What is a Microprocessor?

- The word comes from the combination micro and processor.
  - Processor means a device that processes whatever. In this context, processor means a device that processes numbers, specifically binary numbers, 0's and 1's.
    - To process means to manipulate. It is a general term that describes all manipulation. Again in this content, it means to perform certain operations on the numbers that depend on the microprocessor's design.

What about micro?

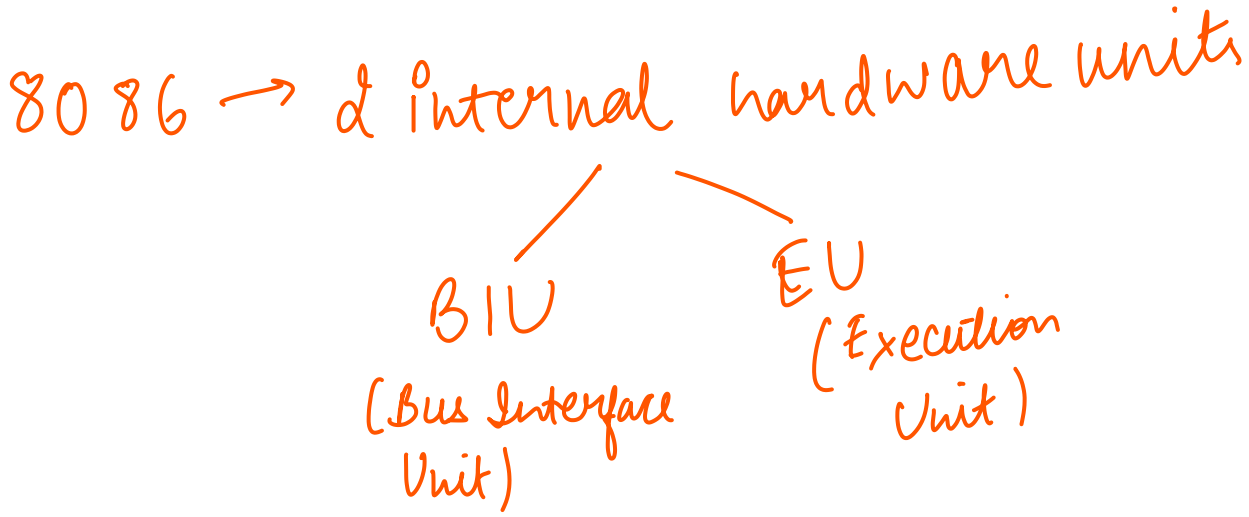
- Micro is a new addition.
  - In the late 1960's, processors were built using discrete elements.
    - These devices performed the required operation, but were too large and too slow.
  - In the early 1970's the *microchip* was invented. All of the components that made up the processor were now placed on a single piece of silicon.
  - The size became several thousand times smaller and the speed became several hundred times faster. The *"Micro"Processor* was born.

Was there ever a “mini”-processor?

- No.
  - It went directly from discrete elements to a single chip. However, comparing today’s microprocessors to the ones built in the early 1970’s, you find an extreme increase in the amount of integration.

Definition of the Microprocessor

- *Microprocessor* is a programmable device that takes in numbers, performs on them arithmetic or logical operations according to the program stored in the memory and then produces other numbers as a result.



Differences between:

- Microprocessor – silicon chip which includes ALU, register circuits and control circuits.
- Microcomputer – a computer with a microprocessor as its Central Processing Unit (CPU). Includes memory and Input/Output devices.
- Microcontroller – silicon chip which includes microprocessor, memory and Input/Output ports in a single package.

- A μP does not have enough memory for program and data storage, neither does it has any input and output devices. Thus when a μP is used to design a system, several other chips, such as memory chips and input/output ports, are also used to make up a complete microcomputer system.
- For many applications, these extra chips imply additional cost and increased size of the product and may not be suitable for the application.
- For example, when used inside a toy, a designer would like to minimize the size and cost of the electronic equipment inside the toy.
- Therefore, in such applications a microcontroller is used more often than a microprocessor. A microcontroller is a chip consisting of a microprocessor, memory and input/output ports.

A Microprocessor-based system

- From the above description, we can draw the following block diagram to represent a microprocessor-based system:

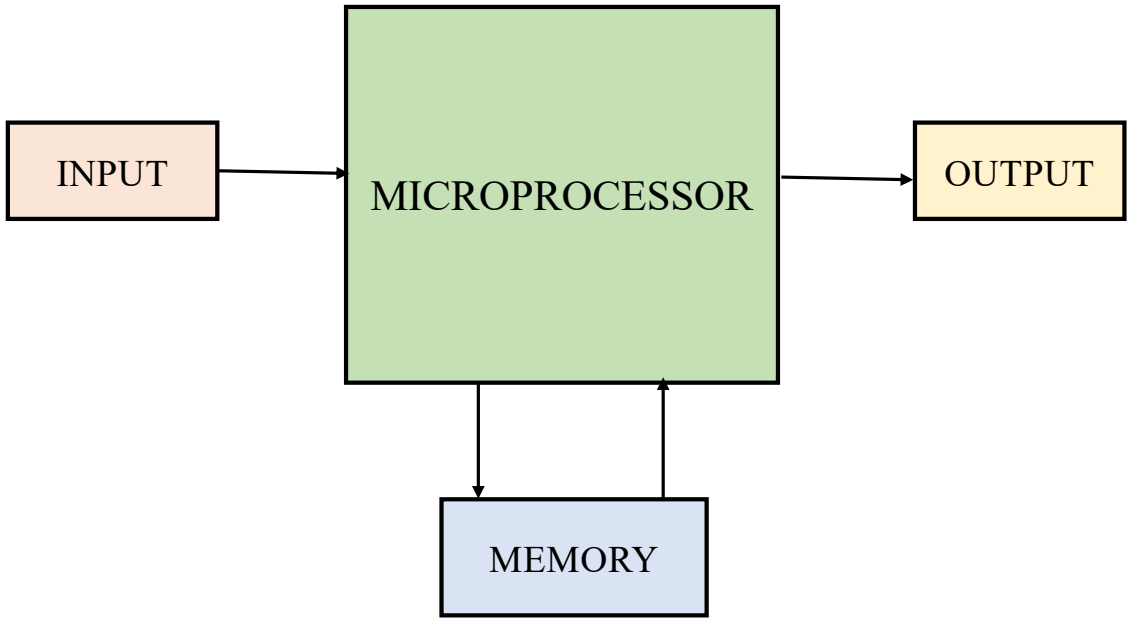


Figure 1.1: Microprocessor-based system

Inside the Microprocessor

- Internally, the microprocessor is made up of three main units.
  - The Arithmetic/Logic Unit (ALU).
  - The Control Unit.
  - An array of registers for holding data while it is being manipulated/executed.

ALU (Arithmetic and Logic Unit)

- This unit performs computing functions on m-bit data, where ‘m’ is the bit size of the processor.
- These functions are arithmetic operations such as addition, subtraction and logical operation such as AND, OR, XOR, rotate, compare etc.
- Results are stored either in registers or in memory or sent to output devices.

Register Unit

- It contains various 8-bit or 16-bit registers.
- These registers are used primarily to store data temporarily during the execution of a program.
- Some of the registers are accessible to the user through instructions. It means there contents can be read and/or changed through instructions.
- Some of the registers are not accessible to user, but they are used by the processor for the execution of an instruction.
- 8085A microprocessor contains 8-bit registers such as Accumulator (Reg. A), B, C, D, E, H, L etc. and16-bit registers such as Program Counter (PC), Stack Pointer (SP).

Control Unit

- It provides necessary timing and control signals required for the operation of microcomputer.
- It controls the flow of data between the microprocessor and peripherals (input, output & memory). The control unit gets a clock signal which determines the speed of the microprocessor.

In all, the CPU has the following basic functions:

a. It fetches an instruction word stored in memory.

b. It decodes the instruction to determine what the instruction is telling it to do.

c. It executes the instruction.

Organization of a microprocessor-based system

Let's expand the Figure 1.1.

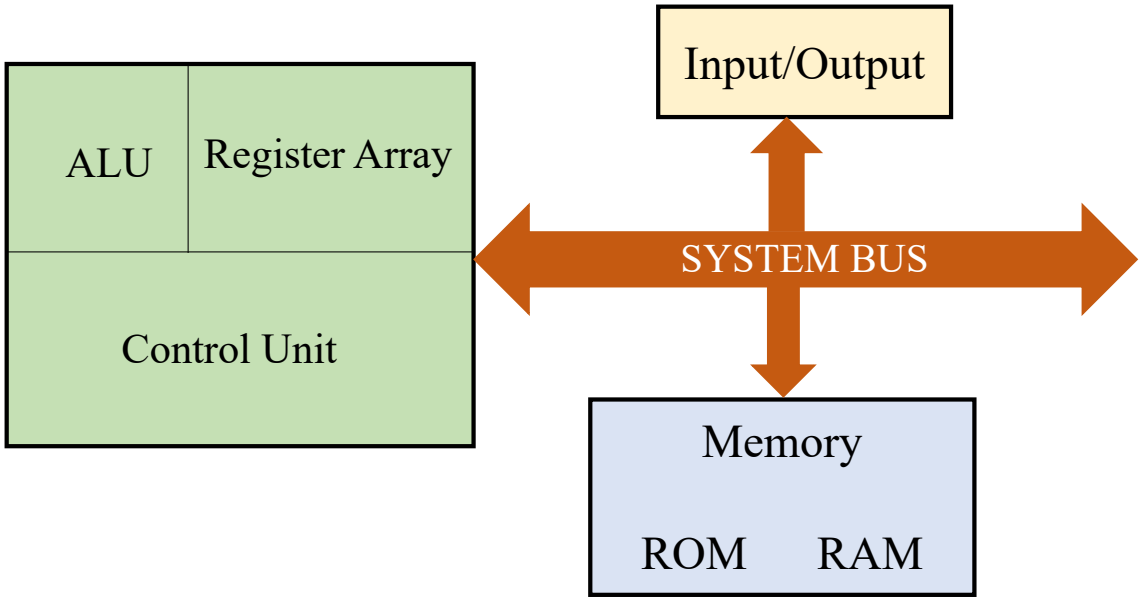


Figure 1.2: Organization of microprocessor-based system

Evolution of Microprocessors

First Generation:

1. The first μP was introduced in 1971 by Intel Corporation. This was the ***Intel 4004***, a processor on a single chip.
2. It had the capability of performing simple *arithmetic and logical operations*. For example, addition, subtraction, comparison, logical AND and OR operations.
3. It also had a *control unit* which could perform various control functions like fetching an instruction from the memory, decoding it and generating control signals to execute it.
4. It was a *4 bit μP* operating on 4 bits of data at a time. The processor was the central component in the chip set, which was called the MCS-4.
5. The other components in the set were a 4001 ROM, 4002 ROM and a 4003 shift register.

Contd….

- The first *8 bit μP*, which would perform arithmetic and logic operations on 8 bit words, was introduced in 1973, by Intel.
- This was *8008* that was followed by an improved version- the *8080* from the same company.
- They were designed using the PMOS technology. This technology provided *low cost*, *slow speed* and low output currents.

Second Generation:

- After 1973, the *second generation* μPs such as *Motorola 6800* and *6809*, *Intel 8085* and *Zilog Z80* evolved.
- These μPs were fabricated using *NMOS technology*. The NMOS process offered *faster speed* and *higher density* than PMOS.

Contd….

Third Generation:

- After 1978, the 3rd generation microprocessors were introduced.
- Typical µPs were *Intel 8086/ 80186/ 80286* and *Motorola 68000/ 68010*. These µPs were designed using *HMOS technology*. HMOS provides the following advantage over NMOS. 1) Speed power produced (SSP) of HMOS is 4 times better than that of NMOS.
- Later, Intel introduced a high speed version of the 8085A called *8085AH* using HMOS technology to fabricate the 8085A.
- One of the most popular *16-bit µP* introduced by Intel was *8088*.

Contd….

Fourth Generation:

- In 1980, the *fourth generation µPs* were evolved.
- Intel introduced the *first commercial 32 bit microprocessor*, *Intel 432*.
- Since 1985, more 32-bit µPs have been introduced. These include *Intel iAPX80386, Intel 80486, Motorola MC68020/68030/68040, National semiconductor NS 32032*.
- These processors were fabricated using the low power version of HMOS technology called *HCMOS*, and they include an on-chip RAM called the *cache memory* to speed up program execution.

Table 1.1: Evaluation of major microprocessor characteristics from Intel

	4004	**8008**	**8085A**	**8086**	**80386**
Year of Introduction	1971	1971	1977	1978	1985
Data Bus	4-bit	8-bit	8-bit	16-bit	32-bit
Technology	PMOS	PMOS	NMOS	HMOS	CHMOS
Word size data/ instr.	4/8	8/8	8/8	16/16	32/32
Address capacity	4K	16K	64K	1M	4G
Clock kHz/phase	740/2	800/2	6250/2	8000/2	16000/2
Addition time	10.8 μs	20 μs	1.3 μs	0.375 μs	0.125 μs
ALU/General Purpose Reg.	1/16	1/6	1/6	1/8	1/8
Stack size	3x12	7x14	RWM	RWM	RWM
Voltages	15-10,5*	-9.5v	+5V	+5V	+5V
Package size	16pin	18pin	40pin	40pin	132pin
Instructions	45	48	74	133	135
Transistors	2,300	2,000	6,200	29,000	2,75,000
Chip size(mil)	117x159	125x170	164x222	225x230	390x390

Applications of Microprocessors

1. Analytical scientific instruments
2. Smart terminals
3. Stacker crane controls
4. Conveyor controls
5. Standalone electronics cash system
6. Electronic games
7. Vending and dispensing machines
7. Market scales
8. Traffic light controls
9. Home heating and lighting controls
10. Security & fire alarm system
11. Home appliances
12. Computer aided instruction
13. Desktop computers
14. Payroll system
14. Automobile diagnostics
15. Data communication processing
16. I/O terminal for computers.

Main Features of 8086

1. Enhanced version of 8085 microprocessor.
2. Designed by Intel in 1976.
3. It is a *16-bit* processor.
4. It has a *16-bit data* bus, $D_0 - D_{15}$.
5. 8086 has a *20-bit address* bus, $A_0 - A_{19}$, which means it can address up to 2^{20} memory locations.
6. It can read/write data to a memory (or) port either 16-bit (or) 8-bit at a time.
7. It uses a *40-pin* dual in line package.
8. Frequency range of 8086 is 6-10 MHz

Main Features of 8086

9. It requires *+5V* power supply.

10. It has powerful instruction set that supports MUL and DIV operations.

11. Designed to operates in two modes:

Minimum mode: system having single processor

Maximum mode: system having multiple processors

12. It consists of *29,000 transistors*.

13. It has *256 vectored interrupts*.

14. Supports two stage pipelining (Fetch and Execute stages)

Main Features of 8086

14. Address and data lines are multiplexed

$AD_0 - AD_{15}$

$A_{16} - A_{19}$

15. Instruction system byte queue

It can *pre-fetch* up to *six instruction bytes* from memory and queues them in order to speed up instruction execution.

16. 8086 has two blocks – BIU and EU.

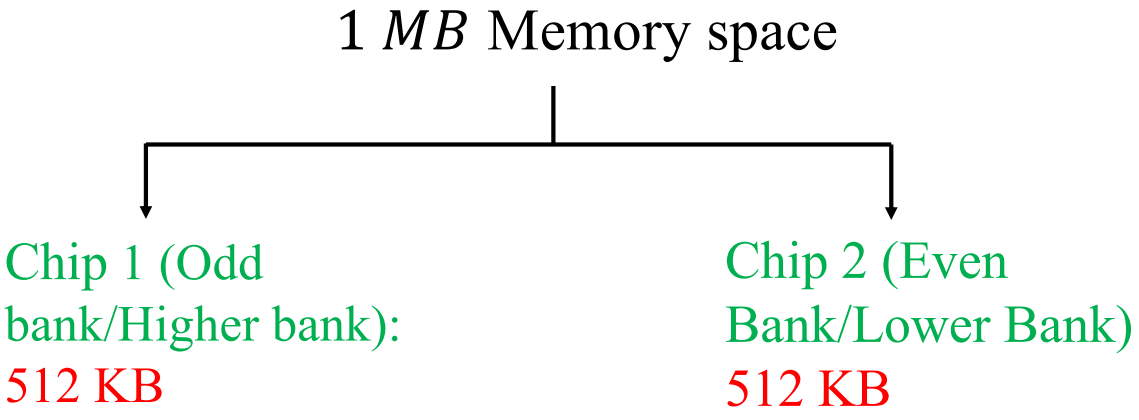
BIU – performs bus operations like instruction *fetching*, *reading/writing operands* for memory and *calculating addresses* of memory operands, *prefetch* up to 6 instruction bytes.

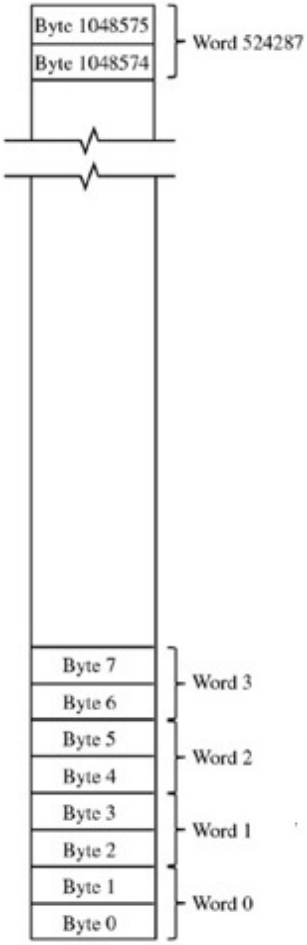
EU – executes instructions from the instruction system byte queue.

Memory Banking

- 20-bit address bus
- Memory space of 8086 = $2^{20} = 1\ MB$
- *Figure 1.4* shows memory space of the 8086

consisting of *1,048,576 bytes* or *524,288 16-bit words*.





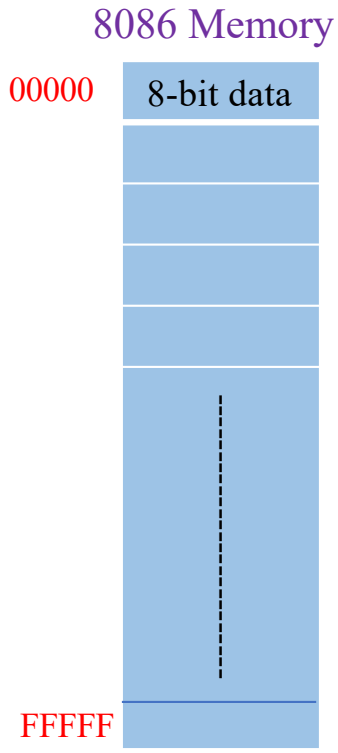


Fig. 1.3: Memory space of 8086

Fig. 1.4: Memory space of the 8086 consisting of 1,048,576 bytes or 524,288 16-bit words.

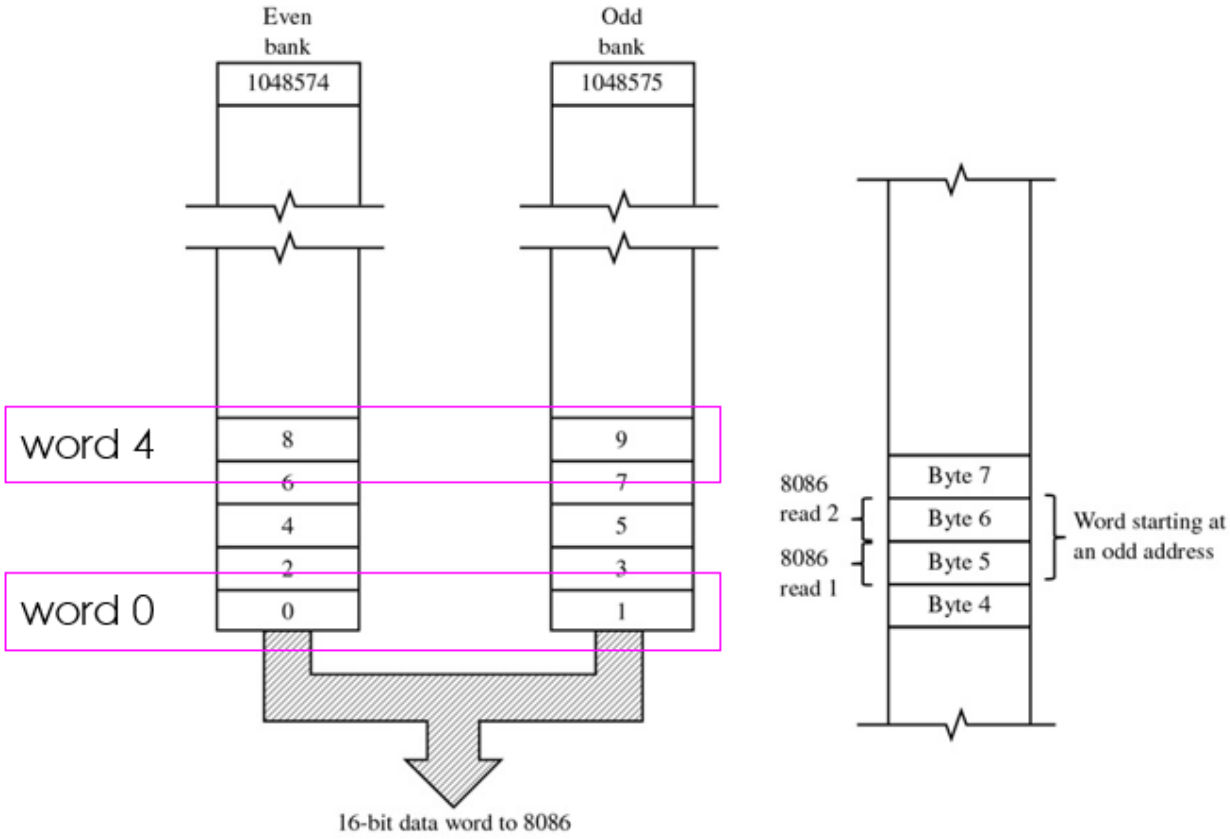
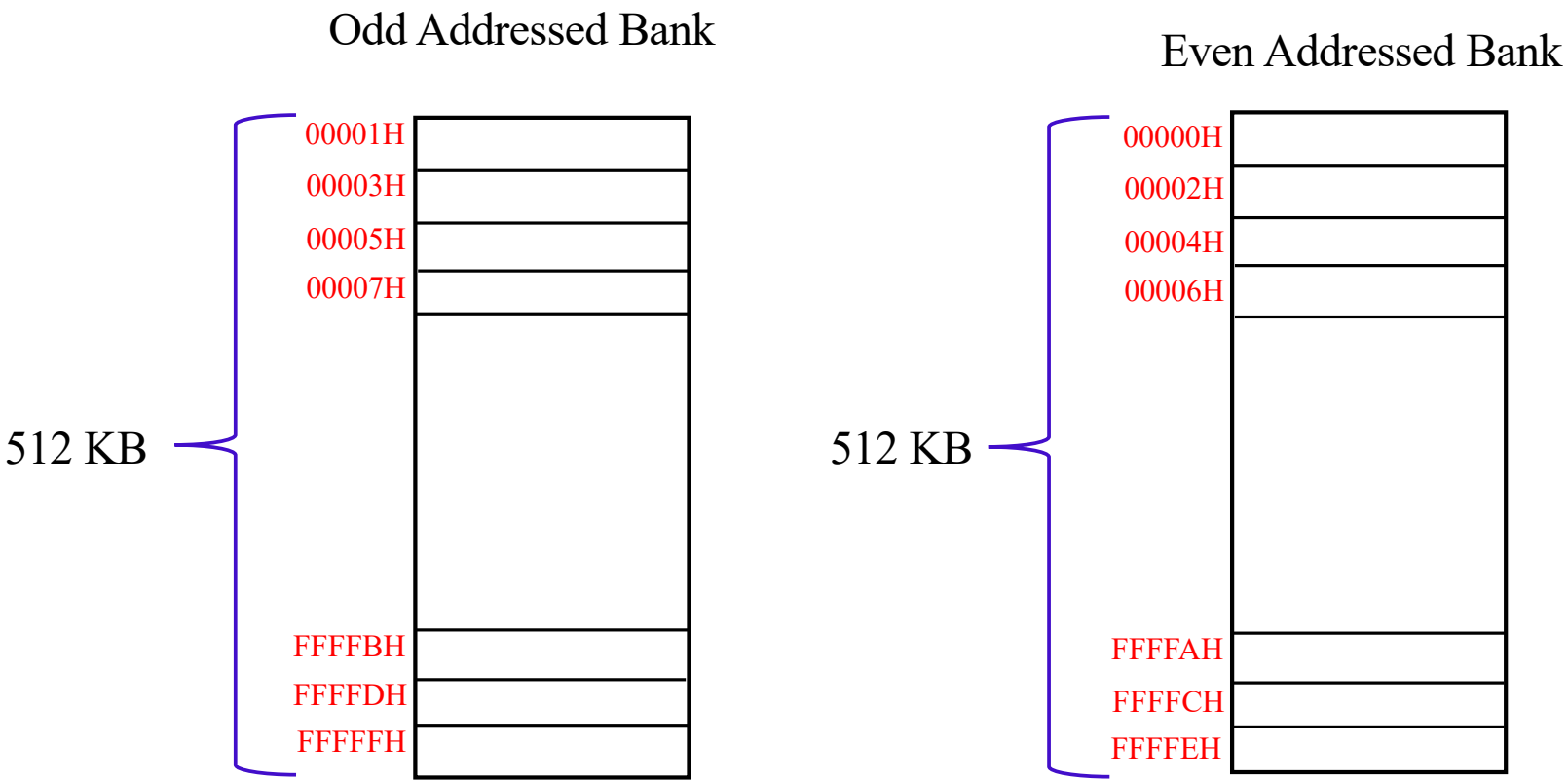
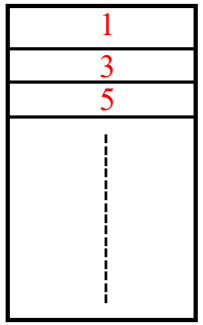
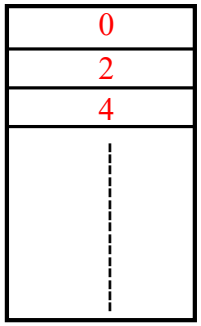


Fig. 1.5: (a) By reading from an even-addressed bank and an odd-addressed bank the 8086 can read two bytes from memory simultaneously. (b) If the 16-bit word begins at an odd address, the 8086 will require two memory read or write cycles.


Odd Addressed Bank
512 KB
00001H
00003H
00005H
00007H
FFFFBH
FFFFDH
FFFFFH
Even Addressed Bank
512 KB
00000H
00002H
00004H
00006H
FFFFAH
FFFFCH
FFFFEH

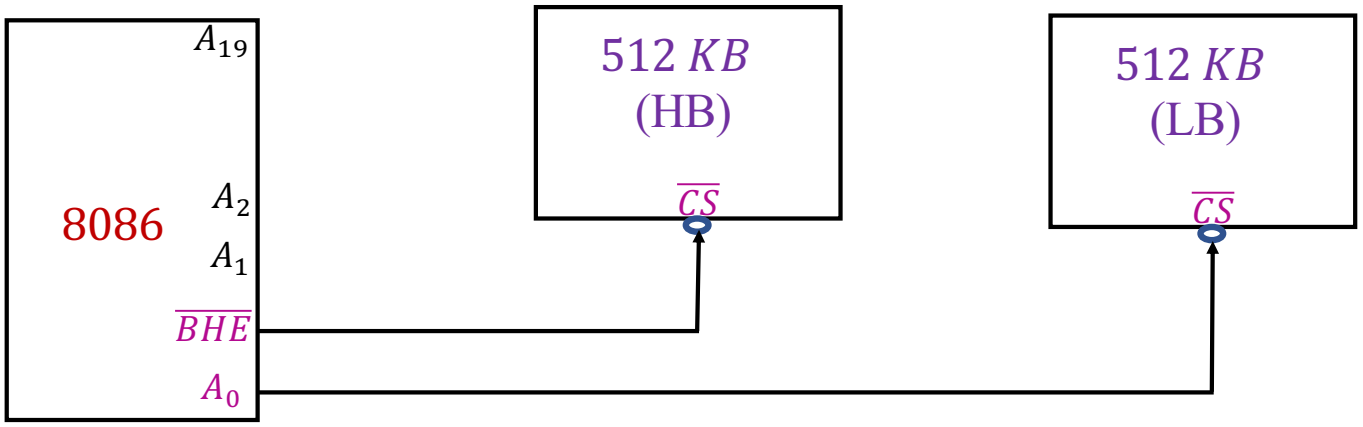




So, A_0 bit will be used to decide which *chip* is to be selected. $A_{19} \text{---} A_3\ A_2\ A_1$ will be used for *address (19-bits)*

Address = 20-bits

A_{19} ---------- A_4	A_3	A_2	A_1	A_0	Address
0 ---------- 0	0	0	0	0	$00000H$
0 ---------- 0	0	0	0	1	$00001H$
0 ---------- 0	0	0	1	0	$00002H$
0 ---------- 0	0	0	1	1	$00003H$



Aim is to Access:

1. 16-bits completely
2. only lower 8-bits
3. Only higher 8-bits

$\overline{BHE}$	A_0	Memory Operation	
0	0	Access 16-bit data	$D_{15} - D_0$
0	1	Access higher 8-bits	$D_{15} - D_8$
1	0	Access lower 8-bits	$D_7 - D_0$
1	1	Idle	

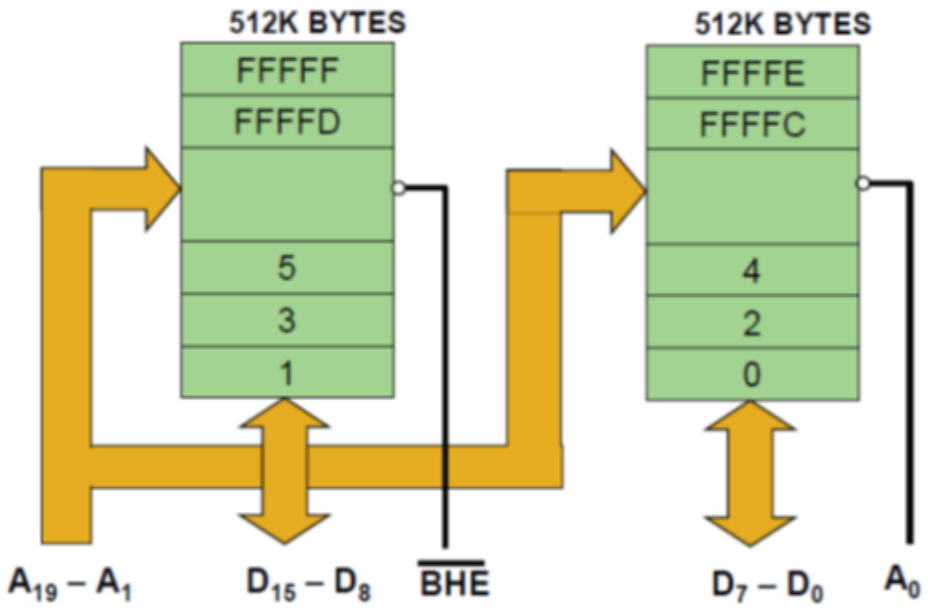
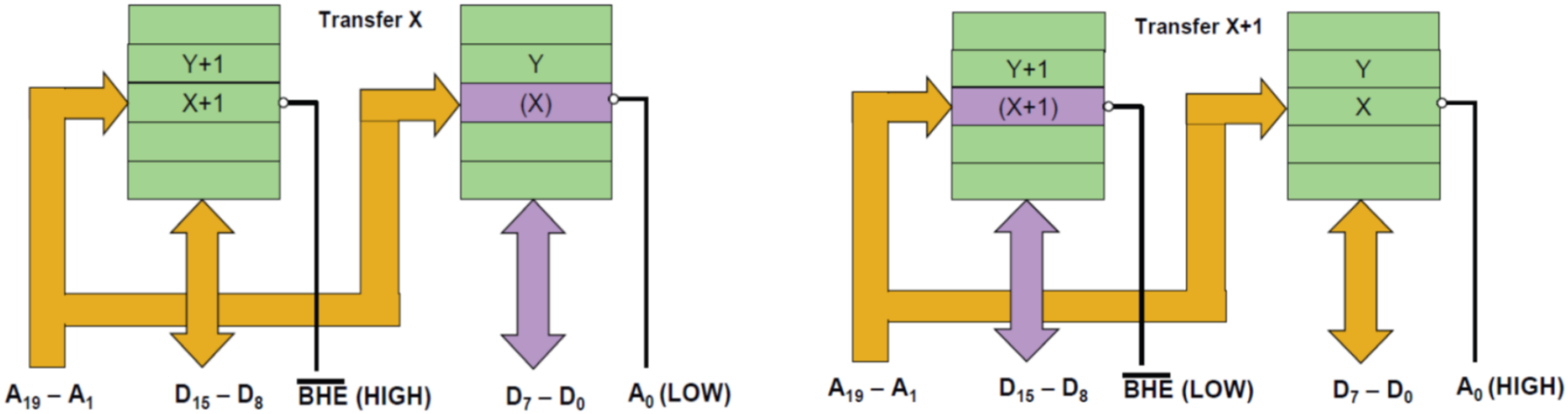


Fig. 1.6: Higher and lower memory banks of 8086

* memory reading is done first in lower/even bank → in one cycle 2 bytes can be accessed simultaneously.

* If start reading from odd level 1 → (1&2) consecutive → but at diff. levels ↓ 2 cycles reqd.



(a) (b)

Fig. 1.7: (a) Even address byte transfer by 8086 and (b) Odd address byte transfer by 8086.

Aligned Word

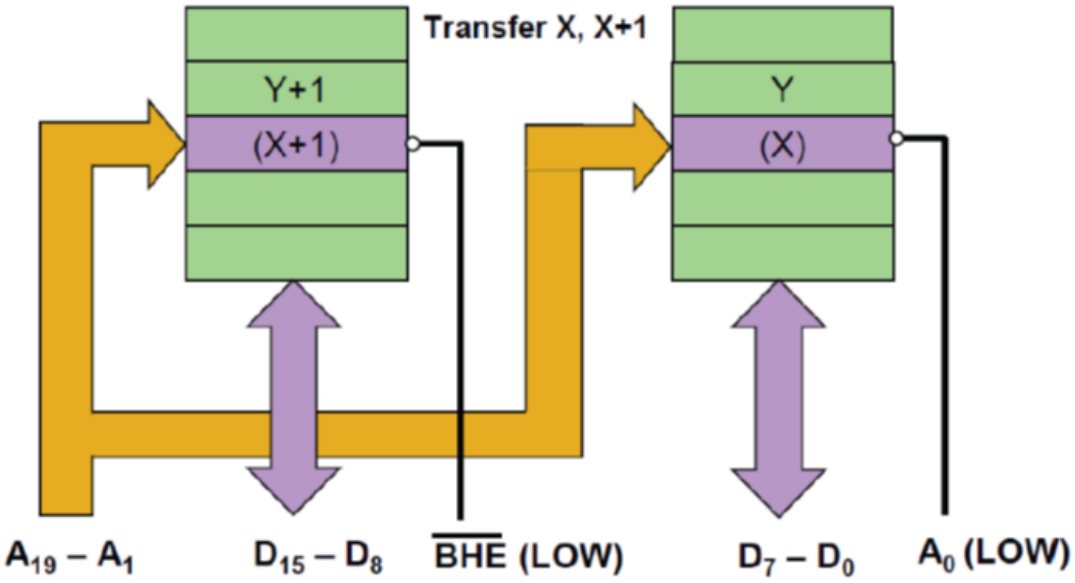


Fig. 1.8: Even address word transfer by 8086.

Non-Aligned Word

→ suppose we want to read 1 & 2 consecutively but not at same level, → reading data starts from odd bank

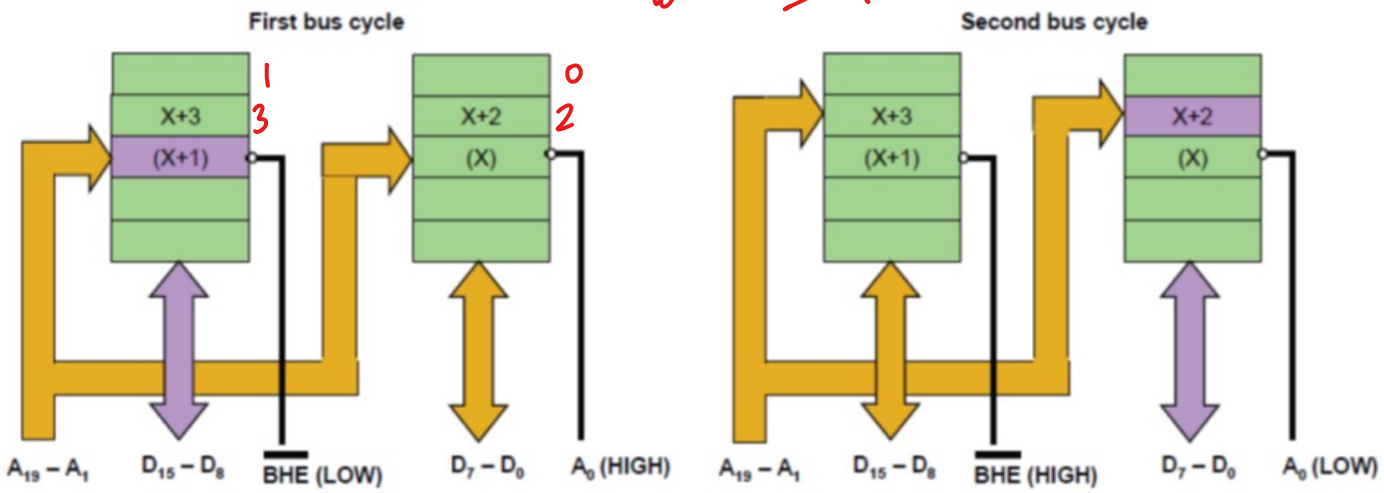
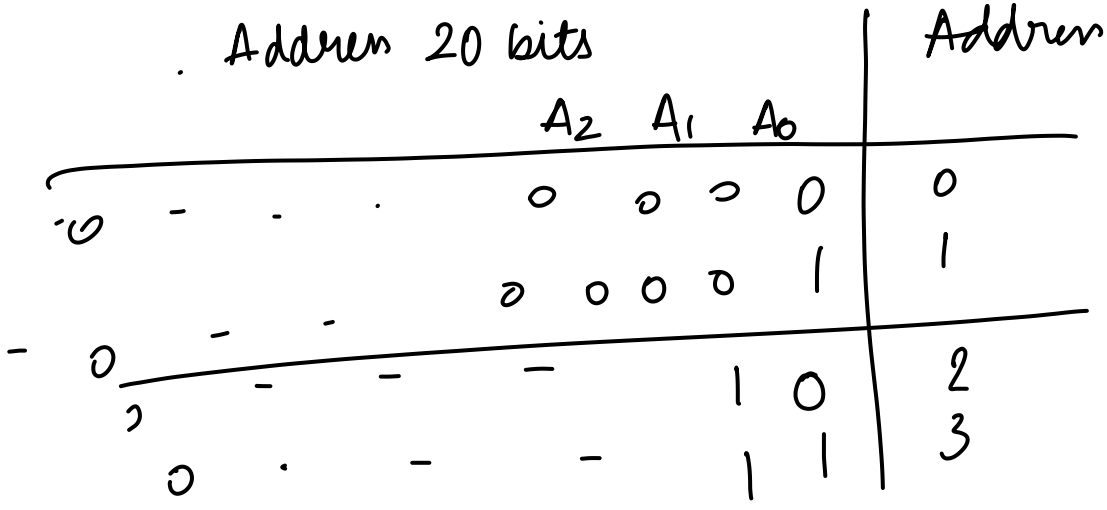


Fig. 1.9: Even address word transfer by 8086.

- Two bus cycles are needed.
- During the first bus cycle, byte of word located at *address X + 1* in the high bank is accessed over *D8 through D15*.
- Even though the data transfer uses data lines D8 through D15, to the processor it is the low byte of the addressed data word.
- In second memory bus cycle, the even byte located at *X + 2* in the low bank is accessed over bus lines *D0 through D7.*

- Aligned operand
  - Operand aligned at even-byte boundaries.
  - Allows single access to read/write one operand.

- Mis-aligned words
  - Word operand does not start at even address.
  - Need 2 read cycles to read/write the word (8086)
    - Issues two addresses to access the two even-aligned words containing the operand in order to access the operand.
    - slower but transparent to programmer.



A0 bit for 2 consecutive addresses is different
↓
A0 bit can be used to select the memory bank.

→dual inline package (DIP)

Pin Diagram of 8086

- The original chip measured 33 mm² and minimum feature size was 3.2 µm.
- *40 pin IC* with dual in-line packing (DIP).
- Operates in two mode: *MAX* and *MIN*.
- *Two stage* pipelining (Fetch and Execute).
- Common pins/signals: 32
- Minimum mode: 8 pins → $Pins = 24 - 31$
- Maximum mode: 8 pins → $Pins = 24 - 31$

MIN MODE MAX MODE

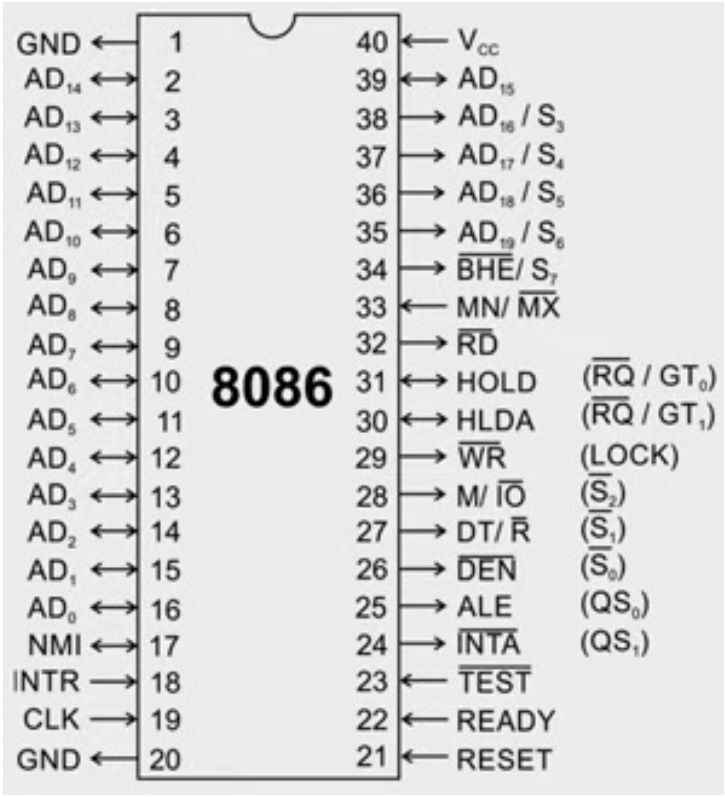


Fig. 1.10: Pin diagram of 8086.

Pin 40 - Vcc
2 ground pins - 1 & 20 (to reduce / nullify noise)
20 address lines, 16 data lines multiplexed.
Higer 4 bits of multiplexed with status signals → multiplexed Address Status Bar

Pin Diagram of 8086

- Power supply, $V_{cc} = +5V \rightarrow Pin = 40$
- Two Ground, GND $\rightarrow Pins = 24\ and\ 1$ (to cancel the effect of noise if any).
- 20 address lines and 16 data lines.

The lower order 16 address lines are multiplexed with 16 data lines and available as $\mathrm{AD_{15} - AD_0}$ $\rightarrow$ Multiplexed Address/Data Bus $\rightarrow Pins = 2 - 16\ and\ 39$.

With multiplexing – number of pins reduces → Area reduce → Power reduces

Function of microprocessor not affected by multiplexing of address and data lines.

Pin Diagram of 8086

When does the multiplexed bus is used to carry address and when carry data?

- During the *first T state* i.e. *T1 state*, address will be carried on the multiplexed bus.

1 clock period = 1 - T state

- 8086 bus cycle takes *four T states.*
- In the first T state i.e. T1, $AD_{15} - AD_0$ used to carry address and act as $A_{15} - A_0$
- So, the first step in any of the microprocessor initiated operations (i.e. I/O read, I/O write, Mem. Read and Mem. Write) is generating address.
- Remaining 4 address lines are multiplexed with status signals

$A_{19}/S_6 - A_{16}/S_3 \rightarrow$ Multiplexed Address/Status Bus $\rightarrow Pins = 35 - 38$.

Pin Diagram of 8086

- So, during T1 state, a total of 20 address lines are available.
- After the first step, control signals will be generated followed by sending the data.
- So, in subsequent T states, $AD_{15} - AD_0$ will be used to carry data, i.e. $D_{15} - D_0$ and $A_{19}/S_6 - A_{16}/S_3$ will be used to carry status signals.

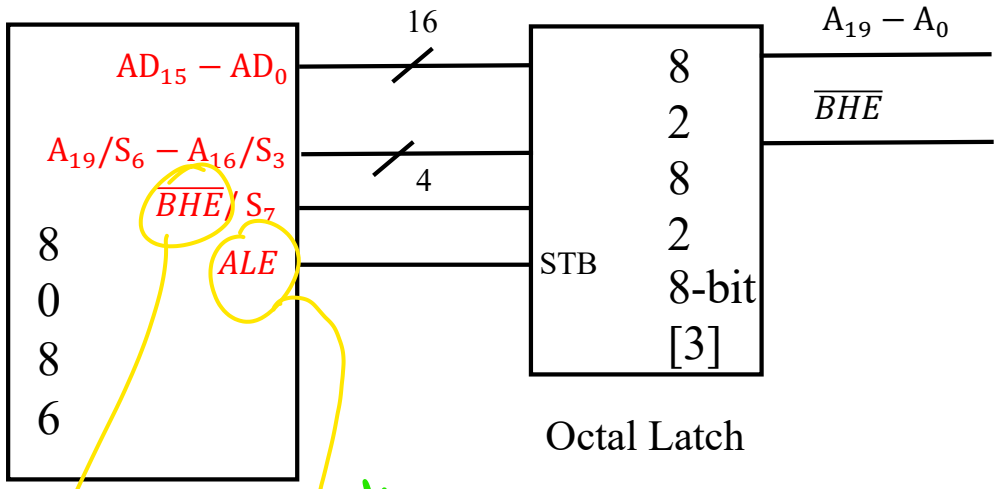
How this will latched?

- Signal *Address Latch Enable* (ALE) will be used → $Pin = 25$

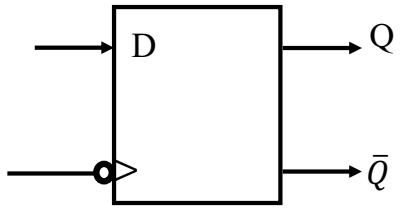
Address Latch Enable (ALE): used to demultiplex the address/data bus and address/status bus. Enables the latching of address.

Latch: clocked D flip-flop acts like a latch.

Pin Diagram of 8086



Octal Latch



D = Q at negative edge triggered of clock
So, this is latching of input data to the output.

Bus High Enable $\overline{BHE}$ & A_0 work together to

1. Select entire 16 bits
2. Select lower 8 bits
3. Select higher 8 bit

$\overline{BHE}$	A_0
0	0
0	1
1	0

Pin Diagram of 8086

- To provide frequency to the Microprocessor (MP) – MP provided with CLK pin → $Pin = 19$

IC 8284 is connected externally to 8086 – clock generator

Crystal oscillator used – are more stable

The crystal oscillator of IC 8284 generates a frequency of 15 MHz and in combination with DIV by 3 Counter generates a frequency of *5 MHz* for 8086.

↓ ↑stable

PIN-19 → clock – [8284 IC ↓ uses crystal oscillator (Piezo electric)

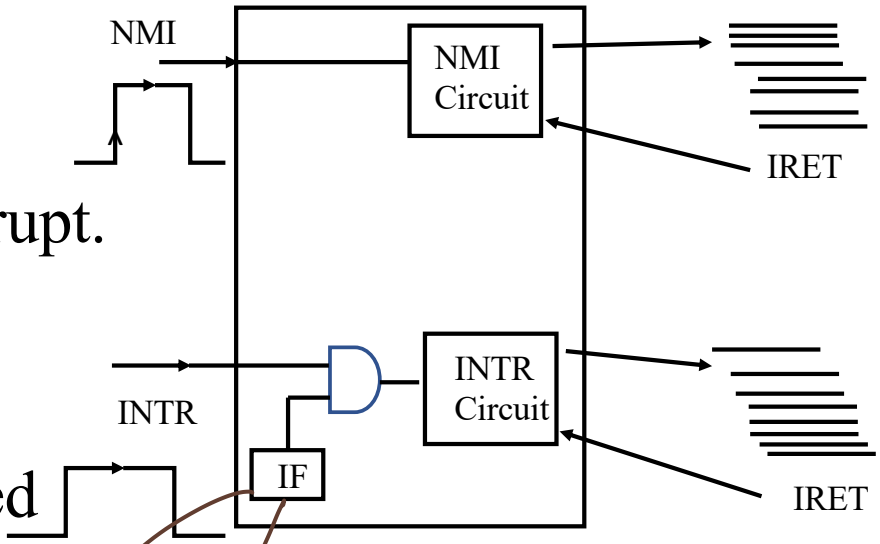
8086 – standard frequency = 5MHz

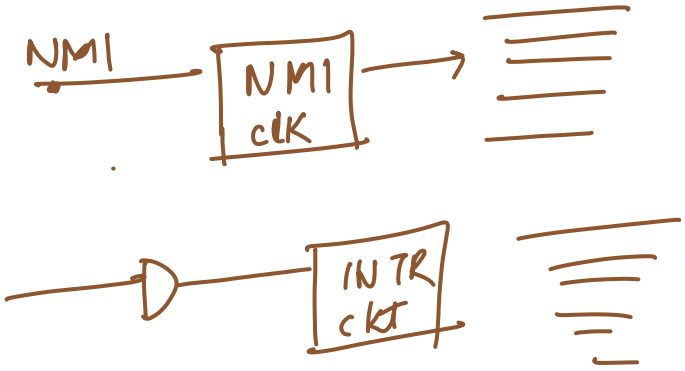
8086 (2) – standard frequency = 8MHz

8086 (1) – standard frequency = 10 MHz

NMI (Non-Maskable Interrupt) and INTR (Interrupt Request)

- In 8086, two pins for interrupts NMI and INTR - these are *hardware* interrupts → $Pin = 17\ and\ 18, respectively$.
- NMI – positive edge high-level triggered.
- Its priority higher then the second interrupt.
- INTR – high-level triggered.
- It is a lower priority interrupt.
- It is maskable interrupt – can be disabled





decides if INTR has to be serviced or not

interrupt flag (1 bit) usually maintained at low → INTR is disabled

- Interrupt flag (IF) is a bit of Flag Register.
- IF is set to zero (0) always, hence it is disabled always.
- If IF = 1 and there is interrupt at INTR pin, INTR circuit activated.
- INTR interrupt should be accessed or not, decided by IF.
- To enable IF (IF=1), the instruction used is *STI* (set interrupt flag)
- To disable IF (IF=0), the instruction used is *CLI* (clear interrupt flag).

active low signal (PIN 24) → signal used by µp to acknowledge the device request interrupt

$\overline{\text{INTA}}$: Interrupt Acknowledgment signal → $Pin = 24$

- This pin is used in Minimum Mode operation of MP.

eg. MR

Four microprocessor initiated operations →

MR	MW	IOR	IOW

MR → memory read

IOR → I/O read

* For any of these operations —
1) Address of device from which to read
2) Control signal for read operation
3) Now data will be transmitted to data lines

only 1st cycle require address

S_6 → always at logic 0 (when up gives away control of bus it becomes high impedance)

Address/Status Bus:

$A_{19}/S_6 - A_{16}/S_3 \rightarrow Pin = 35 - 38$

$S_3 - S_6$

S_6 *is low (logic 0) – when 8086 uses the buses*

S_6 in high impedance state – when 8086 bus control given to Memory

S_4	S_3	Operation
0	0	Extra segment
0	1	Stack segment
1	0	Code or no segment
1	1	Data segment

S_5 represents the status IF Flag Register

S_7 - low during interrupt acknowledgment signal (INTR)

HOLD $\rightarrow Pin = 31$ and HLDA $\rightarrow Pin = 30$

HOLD and HLDA used for *Direct Memory Access (DMA)*

- When Memory wants to interacts with the I/O devices directly, it requires the control of MP buses. It requests the control of MP buses by sending HOLD signal.
- Acknowledgment signal sent by MP – *HLDA*

memory has no bus ↓ during DMA memory request access/control of system bus (data bus & address bus)

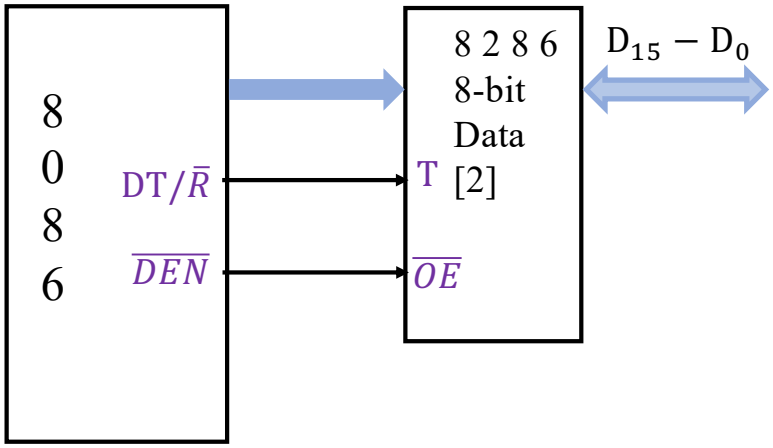
used for DMA operation

$DT/\bar{R} \rightarrow Pin = 27$ (default 0) → used to read/write

- Used in Minimum Mode operation.
- 8086 can read/write data from an external device *8286* called as *Transceiver* → combination of Transmitter and Receiver.

$DT/\bar{R} = 1 \Rightarrow$ Data transmitted from MP $\Rightarrow$ Write operation

$DT/\bar{R} = 0 \Rightarrow$ Data received from peripheral $\Rightarrow$ Read operation



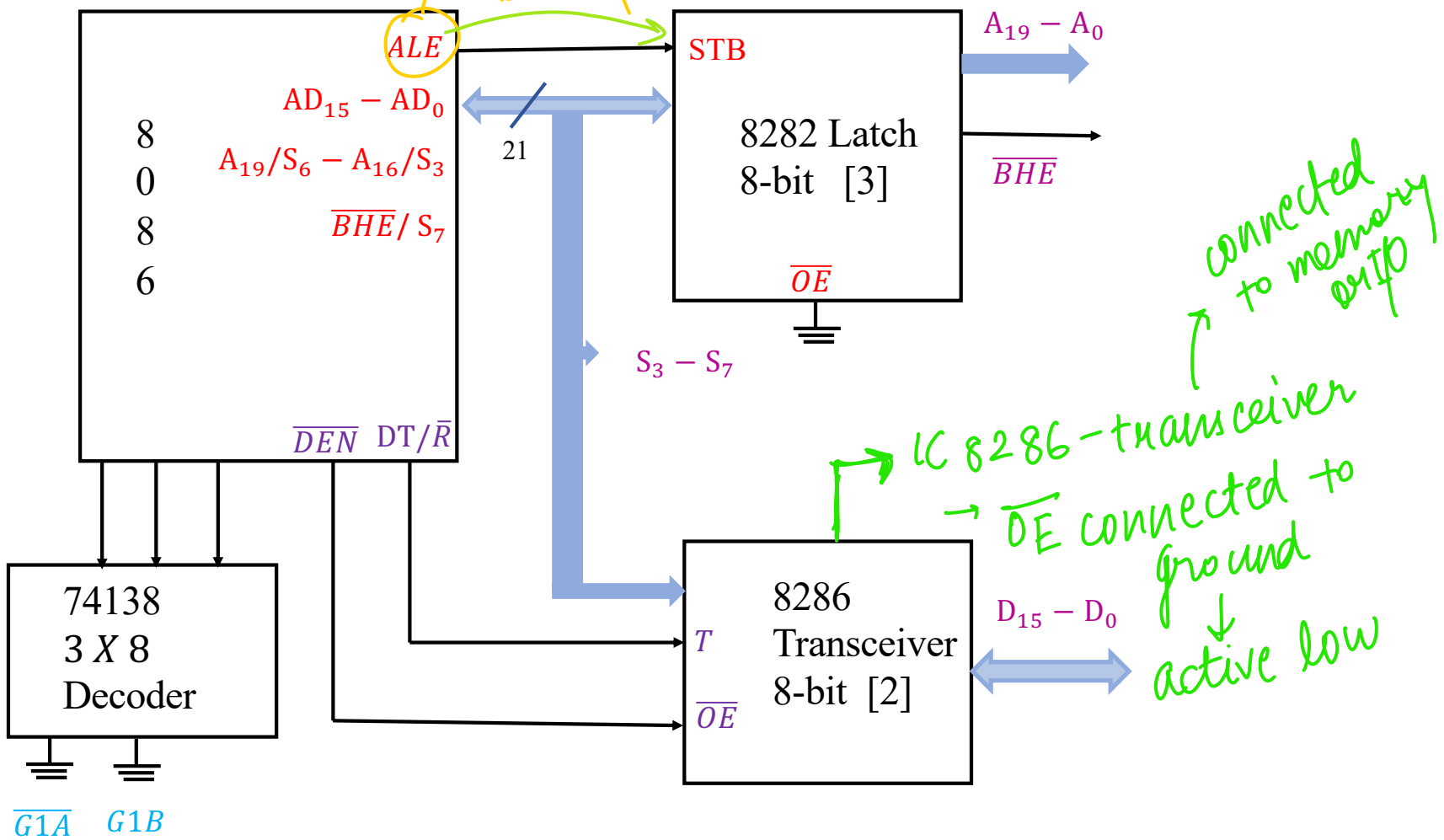
- **DT/R :** Data Transmit/Receive. This pin required in minimum systems, that want to use an 8286 or 8287 data bus transceiver. The direction of data flow is controlled through the transceiver.
- **DEN :** Data enable. This pin is provided as an output enable for the 8286/8287 in a minimum system which uses transceiver. DEN is active low(0) during each memory and input-output access and for INTA cycles

→ 2nd clock cycle → used (R=1) to activate transreceiver

- ALE is used to demultiplex address-data & address status bus.
- ALE is high in first clock cycle, then low.

8282 IC is used to latch

ALE is connected to STB of 8282

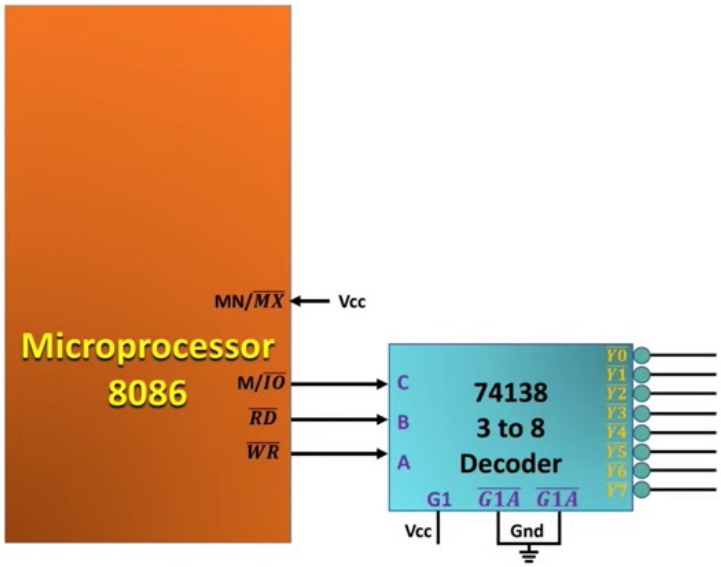


4 up generated operations, 3 pins – 4 control signals

Control Signal Generation (Minimum Mode)

Three pins $\overline{RD}$, $\overline{WR}$ and M/$\overline{IO}$ → $Pin = 32, 29\ and\ 28$, respectively used to generate four control signals.

M/$\overline{IO}$	$\overline{RD}$	$\overline{WR}$	Operation
0	0	1	$Y1 = \overline{IOR}$
0	1	0	$Y2 = \overline{IOW}$
1	0	1	$Y5 = \overline{MEMR}$
1	1	0	$Y6 = \overline{MEMW}$



$MN/\overline{MX} \rightarrow Pin = 33$ → to decide mode

- When $MN/\overline{MX}$= 1; Minimum Mode activated (No coprocessors connected). $MN/\overline{MX}$ connected to V_{CC}
- When $MN/\overline{MX}$= 0 (connected to GND); Maximum Mode activated (8086 connected to coprocessors).

READY $\rightarrow Pin = 22$

- This pin is used to insert Wait state into the timing cycle of 8086.
- If READY = 1, no effect on operation. Data from peripheral devices is ready to be read.
- If READY = 0, 8086 enters into Wait state like idle.
- It is used to synchronize slow peripheral devices.

$\overline{TEST} \rightarrow Pin = 23$ → kab tak wait state m rahega

WAIT instruction. MP will WAIT until $\overline{TEST} = 0$

RESET → this pin has to be high for min 4 clock cycles for µp to be RESET

- This causes processor to immediately terminate its present activity.
- The signal must be active HIGH for at least four clock cycles.

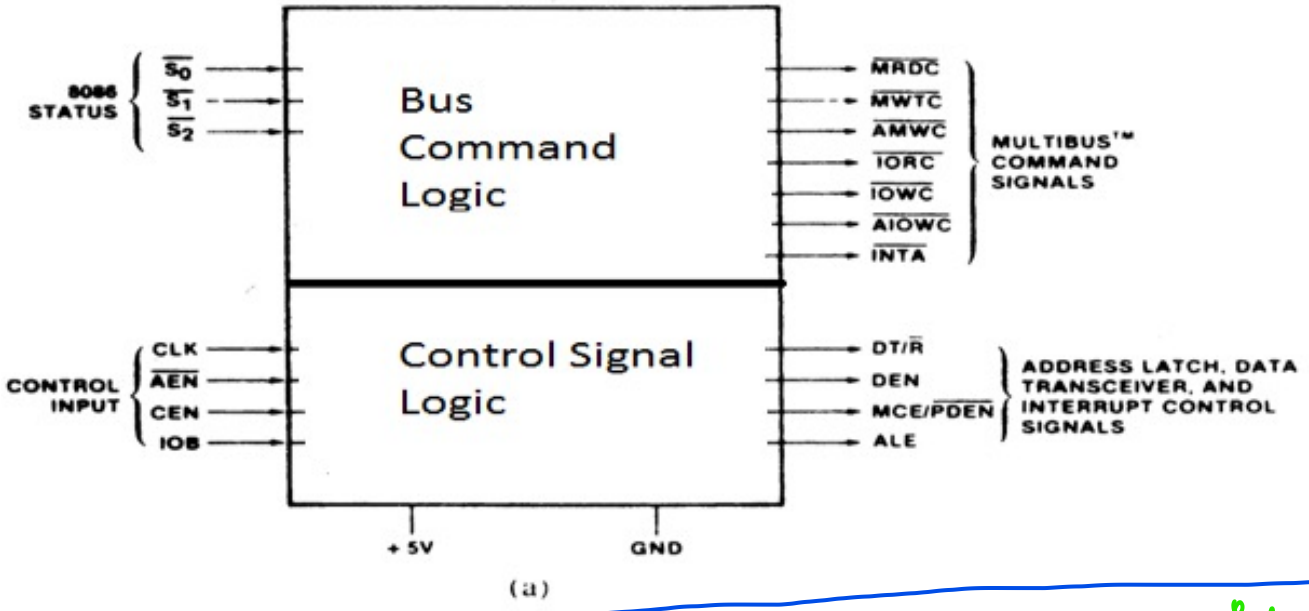
Maximum Mode Signals

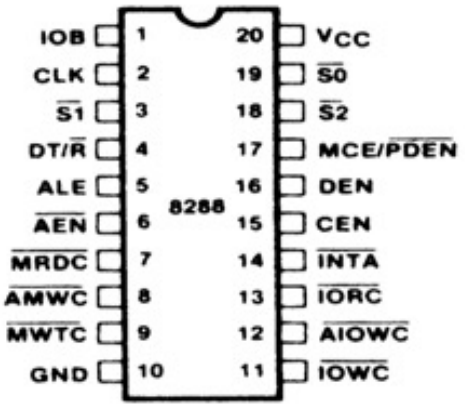
- Here, either a numeric coprocessor of the type 8087 or another processor (8089) is interfaced with 8086.
- The Memory, Address Bus, Data Buses are shared resources between the two processors.
- The control signals for Maximum mode of operation are generated by the *Bus Controller chip 8288.* → bus controller IC (20 pins)
  ↳ provides those signals that are eliminated during maximum mode.
- The three status outputs $\overline{S_0}$, $\overline{S_1}$ *and* $\overline{S_2}$ from the processor are input to 8288.
- The outputs of the bus controller are the Control Signals, namely DEN, DT/R*, IORC*, IOWTC*, MWTC*, MRDC*, ALE etc.

AMWC → advanced memory read/write

AIOWC → " " IO " "

} Prepare I/O device in advance, prevents delay





MOV AL, BL 1000

ADD AL, CL 1001

PC = 1001 [interrupt while executing MOV]

↓

processor will first complete the execution of current inst.

↓

Now PC → ISR and PC goes to stack pointer

↓

PC comes back to ADD after servicing the interrupt. (stack pointer)

In case of Bus request the processor will not complete the entire inst. cycle, it will complete the machine cycle.

LOCK MOV CX,[1800] → This is an important inst. & we don't want disturbances
↳ we need to stop bus request

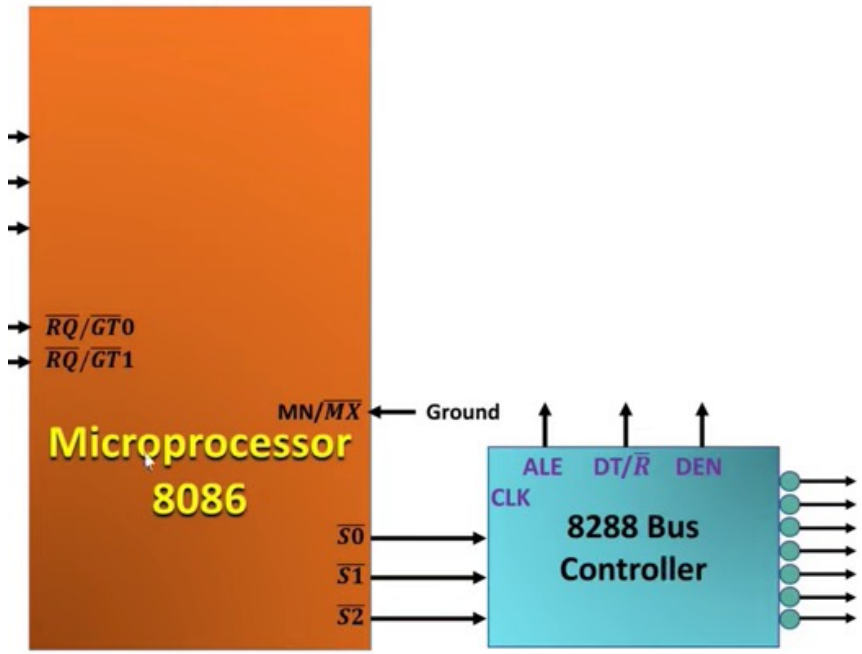
inst. prefix

String inst ← Repeat Escape

[3 inst. prefix for 8086]

$\overline{S_0}, \overline{S_1}$ and $\overline{S_2} \rightarrow Pin = 26, 27 \; and \; 28$, respectively.

LOCK → LOCK = 0



$\overline{S_2}$	$\overline{S_1}$	$\overline{S_0}$	Function
0	0	0	Interrupt acknowledgment
0	0	1	I/O read
0	1	0	I/O write
0	1	1	*Halt*
1	0	0	*Instruction fetch*
1	0	1	Memory read
1	1	0	Memory write
1	1	1	Not used

min

→ 4 T states in 1 Machine cycle. 1st T state → ALE (address is sent)

2nd clock cycle → Bus reserved for data transmission. $\overline{Read} = 0$

$M/\overline{IO} = 1$ → if op is done from memory or I/O device.

→ if read from memory → delay is there → peripheral devices are slow.

if write op. was there, then there would have been no delay.

→ $\overline{DEN}$ is connected to $\overline{OE}$ of transciever of $\overline{OE}$ is low then transceiver is enabled and it will receive data now. → enabled in 2nd clock cycle.

in max mode only diff is that during T1 instead of ALE $\overline{S0}$, $\overline{S1}$ & $\overline{S2}$ are active.

Queue Status signals

QS_1 and QS_0 → Pin = 24 and 25, respectively

QS = Queue status

- As discussed, 8086 contains *6 byte Instruction Queue.*
- So, by using this Instruction Queue, we can overlap Fetch and Execute stages and increase the speed of the operation called Pipelining.

QS_1	QS_0	Operation
0	0	Read the first byte (opcode fetch) from queue
0	1	Empty the queue
1	0	*Not used*
1	1	Read subsequent bytes from the queue

- The status of the Queue is indicated by these two pins.

Status Inputs			CPU Cycles	8288 Command
$\overline{S}_2$	$\overline{S}_1$	$\overline{S}_0$		
0	0	0	Interrupt Acknowledge	$\overline{INTA}$
0	0	1	Read I/O Port	$\overline{IORC}$
0	1	0	Write I/O Port	$\overline{IOWC}$, $\overline{AIOWC}$
0	1	1	Halt	None
1	0	0	Instruction Fetch	$\overline{MRDC}$
1	0	1	Read Memory	$\overline{MRDC}$
1	1	0	Write Memory	$\overline{MWTC}$, $\overline{AMWC}$
1	1	1	Passive	None

AIOWR

- Advanced I/O write is a command output to an advanced I/O write control signal.

IORD

- The I/O read command output provides I/O with its read control signal.

IOWR

- The I/O write command output provides I/O with its write control signal.

AMWR

- Advanced memory write control pin provides memory with an early/advanced write signal.

MWR

- The memory write control pin provides memory with its normal write control signal.

MRD

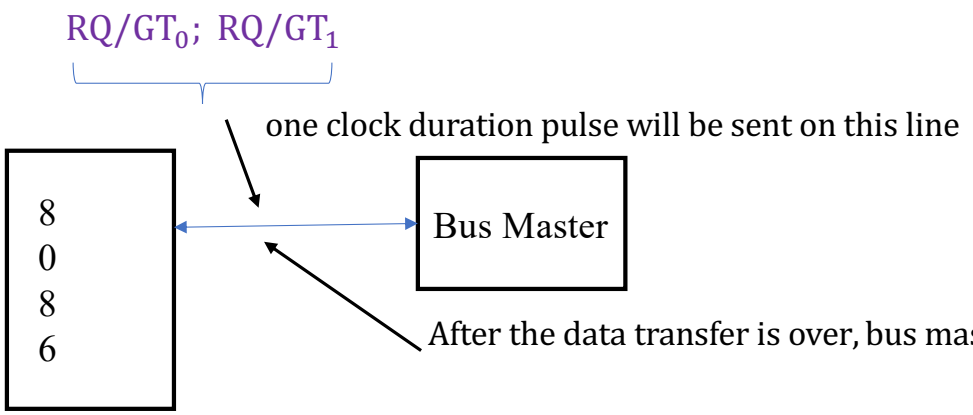
- The memory read control pin provides memory with a read control signal.

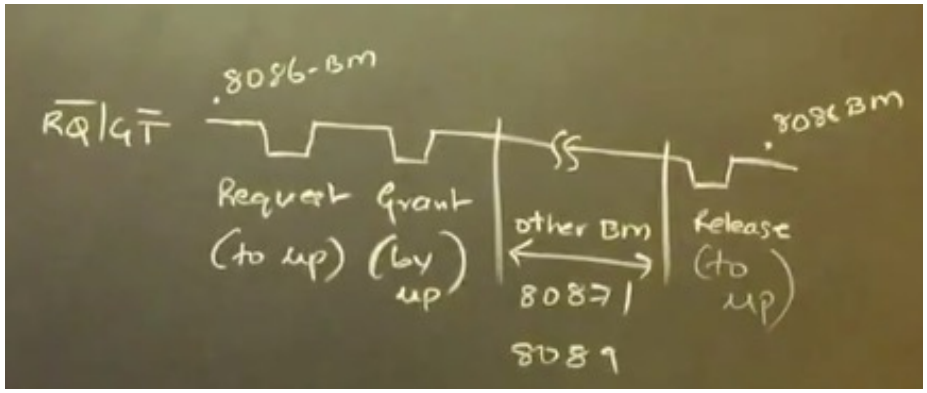
when a up wants to become Bus Master it sends RQ/GT, signal # when another up releases control over bus, it sends RQ/GT0 signal.

RQ/GT_0 and RQ/GT_1 → $Pin = 31\ and\ 30$, respectively

(RQ: Request; GT: Grant)

- Maximum mode – is a multi processor configuration,





- RQ/GT_0 is having higher priority than RQ/GT_1
- So, when simultaneous requests comes on both the signals, then RQ/GT_0 will be served first.
- When the 8086 completes its operation with the system bus, it informs through the same signal by sending 1 clock duration pulse that 8086 is going to relinquish the system bus to the Bus Master

$\overline{LOCK} \rightarrow Pin = 29$

- $\overline{LOCK}$: A low on $\overline{LOCK}$ indicates that the request from other co-processsors /Memory is prohibited

→ when low, prevents requests from other Bus Masters

---

32 common pins → MIN/MAX

minimum mode → 8086 – generate control signal

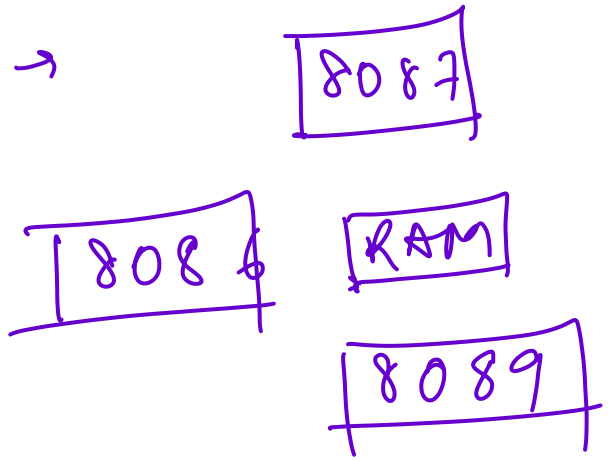
$\overline{RD}/\overline{WR}$, $M/\overline{IO}$

maximum mode →

MN/$\overline{MX}$ = 0

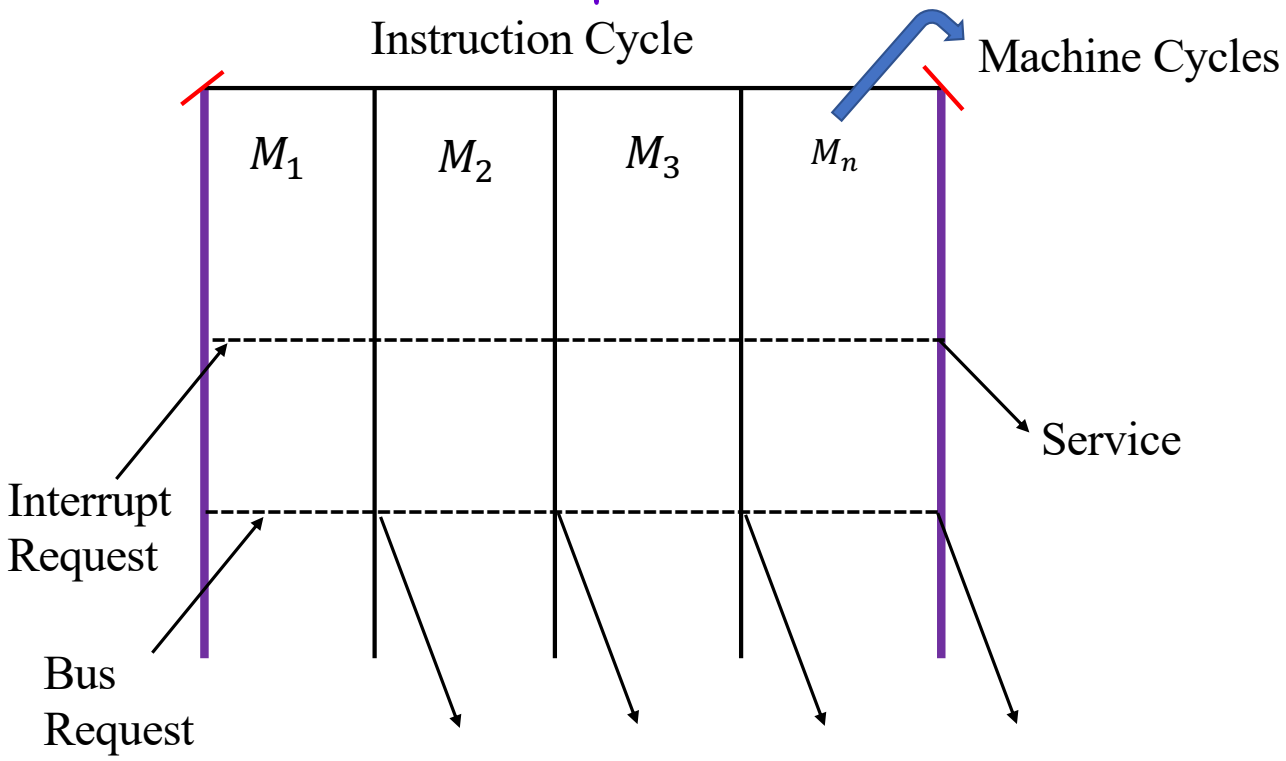
MAX = 0

MAX = 1



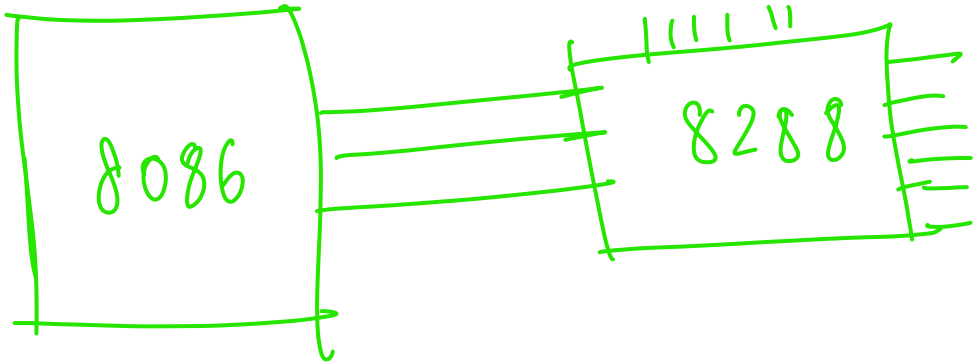
if all generate their own control signals then memory will be accessed by all at same time.

in max mode none of these three processors generate control signals. → We have a Bus controller (Common chip - 8288)

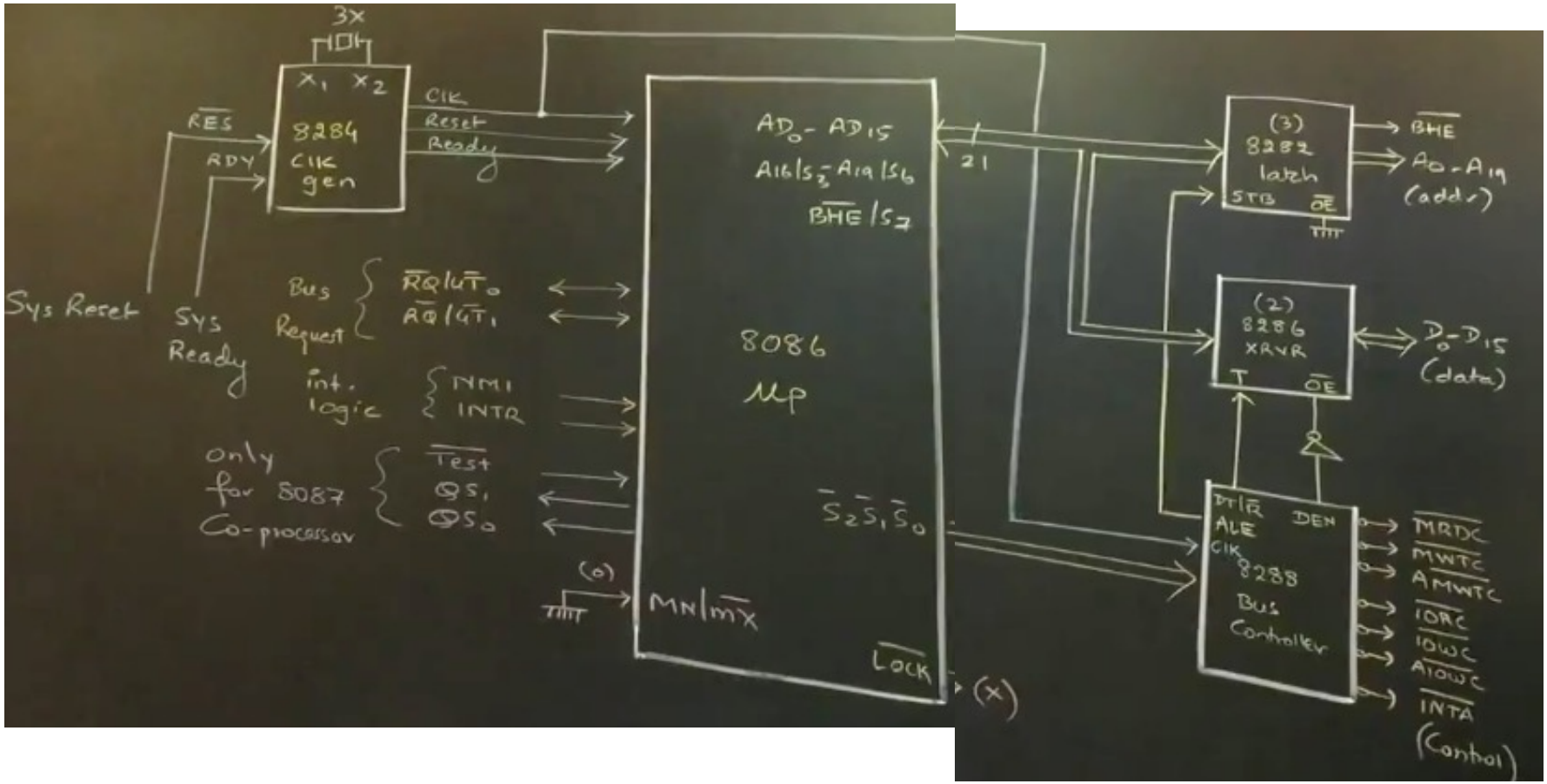


→ ALE is not there in maximum mode.

$\overline{S_0}$ $\overline{S_1}$ $\overline{S_2}$ used to give to 8288



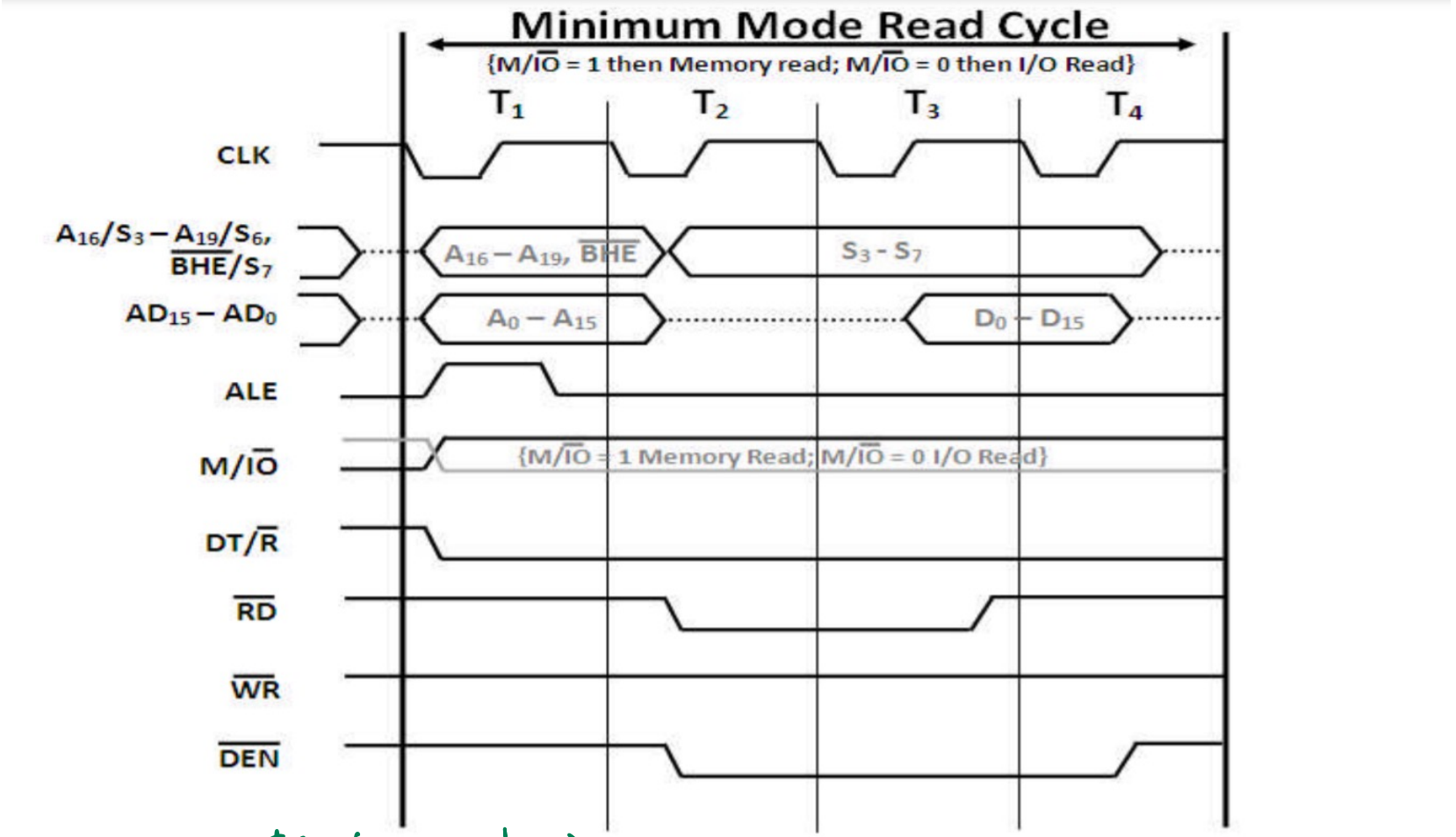
In maximum mode there was an extra command, advanced command, in which it is preparing slower devices for read/write operation in next cycle.



“Read” Cycle Timing Diagram for Minimum Mode

- By using three terminals $[\mathrm{M}/\overline{IO},\ \overline{RD}\ and\ \overline{WR}]$, we can execute four Machine Cycles.
  - . Memory Read $[\mathrm{M}/\overline{IO} = 1,\ \overline{RD} = 0\ and\ \overline{WR} = 1]$
  - . Memory Write $[\mathrm{M}/\overline{IO} = 1,\ \overline{RD} = 1\ and\ \overline{WR} = 0]$
  - . I/O Read $[\mathrm{M}/\overline{IO} = 0,\ \overline{RD} = 0\ and\ \overline{WR} = 1]$
  - . I/O Write $[\mathrm{M}/\overline{IO} = 0,\ \overline{RD} = 1\ and\ \overline{WR} = 0]$

‘Read’ Cycle timing Diagram for Minimum Mode



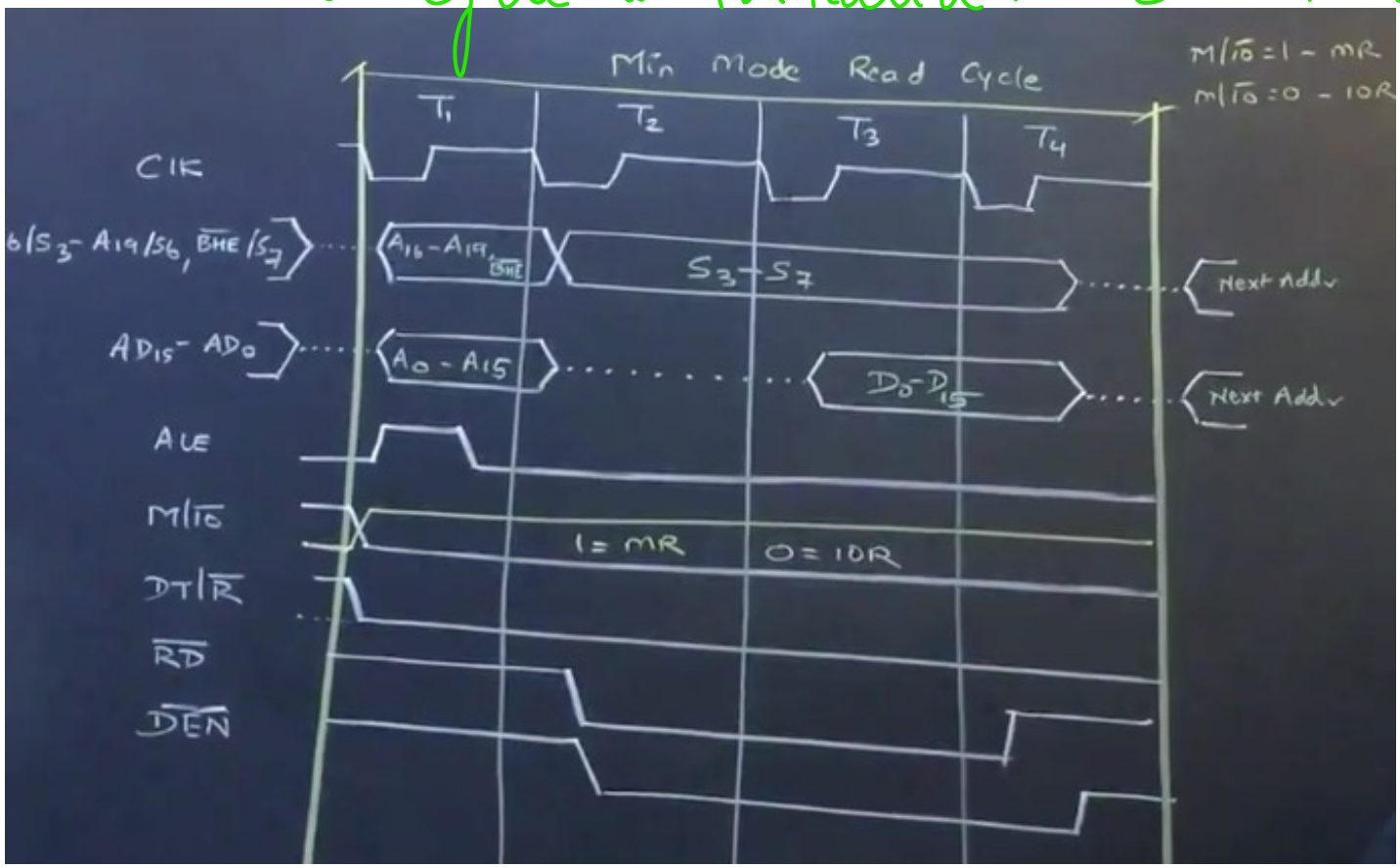
Processor wants to read →

→ places address on the bus

→ In max mode 8288 ka stroke pin is connected to ALE

pin through octal latch which makes ALE pin high.
(ALE is coming from external CRT).

→ How will 8288 know it has to make ALE high
→ when new cycle is initiated → $\overline{S_0}$ $\overline{S_1}$ $\overline{S_2}$ through status signals.



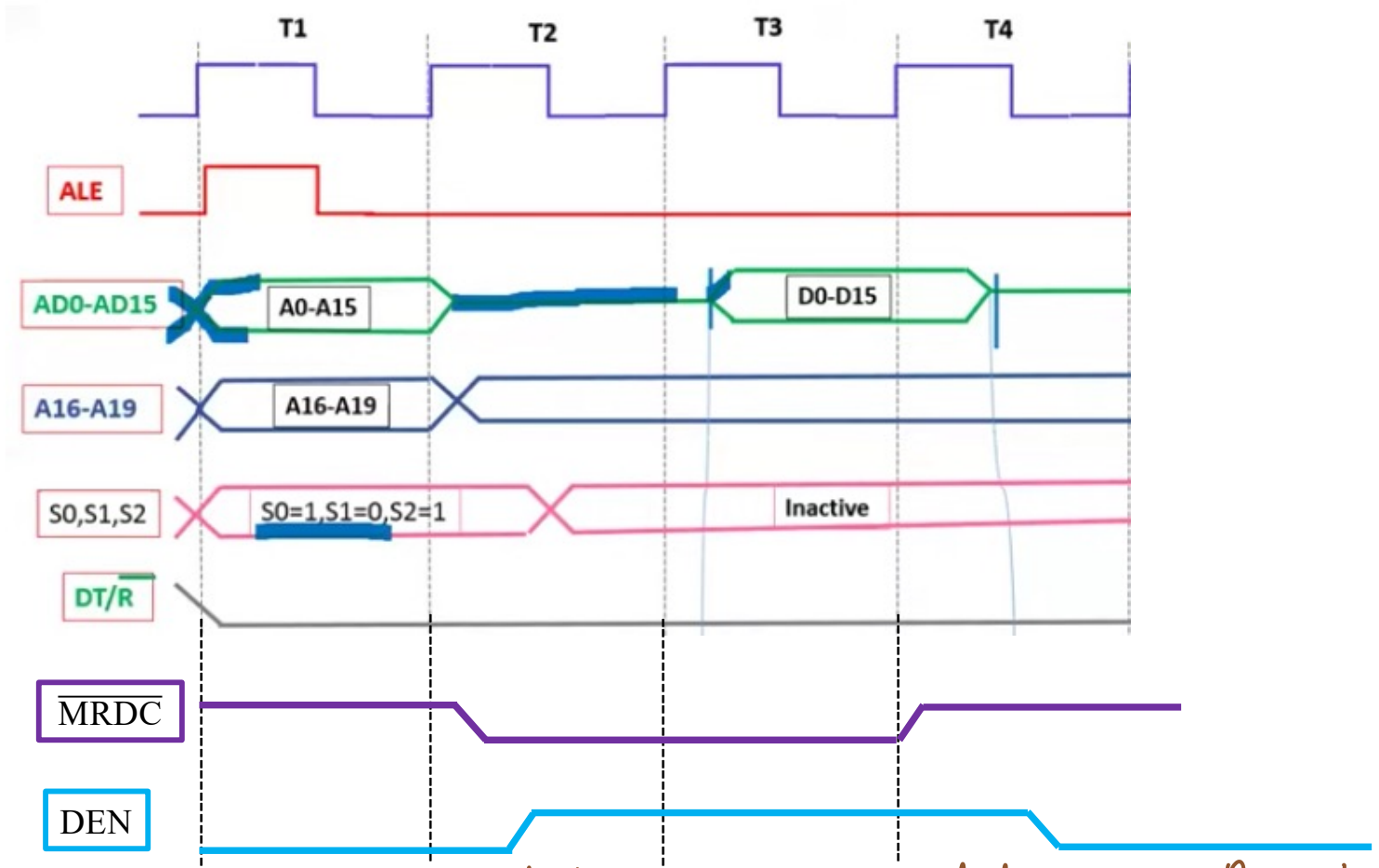
I st T state
↓
(0)
↓
memory read

As ALE is high address goes to latch it also tries to go through transceiver but it is in Off.

→ Now ALE has to go down. ↓

(8284 chip → clock → 5MHz) → same clock provided to bus controller → when 2nd clock goes through 8288, it knows that ALE has to go down.

"Read" Cycle Timing Diagram for Maximum Mode



inst. cycle → total time taken to complete an inst. fetch, decode execute inst. cycle can require several machine cycles.

Machine cycle → any activity done on system bus is c/a machine cycle. eg. Memory read

2 disturbances
Interrupt Bus control

BUS Timing

During T1:

- The address is placed on the Address/Data bus.
- Control signals M/IO, ALE and DT/R specify memory or I/O, latch the address onto the address bus and set the direction of data transfer on data bus.

During T2:

- 8086 issues the RD or WR signal, DEN, and, for a write, the data.
- DEN enables the memory or I/O device to receive the data for writes and the 8086 to receive the data for reads.

BUS Timing

During T3:

- This cycle is provided to allow memory to access data.
- READY is sampled at the end of T2.
  - If low, T3 becomes a wait state.
  - Otherwise, the data bus is sampled at the end of T3.

During T4:

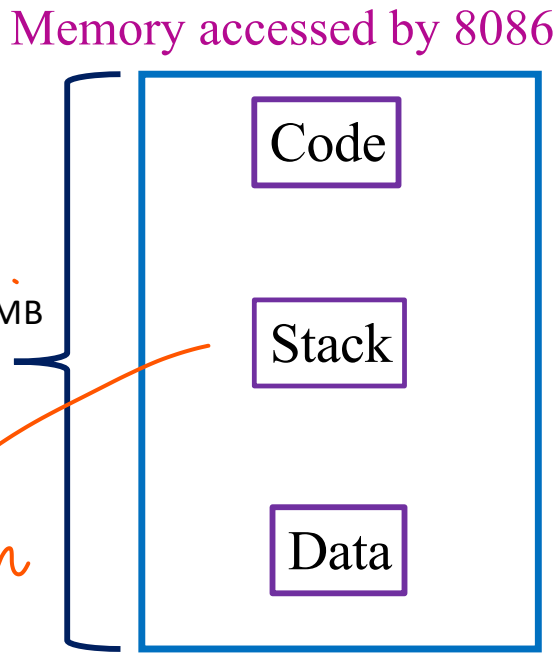
- All bus signals are deactivated, in preparation for next bus cycle.
- Data is sampled for reads, writes occur for writes.

memory → used to store program → data → randomly written in memory
↳ code → instruction (stored sequentially)

Memory Segmentation

- Memory segmentation is a solution to a problem.
- 8085 did not have a segmented memory. The problem started with large memories. Hence, the need of Segmentation.
- 8086 can access a total of 1MB of memory.
- Used: storing programs and data.
- Programs are called Code and Data are called Data.

→ For every data pushed on top, address decrements.
→ data is always written on top.
* Programmer initializes the segments and later up ensures there is no overwriting

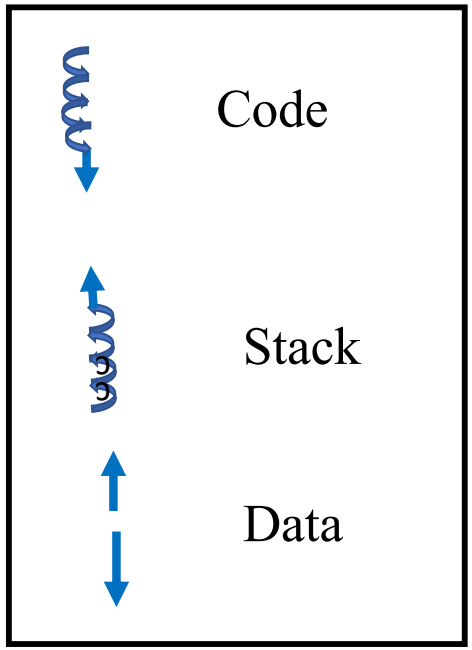


structured form of data

How do you store them?

- Programs (code):
  - Stored sequentially (not randomly).
  - After every instruction, the address gets incremented.
  - So, program (code) will go downwards.
- Data:
  - Data can be stored anyhow
  - Can be stored sequentially (or) randomly (unstructured data)
  - Data can go anywhere in any direction.
- Stack:
  - In memory, you also have Stack.
  - Stack also stores data. This is in structured form.
  - As we keep adding the data in Stack, address gets decremented. Stack grows upwards.

- Code grows downwards.
- Stack grows upwards.
- Data can go anywhere.
- Eventually, they all are going to overwrite on each other.
- But with 8086, as the memory increases, it becomes difficult to manage.

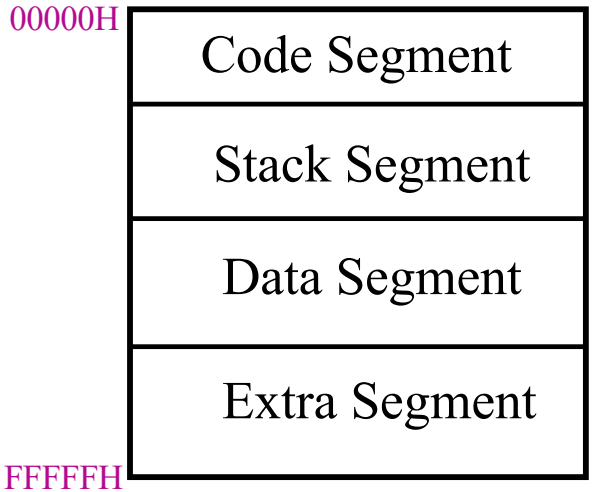


all this is going to overwrite in larger memory.
↓
to avoid segmentation.

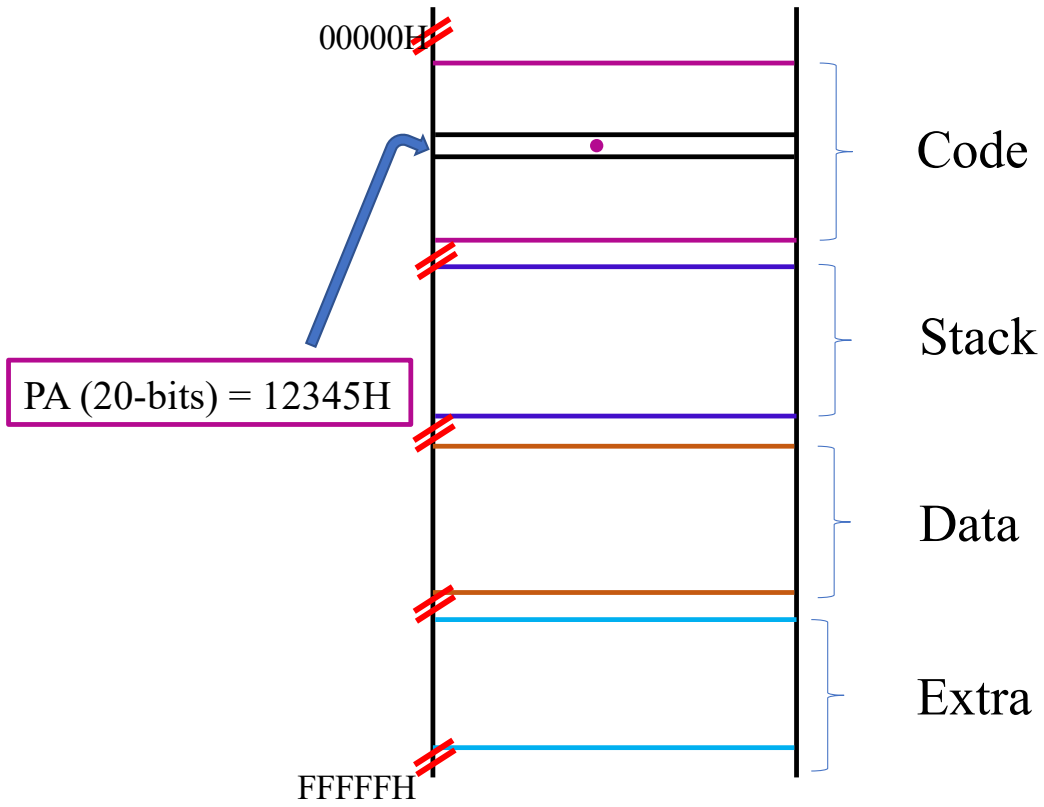
- In 8086, Memory is divided into 4 sections/segments called as *Code Segment*, *Stack Segment*, *Data Segment* and *Extra Segment.*
- Who creates these segments? The programmer.
- At the start of the program, programmer initializes the segments and later the Microprocessor makes sure there is no overwriting.

- Advantage: Prevents overwriting and organizes the memory.

Memory accessed by 8086



- This is the physical form of 8086 Memory – 1MB memory.
- Memory has four segments created by the programmer.



To understand Segmentation, let us consider an example.

- Consider the *Code Segment*. Look at *one location* in the Code Segment.
- How do you identify every memory location? Every *Memory location* must have its own *unique address*.
- This unique address of each Memory location is called as *Physical Address/Actual address (PA).*
- This Physical Address (PA) is of *20-bits*.

Ex: PA = 12345H (20-bits)

(Adv. - this PA is unique for each memory location and Disadv – it is of 20-bits - not computer compatible/byte compatible)

- $20 - bits$ are $2\,{}^{1}/_{2}$ bytes – not byte compatible.
- So, we never use 20-bits PA.

- 8085 had 16-bit address bus and hence 16-bit address which was byte compatible.
- Why we need 20-bit address bus? Since, we want to access large memory.
- 20-bit address bus (or) 16-bits address bus – 20-bit address bus – to access larger memory.
- 20-bit address (or) 16-bit address – 16-bit address - since byte compatible.
- We have 20-bit address bus, so bound to have 20-bit address and this is the physical address. But, no longer going to use.
- We are going to use *16-bit address* called as *Virtual Address*.
- We are creating a Virtual Address.
- Going to abolish Physical Address.

Virtual Address (VA) = Segment Address (16-bit) and Offset Address (16-bit)

Segment Address – starting address of each Segment.

Offset Address – Address of each location within a Segment.

64 kB max size of segment

16 Bytes min size of segment

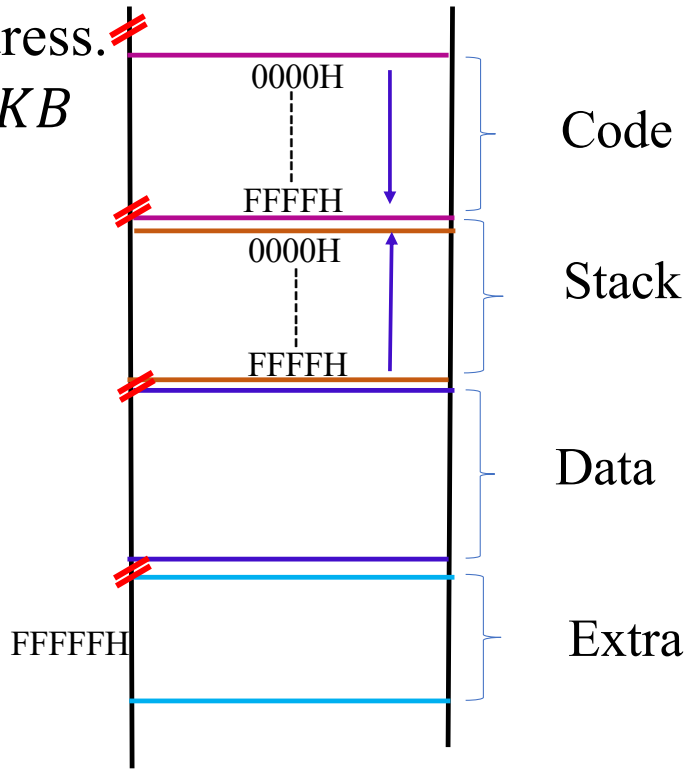
Physical Address (PA) = Segment Address x 10 + Offset Address... ... ... (1)

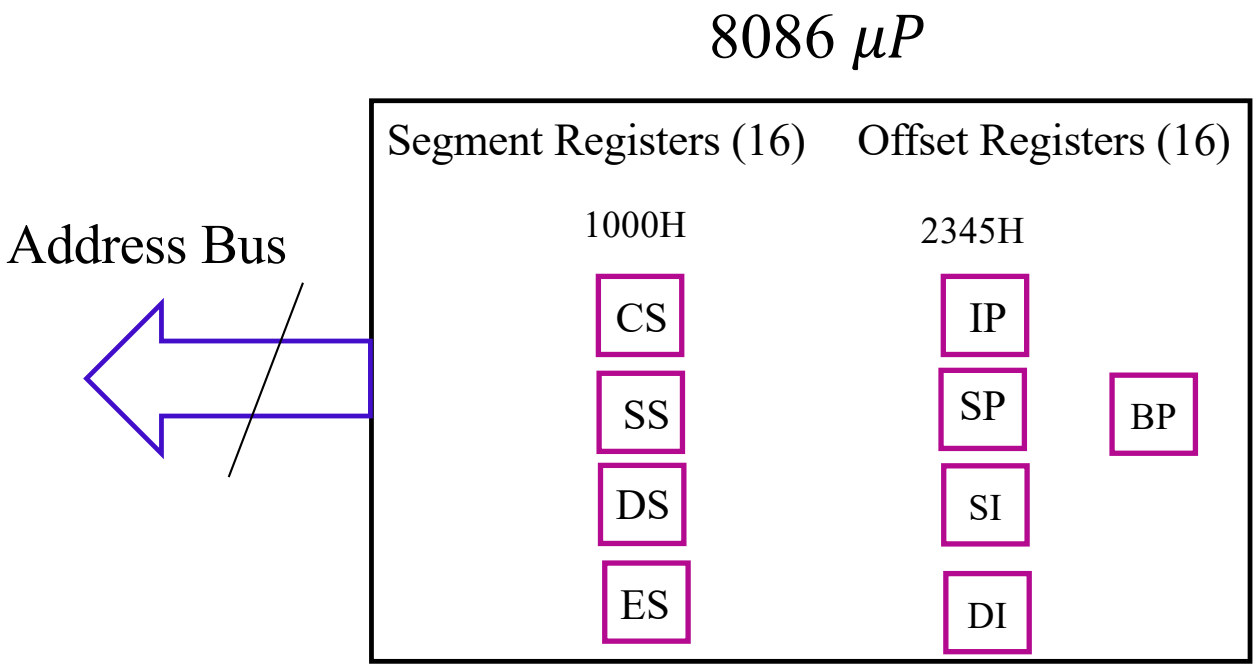
Each of the Segment Address and Offset Address are stored in Segment Registers and Offset Registers, respectively.

Offset Address: is the distance from the starting of the segment to the required location.

- There are three addresses, Physical Address, Segment Address and Offset Address.
- Which address you never give – Physical address.
- Which address you provide only once – Segment address, at the beginning of the program, to identify the program.
- Which address you give every time in the program – Offset address – it is provided always in the program.

- Code Segment – Starting Offset address is 0000H.
- If the programmer starts writing the instructions.
- Will the instructions be overwritten.
- No, as you will run out of Offset Addresses.
- Offset addresses are defined from 0000H – FFFFH.
- They are restricted. There is a limit in the Offset Address.
- Offset Address size = 16-bit = $2^{16} = 2^6$ x $2^{10} = 64\ KB$ (maximum size)





CS: contains address of Code Segment
SS: contains address of Stack Segment
IP: Instruction Pointer - gives Offset address
SP: Stack Pointer – points to top of stack.

Ex: CS = 1000H (16-bit)
IP = 2345H

- MP is given these two addresses and asked to fetch the instruction.
- To go to the memory, MP has to put the address on Address Bus.
- Address bus is of how many bits? 20-bits
- Which address MP places on the Address Bus? PA, SA (or) OA
  PA (or) VA.

- Physical address placed on Address Bus, Virtual Address is for convenience.
- Programmer has given Segment Address and Offset Address, MP evaluates the Physical Address using Eq. 1 and places on Address Bus.

Ex: 1. If, CS = 1000H, what is the physical address at the beginning of code segment .

$1000\ X\ 10 + 0000 = 10000$

The offset at the beginning is 0000H

So, the Code Segment actually begins from 10000.

2. If, SS = 3000H

$3000\ X\ 10 + 0000 = 30{,}000$

If the offset at the beginning is 0000H

So, the Stack Segment actually begins at 30,000.

3. If, DS = 5000H, Data Segment actually begins at 50,000

The ending (last) address $= 5FFFF$ (since, last offset address $= FFFFH$)

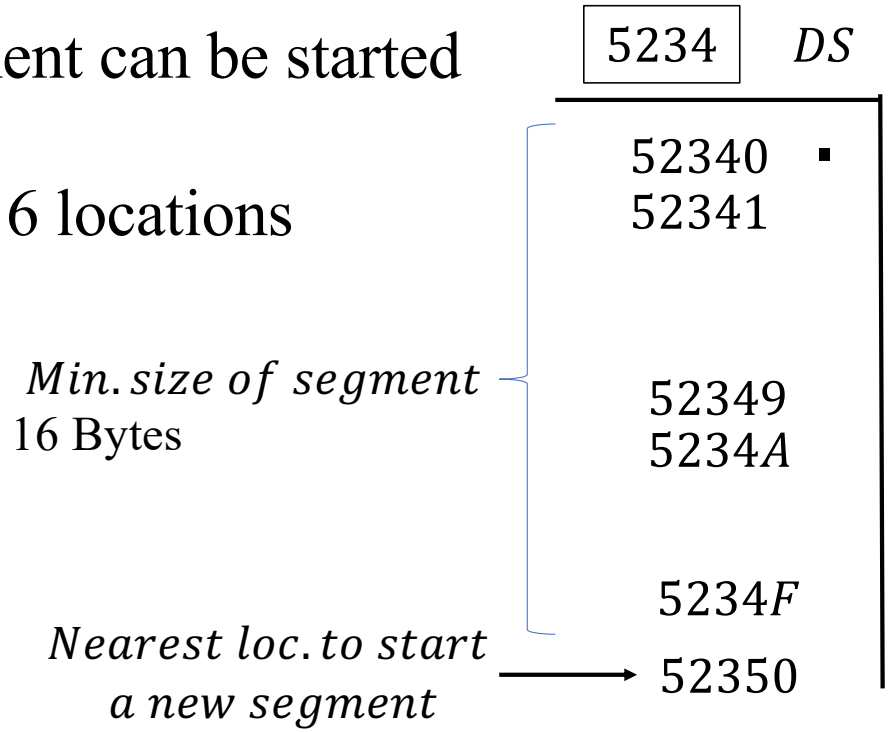
Ex: 1. If, DS = 5134H
So, the Data Segment actually begin at 51340.

2. If, DS = 5237H
So, the Data Segment actually begin at 52370.

As a programmer, want to start Data Segment at 52340, we will put 5234 in the Data register.
If want to start DS from 52345 – not possible.
A segment cannot begin at any location. It has to begin at a location which is multiple of 10, as we cannot put 5234.5 in DS register.

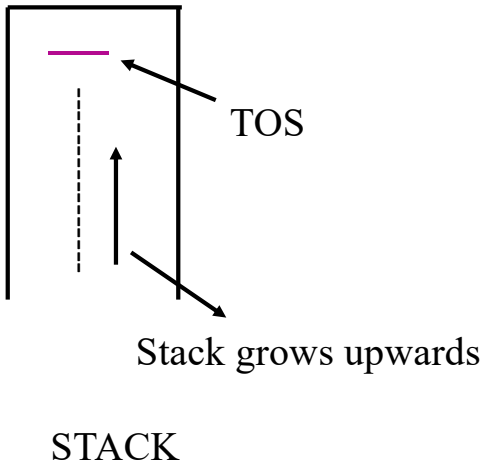
Minimum size of Segment:

- If want to start DS from 52340, will put 5234 in DS register.
- Want this to be the smallest segment possible, storing only 1 byte of data.
- The nearest location where next segment can be started is 52350.
- The minimum size of a segment has 16 locations and 16 Bytes size.



Stack Pointer (SP):

- Stack grows upwards.
- Top of stack (TOS) is pointed by SP.
- Push (or) Pop operation done from TOS.
- Address of TOS is given by SP.
- For Stack Segment, the register used to store the offset addresses is Stack Pointer (SP).
- After every Push operation, SP gets decremented.



Instruction Pointer (IP):

- For Code Segment, the register used to store the offset addresses is Instruction Pointer (IP).
- After every instruction, IP will get incremented.

Source Index (SI):

- For Data Segment, the register that stores the offset address is Source Index (SI).
- After every data operation, SI will either get decremented (or) incremented.

Destination Index (SI):

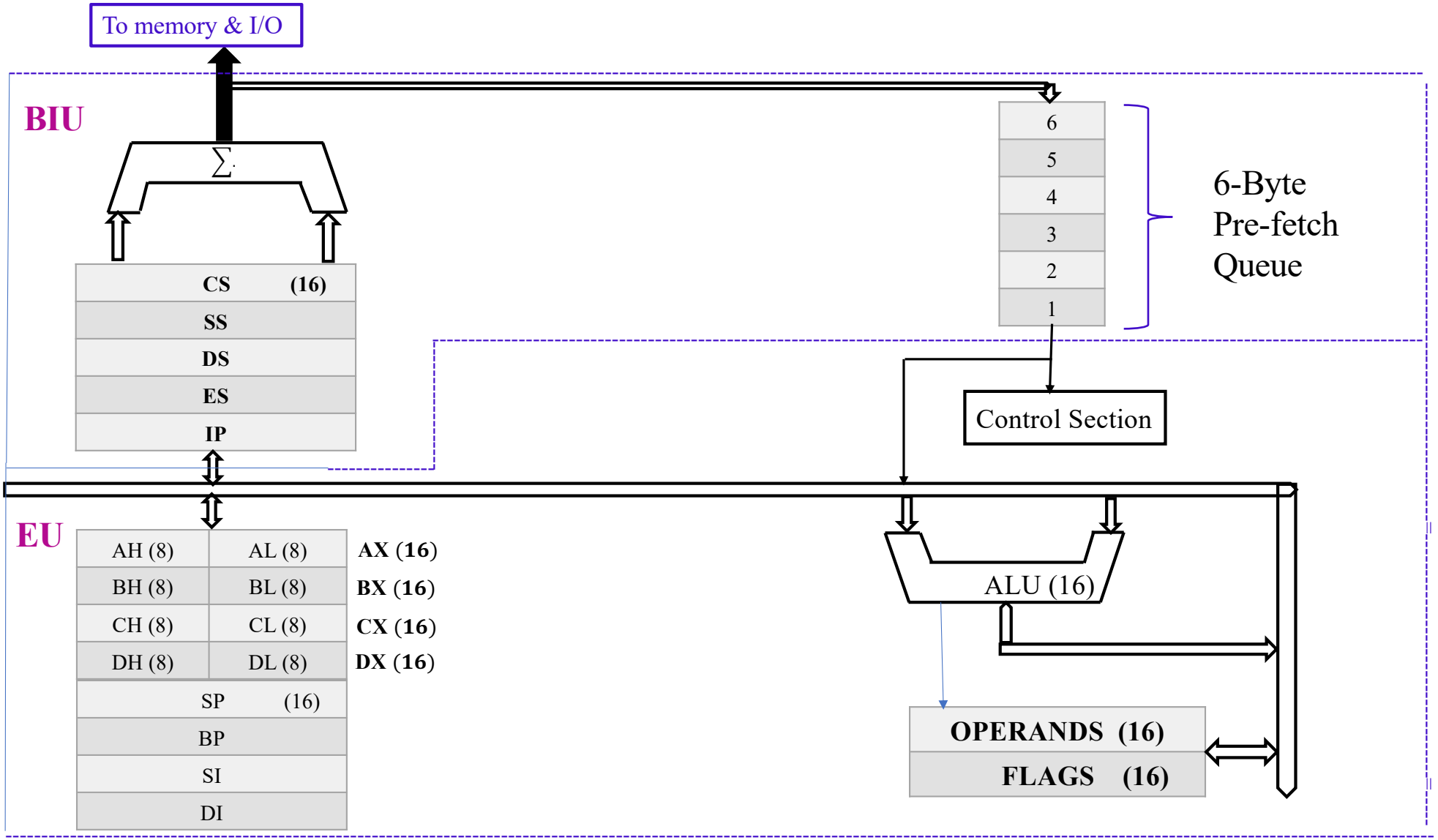
- For Extra Segment, the register that stores the offset address is Destination Index (DI).

Base Pointer (BP):

- Every segment has one Segment register and one offset register.
- Only Stack Segment has 2 Offset registers: SP and BP
- Base Pointer (BP): points anywhere in the Stack
- Stack Pointer (SP): used for Push and Pop operations. Still points at TOS.
- Base Pointer (BP): used for random access of the Stack. Can read any data/ modify any data from the stack.

Architecture of 8086

- 8086 – is a 16-bit processor – can transfer 16-bit data at a time – operates on 16-bit data at a time – 16-bit ALU.
- Architecture divided into 2 units – *BIU (Bus Interface Unit)* and *EU (Execution Unit).*
- Why 2 units? Pipelining
- 8085 – fetching and then execution – no pipelining.
- 8086 – BIU fetches 1st instruction – sends to EU – while, EU executes 1st instruction – BIU fetches 2nd instruction.
- Architecture performing two operation at a time – hence, two units.


To memory & I/O
BIU
∑
CS (16)
SS
DS
ES
IP
6
5
4
3
2
1
6-Byte
Pre-fetch
Queue
Control Section
EU
AH (8)
AL (8)
AX (16)
BH (8)
BL (8)
BX (16)
CH (8)
CL (8)
CX (16)
DH (8)
DL (8)
DX (16)
SP (16)
BP
SI
DI
ALU (16)
OPERANDS (16)
FLAGS (16)

Arithmetic & Logic Unit (ALU): used for performing arithmetic and logical operation

$$ADD\ BL, CL$$

$\sum Unit =$ used to calculate Physical address.

- While, ALU is performing addition, $\sum Unit$ calculating Physical address for fetching the next instruction.

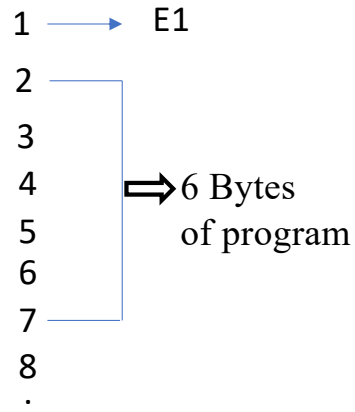
Bus interface Unit (BIU): 1. fetches next instruction; 2. calculate Physical address and 3. manages an object called as *Queue*.

- CS, SS, DS, ES are the registers that contain the addresses of the segments.
- IP, SP, SI and DI – registers used to store offset addresses within a Segment.
- Both these addresses used to get Physical address and given to Memory – from Memory instruction is fetched.

- Instruction enters MP through Data Bus.
- Execute this instruction immediately? No – Some Inst. Executing right one. BIU has fetched next instruction in advance (or) *"prefetched".*
- The Instructions fetched in advance (or) prefetched – stored in an object called *"6- byte pre-fetch Queue".*
- Size of Queue – *6 Bytes*

Definition of Pipelining – when an instruction executing, MP fetches the next instruction.

- Currently executing 1st instruction.
- While, 1st instruction executing, BIU not only fetches next instruction, but it fetches next 6 bytes of program and stores in Queue.



- All instructions are not of same size. Instructions can be of 1byte, 2B, 3B, 4 bytes……6 byte. The biggest instruction in 8086 is of 6 bytes.

MOV BL, CL

MOV BL, 25H

MOV BX, 2000H

} Assembly language

- Programmer writes programs in AL and give to an assembler. Assembler converts it to Machine language and puts the Program into Memory and gives to Processor.
- Every instruction when it converts in ML, it becomes a series of 1's and 0's.
- Suppose, ADD BL, CL in ML becomes 0100001-----11. This binary pattern indicating an operation that we want to perform is called as *OPCODE.*
- Every instruction has its own unique Opcode.

MOV BL, CL – there is an opcode for this

MOV BL, 25H – there is an opcode for MOV BL

- Numbers do not have Opcode.

MOV BL, CL – there is only opcode in this. This is the smallest instruction of size 1Byte.

MOV BL, 25H – there is opcode and operand (25H) – 2 Byte size

MOV BX, 2000H – there is opcode and operand (2000H) – 3 Bytes size.

- So, while an instruction is executing, next 6 bytes (not next 6 instructions) are stored in 6-Byte Pre-fetch Queue.
- When EU finishes executing an instruction, it takes next instruction from the Queue and not the Memory.
- When does BIU fetches instructions for Queue – when 2 bytes of Queue are empty – BIU fetches 2 bytes from memory and stores in Queue – as 8086 supports 16-bit data bus – in a clock cycle 16-bit data can be transferred.

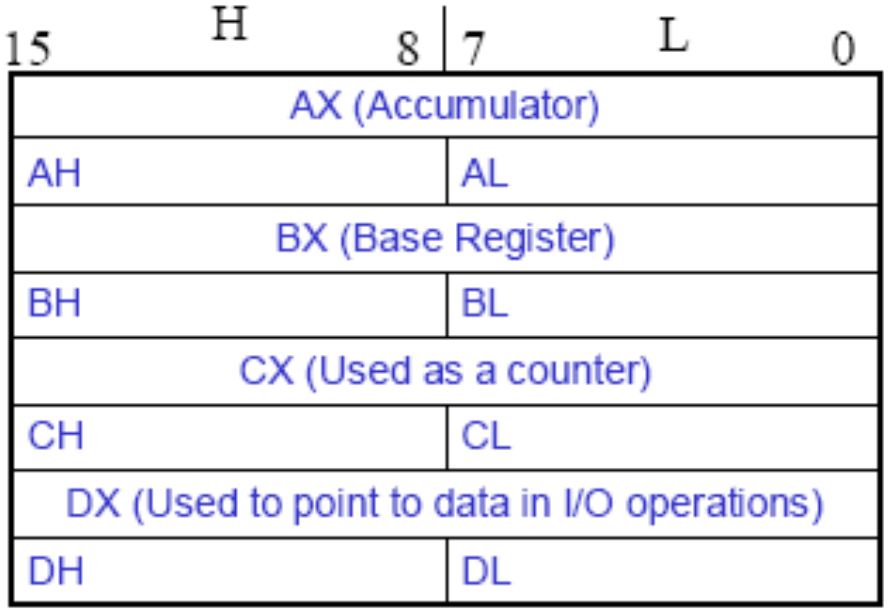
Execution Unit: Executes the instructions present in the Queue.

- Instruction from Queue pass to Control Section, where they get decoded.
- Ex: ADD BL, CL (in assembly language) – decoding of Opcode done in Control section – what the operations is?
- Once decoding is done, Control Section generates control signals – these control signals given to different entity in architecture for performing operations – and then execution takes place.
- SP, BP, SI, DI and IP – offset registers – hold the offset address within a segment in the Memory.
- AH, AL, BH, Bl, CH, CL, DH, DL – General Purpose Registers.
- Use them as 8 registers - AH, AL, BH, BL, CH, CL, DH, DL — each of 8-bits.
- Use them as 4 registers - AX, BX, CX, DX — each of 16-bits.

Ex: $\overset{12}{AH}$ $\overset{34}{AL} \rightarrow AX = \underbrace{12}_{\text{Higher byte}}\ \underbrace{34}_{\text{Lower byte}}$

Higher byte Higher byte

General Purpose Registers



AX - the Accumulator
BX - the Base Register
CX - the Count Register
DX - the Data Register

- Normally used for storing temporary results
- Each of the registers is 16 bits wide **(AX, BX, CX, DX)**
- Can be accessed as either 16 or 8 bits AX, AH, AL

General Purpose Registers

CX

- Count register
- Used as a loop counter
- Used in shift and rotate operations

DX

- Data register
- Used in multiplication and division
- Also used in I/O operations

Pointer and Index Registers

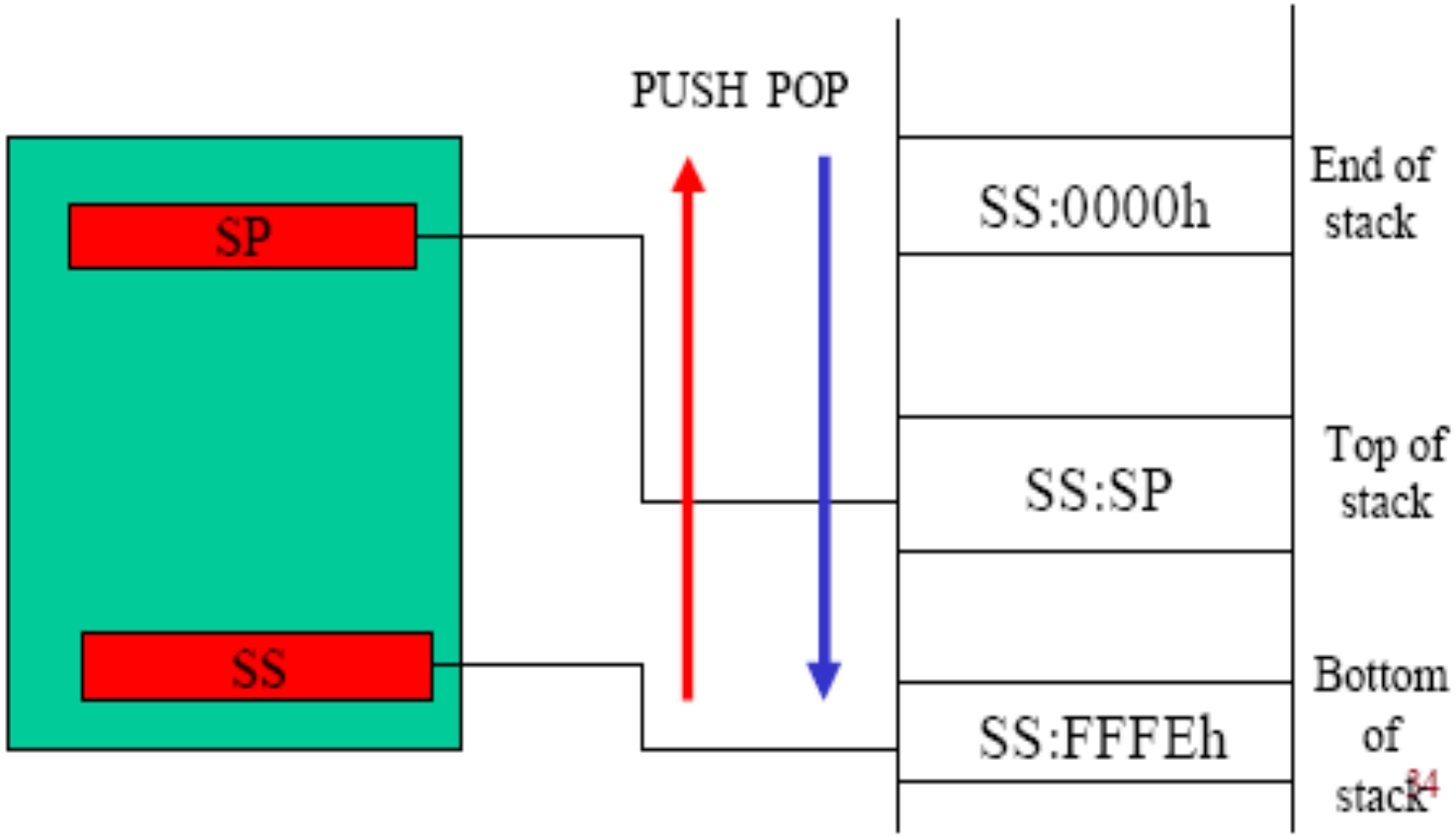
SP	Stack Pointer
BP	Base Pointer
SI	Source Index
DI	Destination Index
IP	Instruction Pointer

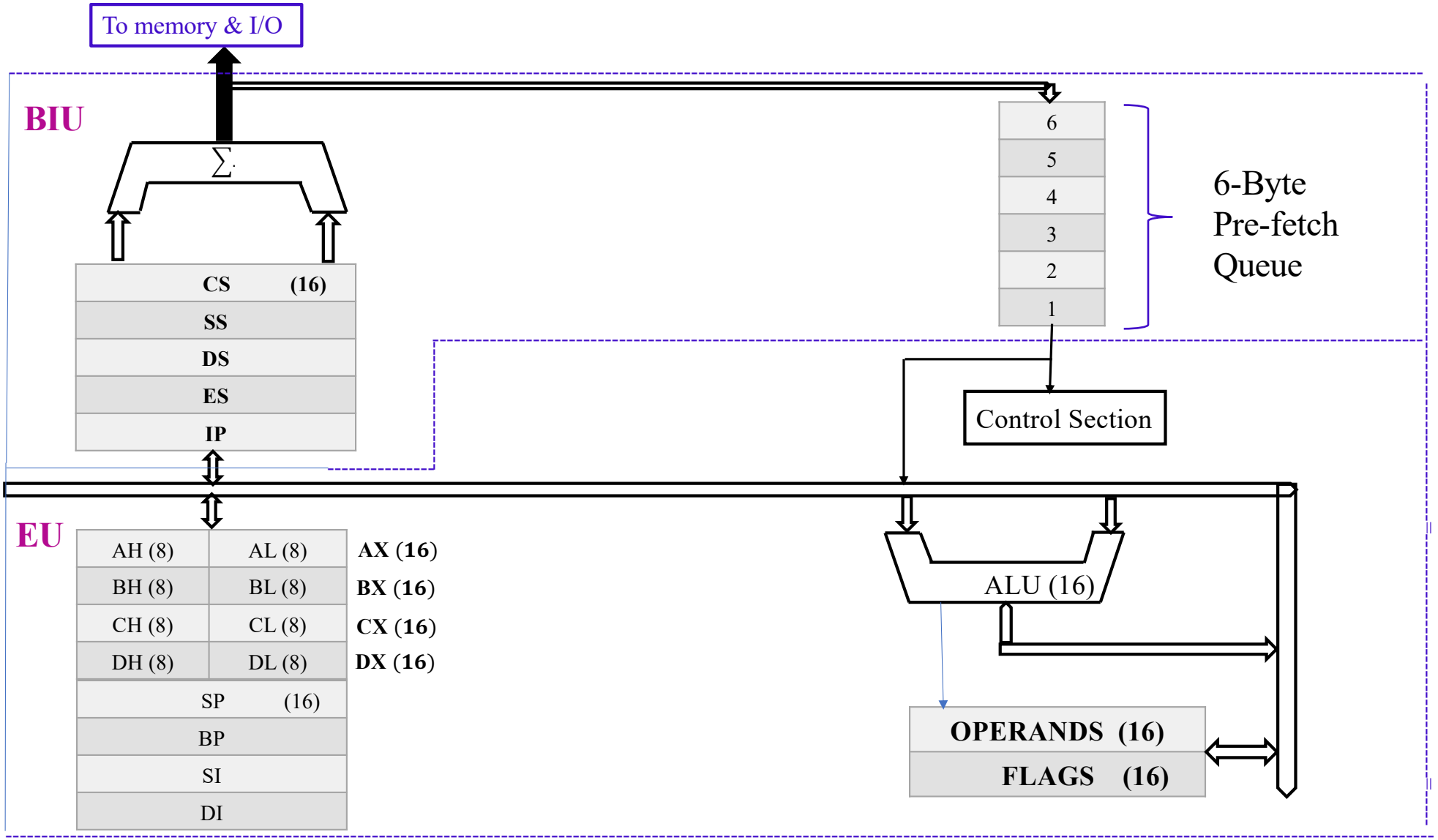
- All 16 bits wide, L/H bytes are not accessible
- Used as memory pointers
  - Example: MOV AH, [SI]
    - *Move the byte stored in memory location whose address is contained in register SI to register AH*
- IP is not under direct control of the programmer

Stack

- The stack is used for temporary storage of information such as data or addresses.
- When a CALL is executed, the 8086 automatically PUSHes the current value of CS and IP onto the stack.
- Other registers can also be pushed
- Before return from the Subroutine, POP instructions can be used to pop values back from the stack into the corresponding registers.

Stack


PUSH POP
SP
SS
SS:0000h
SS:SP
SS:FFFEh
End of stack
Top of stack
Bottom of stack


To memory & I/O
BIU
∑
CS (16)
SS
DS
ES
IP
6
5
4
3
2
1
6-Byte
Pre-fetch
Queue
Control Section
EU
AH (8)
AL (8)
AX (16)
BH (8)
BL (8)
BX (16)
CH (8)
CL (8)
CX (16)
DH (8)
DL (8)
DX (16)
SP (16)
BP
SI
DI
ALU (16)
OPERANDS (16)
FLAGS (16)

Program to Add:

$04H + 05H = 09H$

BL CL BL

MOV BL, 04H

MOV CL, 05H

ADD BL, CL; BL→ $BL + CL$

AH	AL
BH	BL **(04)**
CH	CL **(05)**
DH	DL

Working of ADD BL, CL:

1. Instruction stored in memory.

2. To fetch instruction from Memory, registers CS and IP come into picture. CS gives segment address and IP gives offset address – using these two address – MP calculates Physical address using $\sum Unit$.

3. MP puts this PA on address bus to fetch instruction from memory - through data bus instruction is stored in Queue.

4. Control section receives Opcode of ADD – decoded in Control section.

5. After decoding, Control Section generates control signals for different entities.

Program to Add using one register:

$04H + 05H = 09H$

BL BL

MOV BL, 04H

ADD BL, 05; BL→ $BL + 05H$

- This instruction has opcode and operand. Opcode given to Control Section.
- Operation is decoded and not the operand

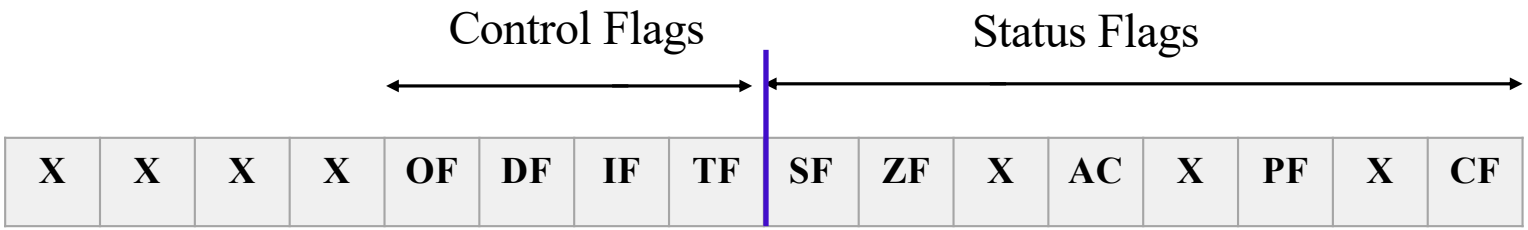
OPERANDS (16-bit register)

- Temporary register.
- Not relevant to programmer. Not used in programming.
- Used by Processor for storing temporary values..

Ex: XCHG BX, CX

Flag Register of 8086

- Flag Register – has various flags.
- Each flag gives some status about the current result.



- Flag Register is a 16-bit register.

Carry Flag: tells whether there is a carry out of the MSB (or) not.

Ex:

$$\begin{array}{r} 1\ 1\ 1\ 1\ \ 1\ 1\ 1\ 1 \\ +\ 0\ 0\ 0\ 0\ \ 0\ 0\ 0\ 1 \\ \hline 0\ 0\ 0\ 0\ \ 0\ 0\ 0\ 0 \end{array}$$

$\boxed{1}$ ⟶ Carry out of MSB

If there is Carry $= 1$ coming out of MSB $\rightarrow CF = 1$ (Carry Flag set to 1)

Parity Flag: tells the Parity of the result.

- Parity is calculated by the number of $1's$ in the result.
- If result has EVEN Parity, P$F = 1$
- If result has ODD Parity, P$F = 0$

Ex: If result of an operation is $\underbrace{1\ 0\ 0\ 0\ \ 0\ 0\ 1\ 1}$

Number of $1's = 3$; ODD Parity $\rightarrow$ PF $= 0$

If result of an operation is $\underbrace{0\ 0\ 0\ 0\ \ 0\ 0\ 1\ 1}$

Number of $1's = 2$; Even Parity $\rightarrow$ PF $= 1$

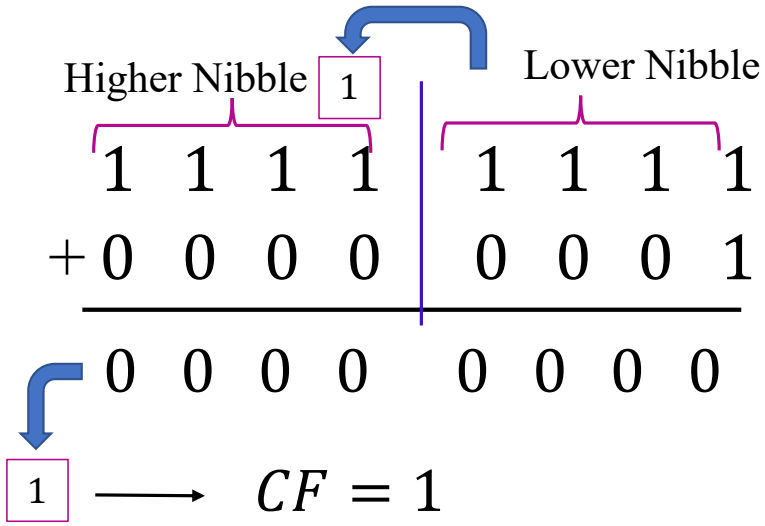
Auxiliary Carry: is a Carry from *Lower Nibble* to *Higher Nibble*.

1 Nibble $= 4\ bits$

AC $= 1 \rightarrow$ Carry from lower nibble to higher nibble

AC $= 0 \rightarrow$ No Carry from lower nibble to higher nibble

Ex:



Zero Flag: tells if result is *Zero* (or) not.

- If result is 0; $ZF = 1$
- If result is 1; $ZF = 0$

Signed Flag (SF): tells whether result is positive (or) negative.

If MSB of result $= 1 \rightarrow$ Negative Number

If MSB of result $= 0 \rightarrow$ Positive Number

Whatever is the value of MSB, it is copied into SF.

If SF $= 1 \rightarrow$ MSB of result $= 1$ (therefore, we conclude number is negative)

If SF $= 0 \rightarrow$ MSB of result $= 0$ (therefore, we conclude number is positive)

Unsigned/Signed Numbers:

- An 8-bit number. Can be treated as unsigned (or) signed number.
- Unsigned numbers do not have sign. They are always assumed to be positive (ex. Your roll number, score od a match).

- In such numbers, all the 8-bits give *Magnitude*.
- No bit is wasted in giving the Sign.
- Irrespective of whether MSB is 0 (or) 1, the number is positive.
- So, the concept which we saw just now, that MSB tells whether no. positive (or) negative does not apply to unsigned numbers.

$2^8 = 256$ positive numbers

$0 \quad - \quad 255$ $\qquad$ 0 0 0 0 0 0 0 0 $\qquad$ → smallest number

$00H \quad - \quad FFH$ $\qquad$ 1 1 1 1 1 1 1 1 $\qquad$ → largest number

Signed Numbers:

$|MSB|$ ⟵ 7-bits. ⟶ $\qquad$ 8-bit number → $2^8 = 256$ – all not positive nor negative

Sign bit $\qquad$ Magnitude

$2^7 = 128$ combinations with S $= 1 \rightarrow$ negative numbers

$2^7 = 128$ combinations with S $= 0 \rightarrow$ positive numbers

$-128 \ldots\ldots\ldots\ldots -1 \quad 0 \ldots\ldots\ldots\ldots\ldots..127$

$\underbrace{-80 \ldots\ldots\ldots\ldots -01}_{\text{Range of negative numbers}} \quad \underbrace{00 \ldots\ldots\ldots\ldots\ldots 7FH}_{\text{Range of positive numbers}}$

For Sign Flag, we deal with Signed representation.

Ex: (10000 0011) →. Result

If Unsigned → $83H$

If Signed → MSB= 1 → negative number → final result in $2's$ complement → $-7DH$

- Sign Flag doesn't always give the correct Sign. Sometimes it gives you the opposite, the wrong Sign – when there is an *Overflow*.
- Overflow Flag – indicates if there is an Overflow in the result - $\mathrm{OF} = 0$
- Means – the result has gone out of range for an Signed Number - a positive number become more than $7FH$ (or) a negative become less than -80.

Ex:

$$\begin{array}{rcl} 0\ 1\ 1\ 1\ \ 1\ 1\ 1\ 1 & \rightarrow & \text{positive number (7FH)} \\ +\qquad\qquad\qquad\ 1 & & \\ \hline 1\ 0\ 0\ 0\ \ 0\ 0\ 0\ 0 & \rightarrow & \text{positive number (80H)} \end{array}$$

MSB $= 1 \rightarrow \mathrm{SF} = 1$ (number is negative, but added two positive numbers) → false result → as result has gone out of range $00 \ldots \ldots \ldots \ldots \ldots 7FH$.

In this case, $\mathrm{OF} = 1$

- Such an event is called as an Overflow and $\mathrm{OF} = 1$, indicating that Signed Flag $(\mathrm{SF} = 1)$ is wrong.
- When a positive number flows out of range i.e. goes beyond $7FH$ and a negative number flows out of range i.e. goes less than $-80H$, this event is called as Overflow making $\mathrm{OF} = 1$ – indicating that Signed Flag is wrong.
- Morale of the story. Putting it all in conclusion, never directly check the SF. If want to know the Sign of a number, first check the OF. If OF=0,

Ex: 1

		OF	SF	ZF	AC	PF	C
$42H$	0 1 0 0 0 0 1 0						
$+23H$	0 0 1 0 0 0 1 1	↓	↓	↓	↓	↓	↓
$65H$	0 1 1 0 0 1 0 1	0	0	0	0	1	0

Ex: 2

		OF	SF	ZF	AC	PF	C
37H	0 0 1 1 0 1 1 1						
+29H	0 0 1 0 1 0 0 1	↓	↓	↓	↓	↓	↓
60H	0 1 1 0 0 0 0 0	0	0	0	1	1	0

Ex: 3

		OF	SF	ZF	AC	PF	C
42H	0 1 0 0 0 0 1 0						
+43H	0 1 0 0 0 0 1 1	↓	↓	↓	↓	↓	↓
85H	1 0 0 0 0 1 0 1	1	1	0	0	0	0

- Carry Flag, Overflow Flag, Parity Flag, Auxiliary Carry and Signed Flag are the ***Status Flags*** – indicate the status of the current result.
- These are changed by the ALU after every arithmetic & logical operation.
- Rest three flags called as – **Control Flags**. These ae used by the programmer – programmer control these 3 flags.

Trapped Flag (TF): indicates whether want to perform single stepping.

Single Stepping $\rightarrow TF = 1$

- This process of executing the program line- by-line and checking the result is called as *Single Stepping*.

Single Stepping $\rightarrow TF = 1$

No Single Stepping $\rightarrow TF = 0$

Programmer controlling the process of Single Stepping.

Interrupt Flag (IF):

- When there is an interrupt from external device, microprocessor decides whether to service it (or) not – whether to enable the interrupt(or) not.
- This is done through Interrupt Flag (IF).

$F = 1 \rightarrow$ Interrupt enabled

I$F = 0 \rightarrow$ Interrupt disable

Direction Flag (DF): decides the direction of the String instruction.

- String instruction operates on "Strings" – String is a set of data.
- Programmer can decide, whether to auto-decrement (or) auto-increment the address.
- Programmer decided the direction of data transfer (string transfer).

Auto-increment $\rightarrow DF = 0$

Auto-decrement $\rightarrow DF = 1$