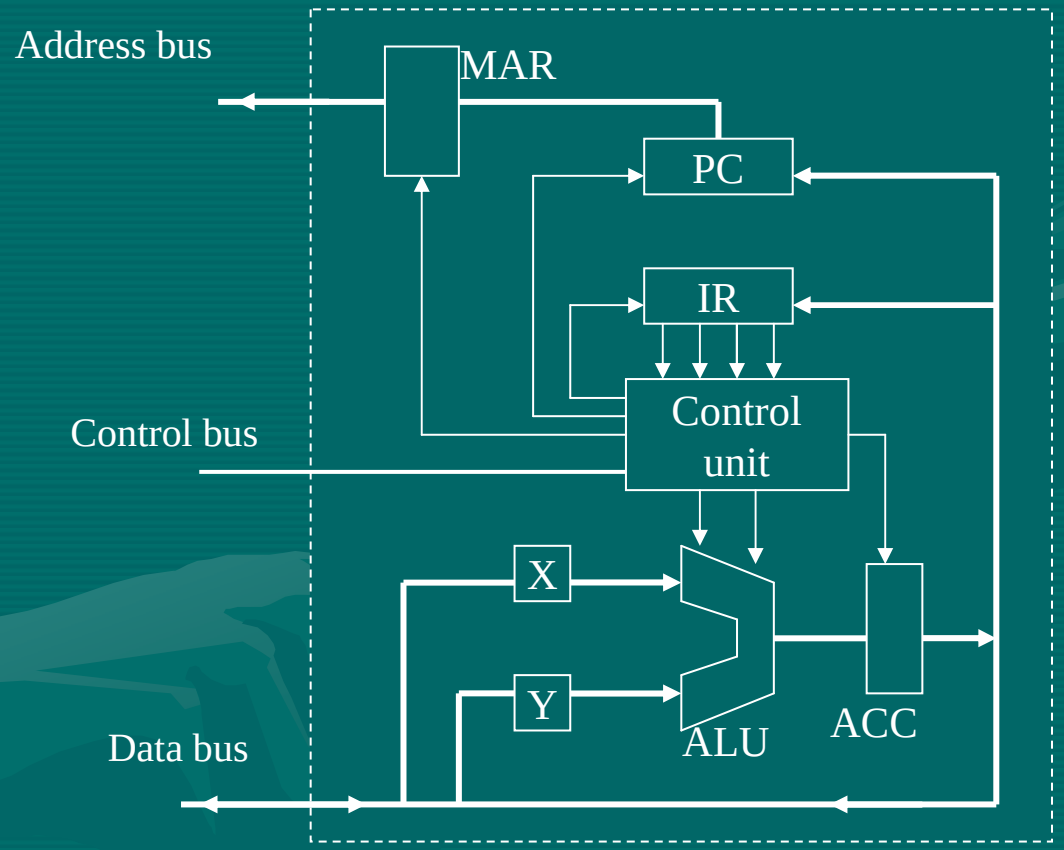
# Intel 8086 Microprocessor

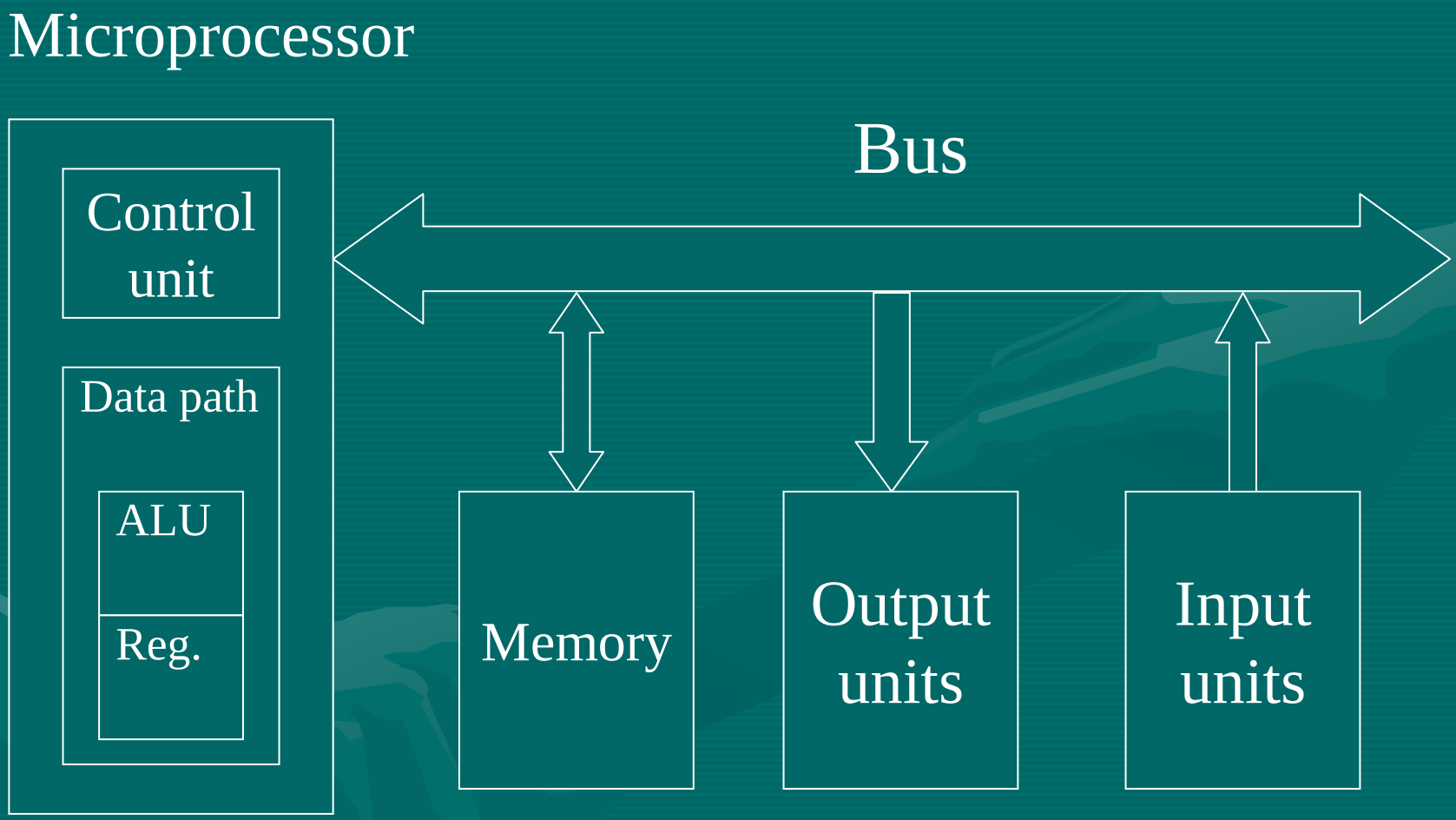
# What are microprocessors?

- A microprocessor is a processor (or Central Processing Unit, CPU) fabricated on a single integrated circuit.

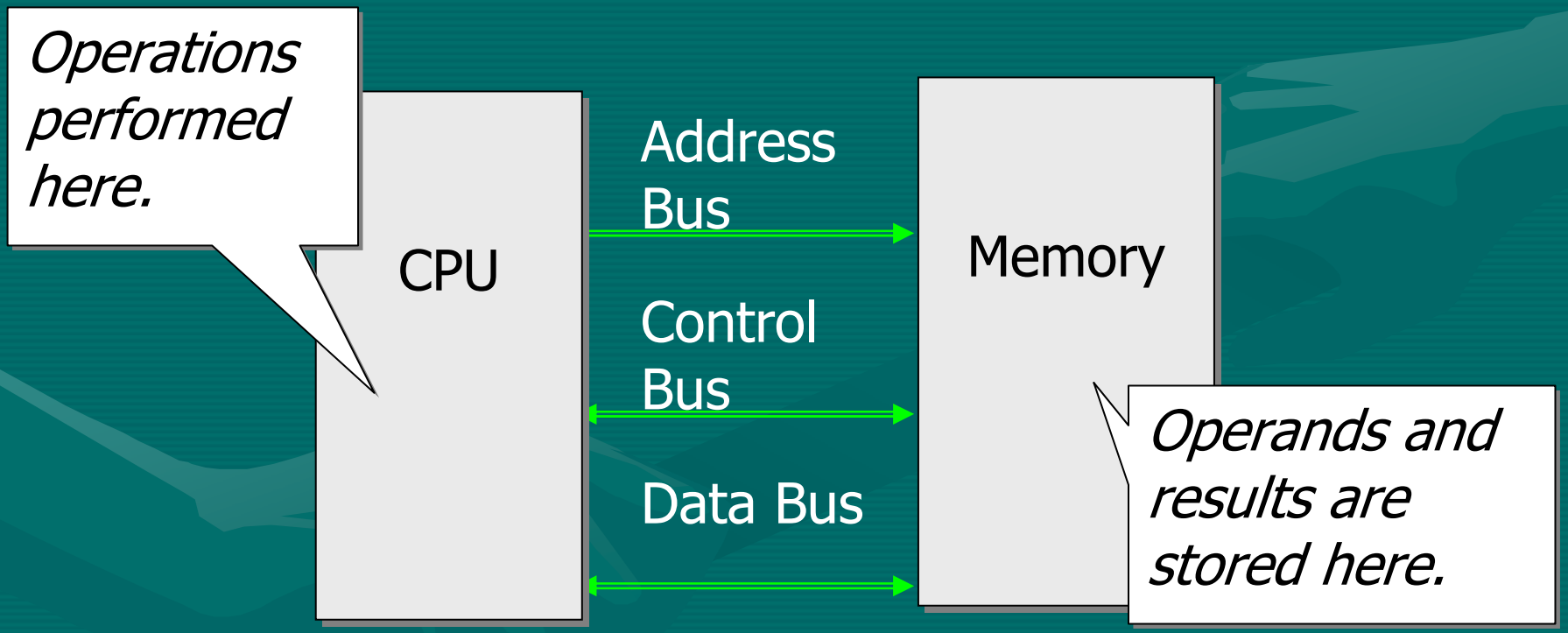


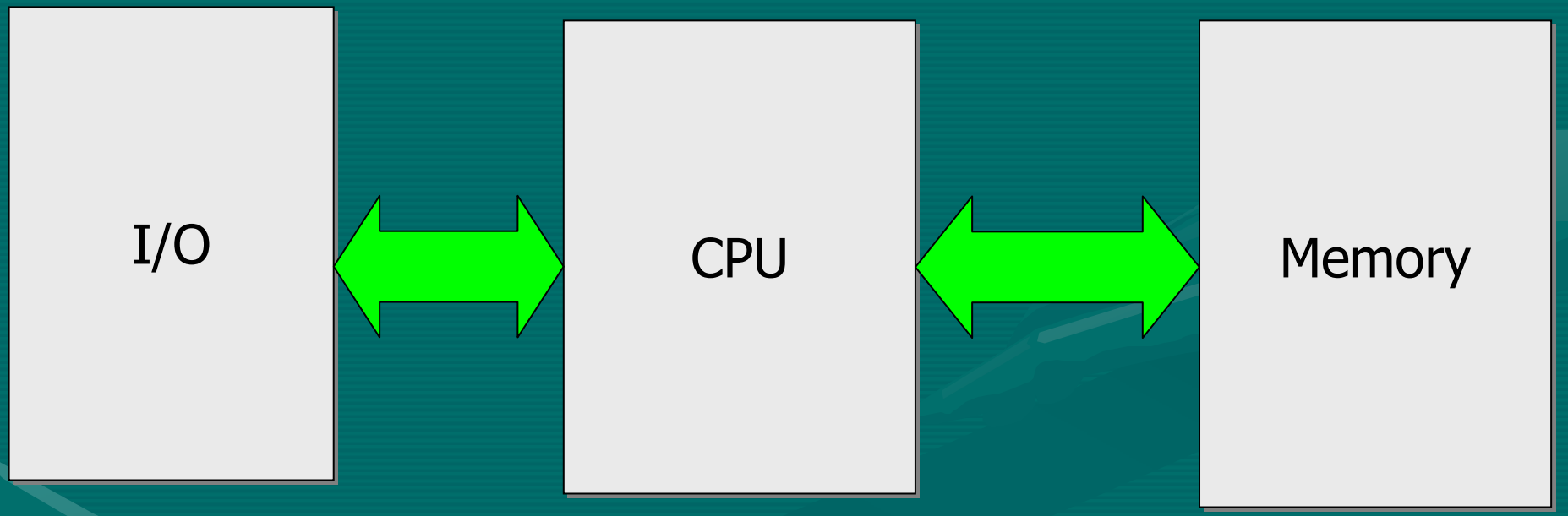

A simple microprocessor architecture

# A Microprocessor based System



# Central Processing Unit and Main Memory of a Computer.

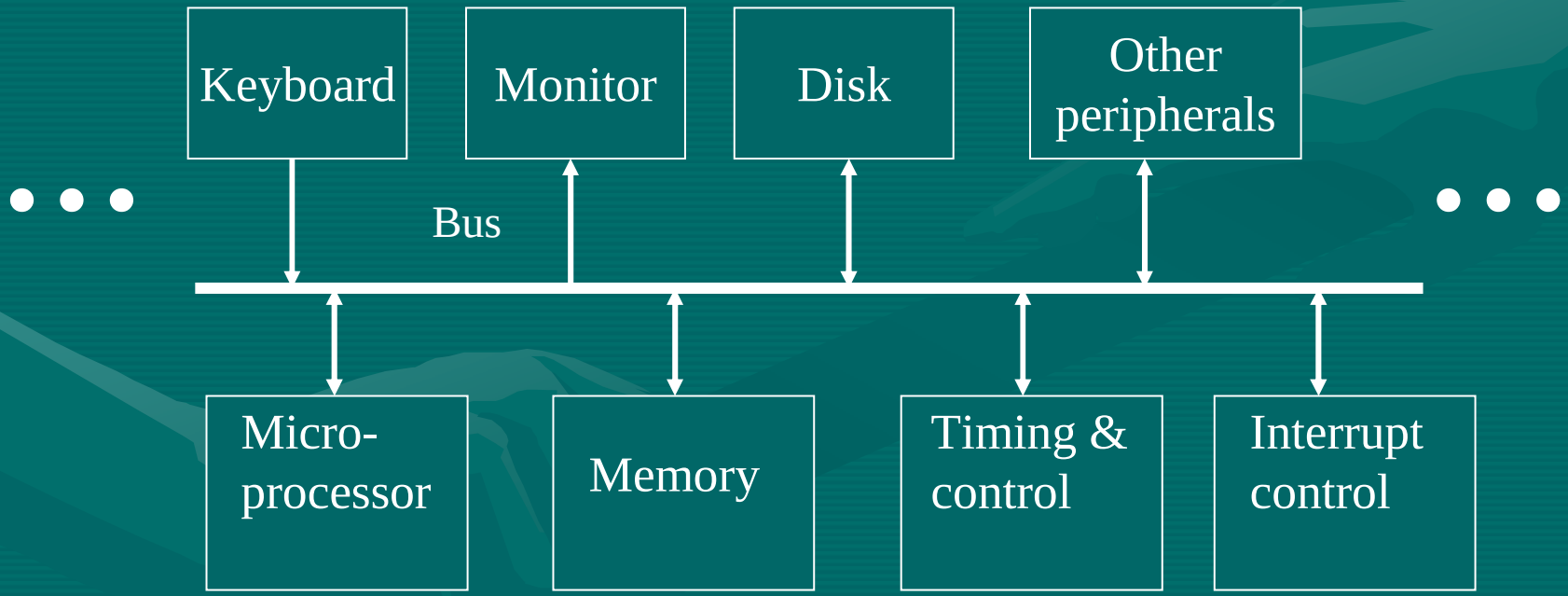





*CPU coordinates transfer between I/O and memory.*

# Applications of Microprocessor-Based Systems

— System performance is normally the most important design concern




Block diagram of a computer

- 8086 is a 16 bit Microprocessor.
- It has a 20 bit address bus can access up to 1 MB memory locations.
- It can support up to 64K I/O ports.
- It provides 14, 16-bit registers.
- It has multiplexed address and data bus $AD_0$-$AD_{15}$ and $A_{16} – A_{19}$.

- It requires single phase clock with 33% duty cycle to provide internal timing.
- 8086 is designed to operate in two modes, Minimum and Maximum mode.
- It can prefetch up to 6 instruction bytes from memory and queues them in order to speed up instruction execution.
- It requires +5V power supply.
- A 40 pin dual in line package.

# Minimum and Maximum Modes:

- The minimum mode is selected by applying logic 1 to the MN / $\overline{MX}$ input pin. This is a single microprocessor configuration.

- The maximum mode is selected by applying logic 0 to the MN / $\overline{MX}$ input pin. This is a multi micro processors configuration.

# Internal Block Diagram of 8086

# Internal Architecture of 8086

- 8086 has two blocks BIU and EU.
- The BIU performs all bus operations such as instruction calculating the addresses of the memory operands, fetching, reading and writing operands etc.
- The instruction bytes are transferred to the instruction queue.
- EU executes instructions from the instruction system byte queue.
- Both units operate asynchronously to give the 8086 an overlapping instruction fetch and execution mechanism which is called as ***Pipelining***.

- This results in efficient use of the system bus and system performance.
- BIU contains Instruction queue, Segment registers, Instruction pointer, Address adder.
- EU contains Control circuitry, Instruction decoder, ALU, Pointer and Index register, Flag register.

# Bus Interface Unit (BIU)

- It provides a full 16 bit bidirectional data bus and 20 bit address bus.

- The bus interface unit is responsible for performing all external bus operations.

***Specifically it has the following functions***:

- Instruction fetch, Instruction queuing, Operand fetch and storage, Address relocation and Bus control.

- The BIU uses a mechanism known as an instruction stream queue to implement a ***pipeline architecture.***

# Instruction stream queue

- This queue permits prefetch of up to six bytes of instruction code.
- When ever the queue of the BIU is not full, it has room for at least two more bytes and at the same time the EU is not requesting it to read or write operands from memory, the BIU is free to look ahead in the program by prefetching the next sequential instruction.
- These prefetching instructions are held in its FIFO queue.
- With its 16 bit data bus, the BIU fetches two instruction bytes in a single memory cycle.
- After a byte is loaded at the input end of the queue, it is automatically shifts up through the FIFO to the empty location nearest the output.

# Execution Unit (EU)

- The EU accesses the queue from the output end. It reads one instruction byte after the other from the output of the queue. If the queue is full and the EU is not requesting access to operand in memory.

- These intervals of no bus activity, which may occur between bus cycles are known as ***Idle state***.

- If the BIU is already in the process of fetching an instruction when the EU request it to read or write operands from memory or I/O, the BIU first completes the instruction fetch bus cycle before initiating the operand read / write cycle.

- The BIU also contains a dedicated adder which is used to generate the 20 bit physical address that is output on the address bus. This address is formed by adding an appended 16 bit segment address and a 16 bit offset address.
- For example, the physical address of the next instruction to be fetched is formed by combining the current contents of the code segment CS register and the current contents of the instruction pointer IP register.
- The BIU is also responsible for generating bus control signals such as those for memory read or write and I/O read or write.

- The Execution unit is responsible for decoding and executing all instructions.

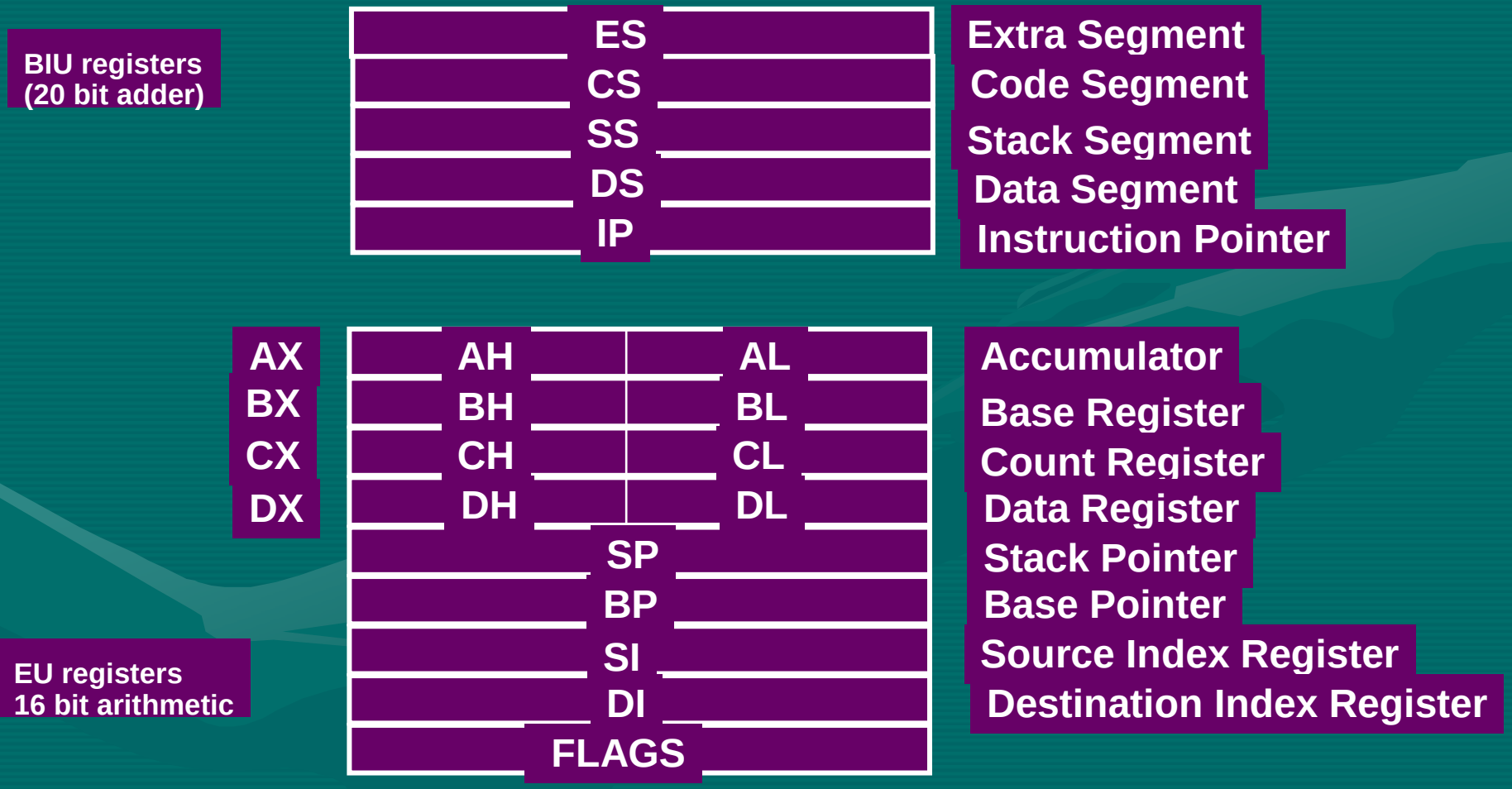
- The EU extracts instructions from the top of the queue in the BIU, decodes them, generates operands if necessary, passes them to the BIU and requests it to perform the read or write bys cycles to memory or I/O and perform the operation specified by the instruction on the operands.

- During the execution of the instruction, the EU tests the status and control flags and updates them based on the results of executing the instruction.

- If the queue is empty, the EU waits for the next instruction byte to be fetched and shifted to top of the queue.
- When the EU executes a branch or jump instruction, it transfers control to a location corresponding to another set of sequential instructions.
- Whenever this happens, the BIU automatically resets the queue and then begins to fetch instructions from this new location to refill the queue.

# Register Organization of 8086

- The 8086 has four groups of the user accessible internal registers.
- They are the
  1. instruction pointer (IP),
  2. four data registers - AX, BX, CX, DX
  3. four pointer and index registers – SP, BP, SI, DI
  4. four segment registers – CS, DS, ES, SS
- The 8086 has a total of fourteen 16-bit registers including a 16 bit register called the ***status register or flag register***, with 9 of bits implemented for status and control flags.


BIU registers
(20 bit adder)
ES
CS
SS
DS
IP
Extra Segment
Code Segment
Stack Segment
Data Segment
Instruction Pointer
AX
BX
CX
DX
AH
AL
BH
BL
CH
CL
DH
DL
SP
BP
SI
DI
FLAGS
Accumulator
Base Register
Count Register
Data Register
Stack Pointer
Base Pointer
Source Index Register
Destination Index Register
EU registers
16 bit arithmetic

# Data Registers

| Register | Operations |
|---|---|
| AX | Word multiply, word divide, word I/O |
| AL | Byte multiply, byte divide, byte I/O, decimal arithmetic |
| AH | Byte multiply, byte divide |
| BX | Store address information |
| CX | String operations, loops |
| CL | Variable shift and rotate |
| DX | Word multiply, word divide, indirect I/O |

- All general registers of the 8086 microprocessor can be used for arithmetic and logic operations. The general registers are:
  - ***Accumulator, AX*** register consists of two 8-bit registers AL and AH, which can be combined together and used as a 16- bit register AX.
  - AL in this case contains the low-order byte of the word, and AH contains the high-order byte.
  - Accumulator can be used for I/O operations and string manipulation.

- ***Base register BX,*** consists of two 8-bit registers BL and BH, which can be combined together and used as a 16-bit register BX.

- BL in this case contains the low-order byte of the word, and BH contains the high-order byte. BX register usually contains a data pointer used for based, based indexed or register indirect addressing.

- ***Count register CX*** consists of two 8-bit registers CL and CH, which can be combined together and used as a 16-bit register CX.

❖When combined, CL register contains the low-order byte of the word, and CH contains the high order byte.

❖Count register can be used in Loop, shift/rotate instructions and as a counter in string manipulation.

- ***Data register DX*** consists of two 8-bit registers DL and DH, which can be combineds together and used as a 16-bit register DX.

- When combined, DL register contains the low-order byte of the word, and DH contains the high order byte.

- Data register can be used as a port number in I/O operations.

- In integer 32-bit multiply and divide instruction the DX register contains high-order word of the initial or resulting number.

# 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
- IP is not under direct control of the programmer

- The following registers are both general and index registers:
- ***Stack Pointer*** *(SP)* is a 16-bit register pointing to program stack.

- ***Base Pointer*** *(BP)* is a 16-bit register pointing to data in stack segment. BP register is usually used for based, based indexed or register indirect addressing.
- ***Source Index*** *(SI)* is a 16-bit register. SI is used for indexed, based indexed and register indirect addressing, as well as a source data address in string manipulation instructions.

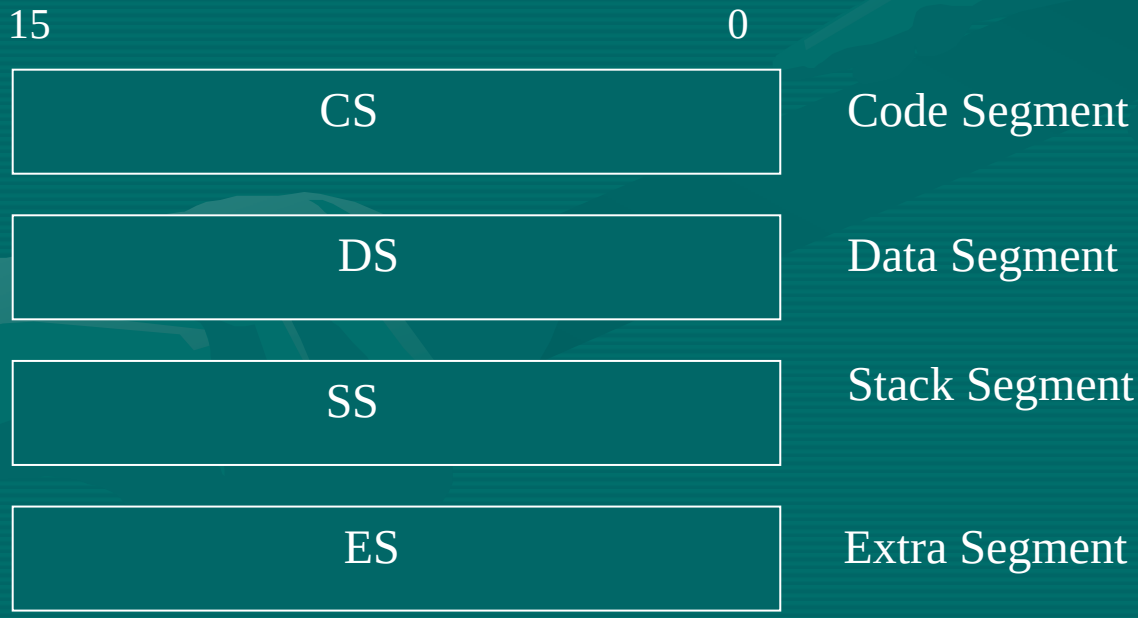
- ***Destination Index*** *(DI)* is a 16-bit register. DI is used for indexed, based indexed and register indirect addressing, as well as a destination data address in string manipulation instructions.

- ***Instruction Pointer*** *(IP)* is a 16-bit register.

**Flags** is a 16-bit register containing 9 one bit flags.

# Segment Registers

- Most of the registers contain data/instruction offsets within 64 KB memory segment.
- There are four different 64 KB segments for instructions, stack, data and extra data.
- To specify where in 1 MB of processor memory these 4 segments are located the processor uses four segment registers:



- ***Code segment (CS)*** is a 16-bit register containing address of 64 KB segment with processor instructions.

- The processor uses CS segment for all accesses to instructions referenced by instruction pointer (IP) register.

- CS register cannot be changed directly. The CS register is automatically updated during far jump, far call and far return instructions.

- ***Stack segment (SS)*** is a 16-bit register containing address of 64KB segment with program stack.

- By default, the processor assumes that all data referenced by the stack pointer (SP) and base pointer (BP) registers is located in the stack segment.

- SS register can be changed directly using POP instruction.

- ***Data segment (DS)*** is a 16-bit register containing address of 64KB segment with program data.

- By default, the processor assumes that all data referenced by general registers (AX, BX, CX, DX) and index register (SI, DI) is located in the data segment.

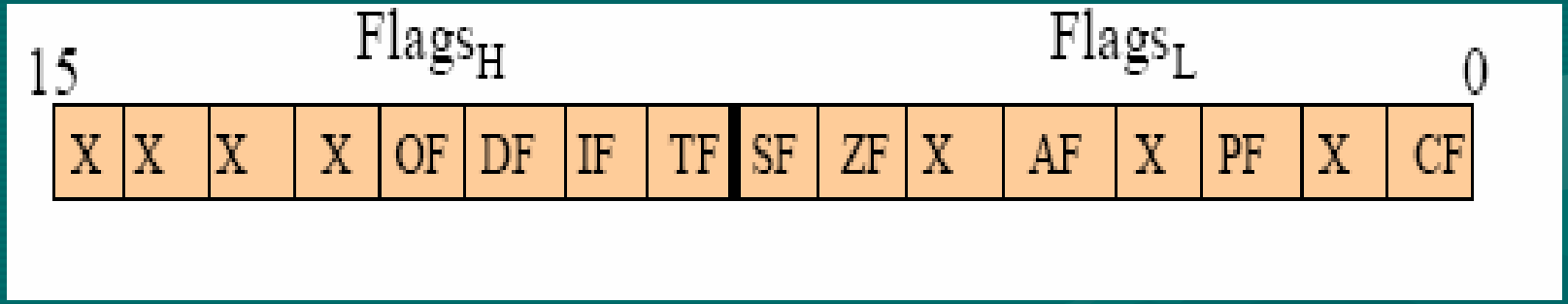
- DS register can be changed directly using POP and LDS instructions.

- ***Extra segment (ES)*** is a 16-bit register containing address of 64KB segment, usually with program data.

- By default, the processor assumes that the DI register references the ES segment in string manipulation instructions. ES register can be changed directly using POP and LES instructions.
- It is possible to change default segments used by general and index registers by prefixing instructions with a CS, SS,DS or ES prefix.

# Flag Register of 8086

Flag register contains information reflecting the current status of a
processor. It also contains information which controls the operation of the microprocessor.




- Status Flags

CF: Carry flag
PF: Parity flag
AF: Auxiliary carry flag
ZF: Zero flag
SF: Sign flag
OF: Overflow flag

- Control Flags

IF: Interrupt enable flag
DF: Direction flag
TF: Trap flag

- **Overflow Flag** (OF) - set if the result is too large positive number, or is too small negative number to fit into destination operand.

- **Direction Flag** (DF) - if set then string manipulation instructions will auto-decrement index registers.

  If cleared then the index registers will be auto-incremented.

- **Interrupt-enable Flag** (IF) - setting this bit enables maskable interrupts.

- **Single-step Flag** (TF) - if set then single-step interrupt will occur after the next instruction.

- **Sign Flag** (SF) - set if the most significant bit of the result is set.
- **Zero Flag** (ZF) - set if the result is zero.
- **Auxiliary carry Flag** (AF) - set if there was a carry from or borrow to bits 0-3 in the AL register.
- **Parity Flag** (PF) - set if parity (the number of "1" bits) in the low-order byte of the result is even.
- **Carry Flag** (CF) - set if there was a carry from or borrow to the most significant bit during last result calculation.

# Addressing Modes of 8086

# 1. Immediate Addressing Mode

- In this addressing immediate data is a part of the instruction.
- The immediate data may be 8-bit or 16-bit in size.
- E.g. MOV AL , 05H

  MOV BX , 3000H
- In the above instructions 05H , 3000H are referred as immediate data.
- Immediate data cannot be a destination operand

# 2. Direct Addressing Mode

- In this addressing mode the offset (16 bit ) address is directly specified in the instruction.
- E.g. MOV  BX , [4000H]
- In the above example the data located in memory pointed by the offset address 4000 and 4001 is loaded in BX.
- Note that the data is taken from two locations as the destination is  16 bit.
- Here we consider data segment as the default segment.

# 3. Register Addressing Mode

- In this addressing mode both the source and the destination operands are registers.
- E.g. MOV  AX , DX
- In the above instruction, the data in the register DX is moved to AX.
- Note that the data from DX is moved i.e., copied to the register AX and not physically moved.
- The contents of both the registers, AX and DX are same after the execution of the instruction.

# 4. Register Indirect Addressing Mode

- It is similar to the direct addressing mode except that the offset address is first stored in a register and then referred using that particular register.
- E.g. MOV AX , [BX]
- In the above example the offset address is first stored in BX register and then referred using that register.
- All registers can be used in this mode except IP

# 5. Indexed Addressing Mode

- In this addressing mode the offset of the operand is stored in any one of the index registers.
- DS is the default segment for index registers SI and DI.
- In case of string instructions DS and ES are the default segments for SI and DI respectively.
- MOV AX , [SI]
- The offset address is stored in SI. The data is available at the location in memory pointed by DS:SI.

# 6. Register Relative Addressing Mode

- In this addressing mode, the data is available by adding an 8-bit or 16-bit displacement with the content of any one of the registers BX, BP, SI, and DI in the default segment (either DS or ES).
- E.g. MOV AX , 50H[BX]
- In the above example the data is located at the location in memory at the physical address computed as follows and stored in AX.
- 10* DS + 50H + [BX]

# 7. Base Indexed Addressing Mode

- In this addressing mode the address of the data is formed by adding the content of a base register ( any one of BX or BP ) to the content of an index register ( any one of SI or DI ) respectively.

The default segment is ES or DS

E.g. MOV AX , [BP][DI]

# 8. Relative Base Indexed Addressing Mode

- In this addressing mode the address of the memory location where the data is located is computed by adding the 8-bit or 16-bit displacement to the content of any one of the base registers ( BX or BP ) and any one of the index registers ( SI or DI ) respectively.
- E.g. MOV AX , 50H[BP][SI]
- In the above example the address of the data is computed as 10*DS + 50H + [BP] + [SI].

- For the control transfer instructions, the addressing modes depend upon the destination location is within the same segment or in a different one.
- It also depends upon the method of passing the destination address to the processor.
- Basically there are two addressing modes for the control transfer instructions
- Intersegment Addr. Mode ( Direct & Indirect)
- Intrasegment Addr. Mode ( Direct & Indirect)

# Memory and Segmentation

- Program, data and stack memories occupy the same memory space.
- As the most of the processor instructions use 16-bit pointers the processor can effectively address only 64 KB of memory.
- To access memory outside of 64 KB the CPU uses special segment registers to specify where the code, stack and data 64 KB segments are positioned within 1 MB of memory

- ***Program memory*** - program can be located anywhere in memory.
- Jump and call instructions can be used for short jumps within currently selected 64 KB code segment, as well as for far jumps anywhere within 1 MB of memory.
- All conditional jump instructions can be used to jump within approximately +127 to -127 bytes from current instruction.
- ***Data memory*** - the processor can access data in any one out of 4 available segments, which limits the size of accessible memory to 256 KB (if all four segments point to different 64 KB blocks).

- Accessing data from the Data, Code, Stack or Extra segments can be usually done by prefixing instructions with the DS:, CS:, SS: or ES: (some registers and instructions by default may use the ES or SS segments instead of DS segment).

- Word data can be located at odd or even byte boundaries.

- The processor uses two memory accesses to read 16-bit word located at odd byte boundaries.
- Reading word data from even byte boundaries requires only one memory access.

- **Stack memory** can be placed anywhere in memory. The stack can be located at odd memory addresses, but it is not recommended for performance reasons.

- **Reserved locations**:

- 0000h - 03FFh are reserved for interrupt vectors.

- Each interrupt vector is a 32-bit pointer in format segment: offset.

- FFFF0h - FFFFFh - after RESET the processor always starts program execution at the FFFF0h address.

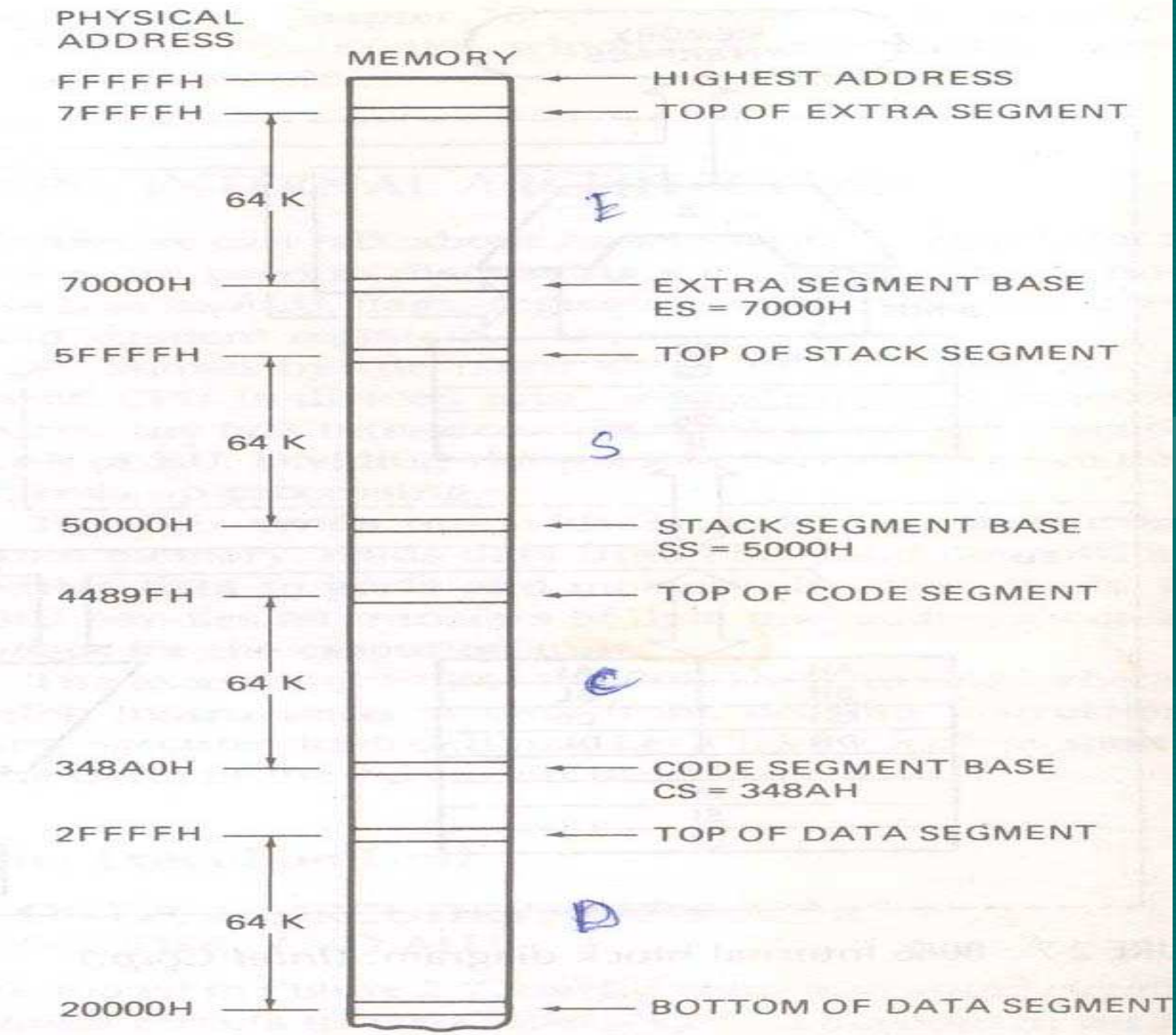
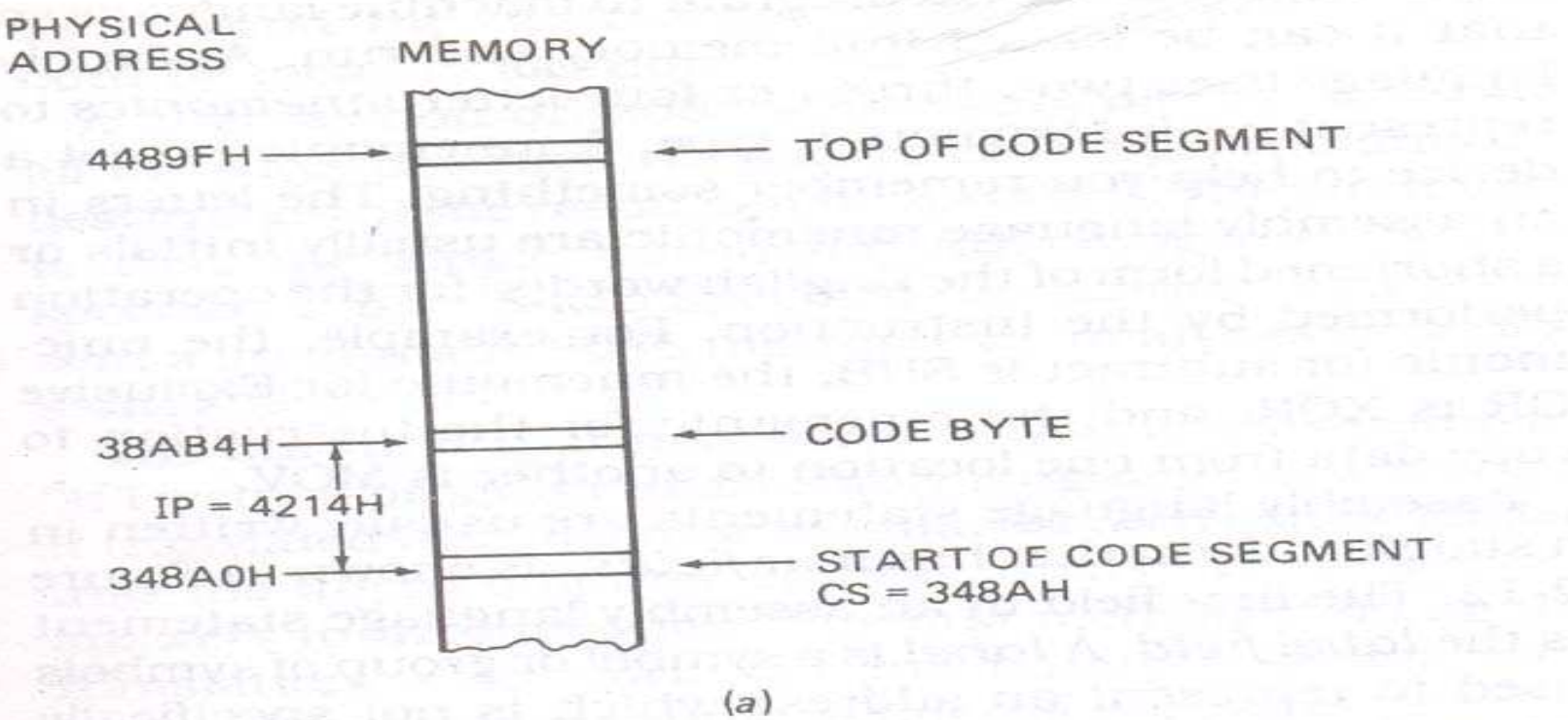



FIGURE 2-9 One way four 64-Kbyte segments might positioned within the 1-Mbyte address space of an 80




(*a*)

| | | | | | | |
|---|---|---|---|---|---|---|
| CS | 3 | 4 | 8 | A | 0 | ← HARDWIRED ZERO |
| IP + | | 4 | 2 | 1 | 4 | |
| PHYSICAL ADDRESS | 3 | 8 | A | B | 4 | |

(*b*)

FIGURE 2-10 Addition of IP to CS to produce the physical address of the code byte. (a) Diagram. (*b*) Computation.