDC}$
1	1	0	Memory Write	$\overline{MWTC}$ and $\overline{AMWTC}$
1	1	1	Inactive	None



(1A2)**Fig. 1.4.1 : 8086 Architecture**

8086 is a 2-stage pipelined processor. The two stages of pipelining are fetching and execution.

Based on this the 8086 architecture is divided into two functional units :

1. Bus Interface Unit (BIU)
2. Execution Unit (EU)

1.4.1 Bus Interface Unit (BIU)

- The main function of the BIU is to fetch instructions.
- Instructions are we know are stored in the memory are fetched using the various buses.
- The Bus Interface Unit connects the processor with the external components such as memory and I/O.
- All external data transfers with memory and I/O namely Memory Read, Memory Write, I/O Read and I/O write as well as Instruction Fetches are handled by the BIU. These operations are also called Machine cycles or Bus Cycles. Hence we say the BIU operates with respect to bus cycles.
- To perform any memory operation the BIU needs to calculate the 20-bit Physical Address.
- This is done using the formula Physical Address = (Segment Address × 10H) + Offset Address.
- Instructions are fetched from the code segment. CS register provides the segment address.
- IP register carries the offset address of the nest instruction.
- Hence the Physical Address is calculated as CS × 10H + IP.

  E.g.: If CS = 1234H and IP = 0005H, Physical Address = 12345H
- This calculation is done by the arithmetic circuit shown in the BIU.

Functions of BIU

- It performs the following functions :
  (1) It **generates** the 20-bit **physical address** for memory access.
  (2) **Fetches Instruction** from memory.
  (3) **Transfers data** to and from the **memory and IO.**
  (4) **Manages Pipelining** using the 6-byte instruction queue.
- The main components of the BIU are :
  (1) Segment Registers (2) IP
  (3) Address Generation Circuit (4) Prefetch Queue.

(1) Segment Registers

CS Register

- It is a 16-bit register.
- CS holds the **base** (Segment) **address** for the **Code Segment.**
- Code segment contains all the programs and hence is the segment from which instructions are fetched.
- The value of CS register is **multiplied by 10H** (16_d), to generate**20-bit physical address** of the starting of the **Code Segment.**

  Eg.: If **CS = 4321H** then CS × 10H = **43210H → Starting address** of Code Segment.
- CS register cannot be modified by executing any instruction except branch instructions

DS Register

- It is a 16-bit register.
- DS holds the **base** (Segment) **address** for the **Data Segment.**
- It is **multiplied by 10H** (16_d), to give **the 20-bit physical address** of the **Data Segment.**

  Eg: If **DS = 4321H** then DS × 10H = **43210H → Starting address** of Data Segment.

SS Register

- It is a 16-bit register.
- SS holds the **base** (Segment) **address** for the **Stack Segment.** It is **multiplied by 10H** (16_d), to give the **20-bit physical address** of the **Stack Segment.**

  Eg: If **SS = 4321H** then SS × 10H = **43210H → Starting address** of Stack Segment.

ES Register

- It is a 16-bit register.
- ES holds the **base** (Segment) **address** for the **Extra Segment.**
- It is **multiplied by 10H** (16_d), to give the **20-bit physical address** of the **Extra Segment.**

  Eg: If **ES = 4321H** then ES × 10H = **43210H → Starting address** of Extra Segment.

☞ **(2) Instruction Pointer (IP register)**

- It is a **16-bit register.**
- **It holds offset of the next instruction in the Code Segment.**
- Address of the **next instruction** is calculated as **CS × 10H + IP.**
- IP is **incremented after every instruction byte is fetched.**
- IP gets a new value whenever a branch occurs.

☞ **(3) Address Generation Circuit**

- The BIU has a **Physical Address Generation Circuit.**
- It generates the 20-bit physical address using Segment and Offset addresses using the formula :

  Physical address = Segment Address × 10h + Offset Address

- **Viva Question : Explain the real procedure to obtain the Physical Address?**

  The Segment address is left shifted by 4 positions, this multiplies the number by 16 (i.e. 10h) and then the offset address is added.

- EU has several components such as :

1. Control Section	2. General Purpose Registers
3. ALU	4. Offset Registers
5. Flag Register	

☞ 1. Control Section

- Instructions are already prefetched by the BIU and stored in the queue.
- The EU removes the first instruction from the queue.
- It goes into the Control section for decoding. Every instruction has a unique binary code that represents the operation to be performed called the opcode. Decoding simply means analyzing the opcode to understand the operation to be performed.
- The control section then generates internal control signals that inform all parts of the architecture as to what needs to be done for the execution of the corresponding instruction.

☞ 2. General Purpose Registers

- 8086 has four 16-bit general-purpose registers **AX, BX, CX** and **DX.**
- These are **available** to the programmer, for storing values during programs.
- Each of these can be **divided** into two independent **8-bit registers** such as AH, AL; BH, BL; etc.
- Beside their general use, these registers also have some **specific functions.**

AX Register (16-Bits)

- AX is probably the most useful register from a programmers point of view.
- It is also called the Accumulator, which means it hold the first operand and the result in complex arithmetic operations like Multiply and Divide.
- **E.g. :: MUL BL;** BL will be multiplied with AL and the result will be stored in AX.

Note : Please include such examples of instructions in your exam answer to ensure maximum marks. You will find detailed explanation of these instructions in later chapters of this book.

- Moreover, **All IO data transfers** using IN and OUT instructions use "A" register.
- AL or Al for 8-bit data transfers and AX for 16-bit.
- It functions as accumulator during **string operations such as LODS and STOS** etc.

BX Register (16-Bits)

- BX is also called a memory pointer.
- It is the only General Purpose register that can **hold a memory address in Indirect addressing mode.**
- **E.g.:: MOV CL, [BX];** CL register gets the data from the memory location pointed by BX

CX Register (16-Bits)

- "C" register acts as a default count in many instructions.
- For Rotate and Shift instructions the count is in CL register.
- For Loop and String instructions the count is in CX register.
- **E.g.::LOOP Back :** This instruction creates a loop. The loop count is equal to the value in CX Register.

DX Register (16-Bits)

- DX acts as an extension of the accumulator when 32 bits are needed.
- E.g.:: In 16bit × 16 bit multiplication like **MUL BX,** the answer is 32-bits and is stored in a combination of DX and AX wherein DX gets the higher 16-bits and AX gets the lower.
- DX is also used to **hold** the **address** of the **IO Port** in **Indirect IO addressing** mode.

☞ 3. ALU - Arithmetic and Logic Unit

- 8086 is a 16-bit μP. This means it has a 16-bit ALU.
- The ALU can perform 8-bit and 16-bit operations.
- **E.g.:: ADD BL, CL;** BL gets the result of BL + CL. This is an 8-bit operation.
- **E.g.:: ADD BX, CX;** BX gets the result of BX + CX. This is a 16-bit operation.
- Whenever temporary storage is needed in operations such as XCHG or MUL etc. the ALU uses a temporary register called "Operands". This is not available to the programmer.

☞ 4. Special Purpose Registers (Offset Registers)

SP – Stack Pointer

- SP is 16 bit offset register. SP holds **offset address of** the **top of the Stack.**
- **Stack is a set of memory locations operating in LIFO manner.**
- Stack is present in the memory in **Stack Segment.**
- SP is used with the SS Reg to calculate physical address for the Stack Segment.
- Physical Address of top of stack = SS × 10H + SP.
- It used during instructions like PUSH, POP, CALL, RET etc.
- During **PUSH** instruction, SP is **decremented** by 2.
- During **POP** instruction, SP is **incremented** by 2.

BP – Base Pointer

- BP is 16 bit offset register.
- BP can hold **offset address of** any location in the **stack segment.**
- It is used to access random locations of the stack.

SI – Source index

- SI is 16 bit offset register.
- It is normally used to hold the **offset address** for **Data segment** but can also be used for other segments using Segment Overriding.
- During String instructions like **"MOVSB"**, SI holds **offset address** of **source data** in Data Segment, hence the name Source Index.

DI – Destination index

- DI is 16 bit offset register.
- It is normally used to hold the **offset address** for **Extra segment** but can also be used for other segments using Segment Overriding.
- During String instructions like **"MOVSB"**, DI holds **offset address** of **destination data** in Extra Segment, hence the name Destination Index.

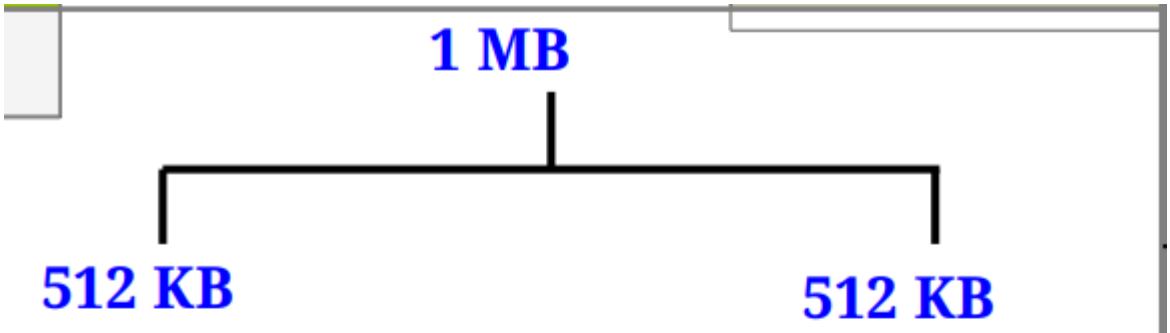
Memory Banking

Memory Banking

- 8086 uses 16 bit data bus i.e., it should be able to access 16 bit data in one cycle.
- And so it needs to read from 2 memory location as one memory location carries 1 Byte data.
- 16 bit data is stored in 2 consecutive location.
- However, if both these memory locations are in same memory chip then they cannot be accessed at the same time, as the address bus of the chip cannot contain two addresses simultaneously.
- **Hence, the memory of 8086 is divided into two banks, each bank provide 8 bits.**

Even Bank and Odd Bank

- Memory is divided in such a manner any two consecutive location lie in two different chips. Hence each chip contain an alternate location.
- One bank contains all even addresses called the **EVEN BANK** while the other containing odd addresses is called the **ODD BANK**.
- Generally for any 16 bit operation, even bank provides the lower byte and the odd bank provides higher byte. Due to which, **Even Bank is also known as Lower Bank and Odd Bank is also known as Higher**



ODD Bank

- Also Called as "Higher Bank"
- Address range:
  00001H
  00003H
  00005H
  .
  .
  FFFFFH
- Selected when $\overline{BHE} = 0$

EVEN Bank

- Also Called as "Lower Bank"
- Address range:
  00002H
  00004H
  00006H
  .
  .
  FFFFEH
- Selected when $A_0 = 0$

$\overline{BHE} = 0$	A_0	OPERATIONS
0	0	R/W 16 bit from both the banks
0	1	R/W 8 bit from Higher Bank
1	0	R/W 8 bit from Lower Bank
1	1	No Operation