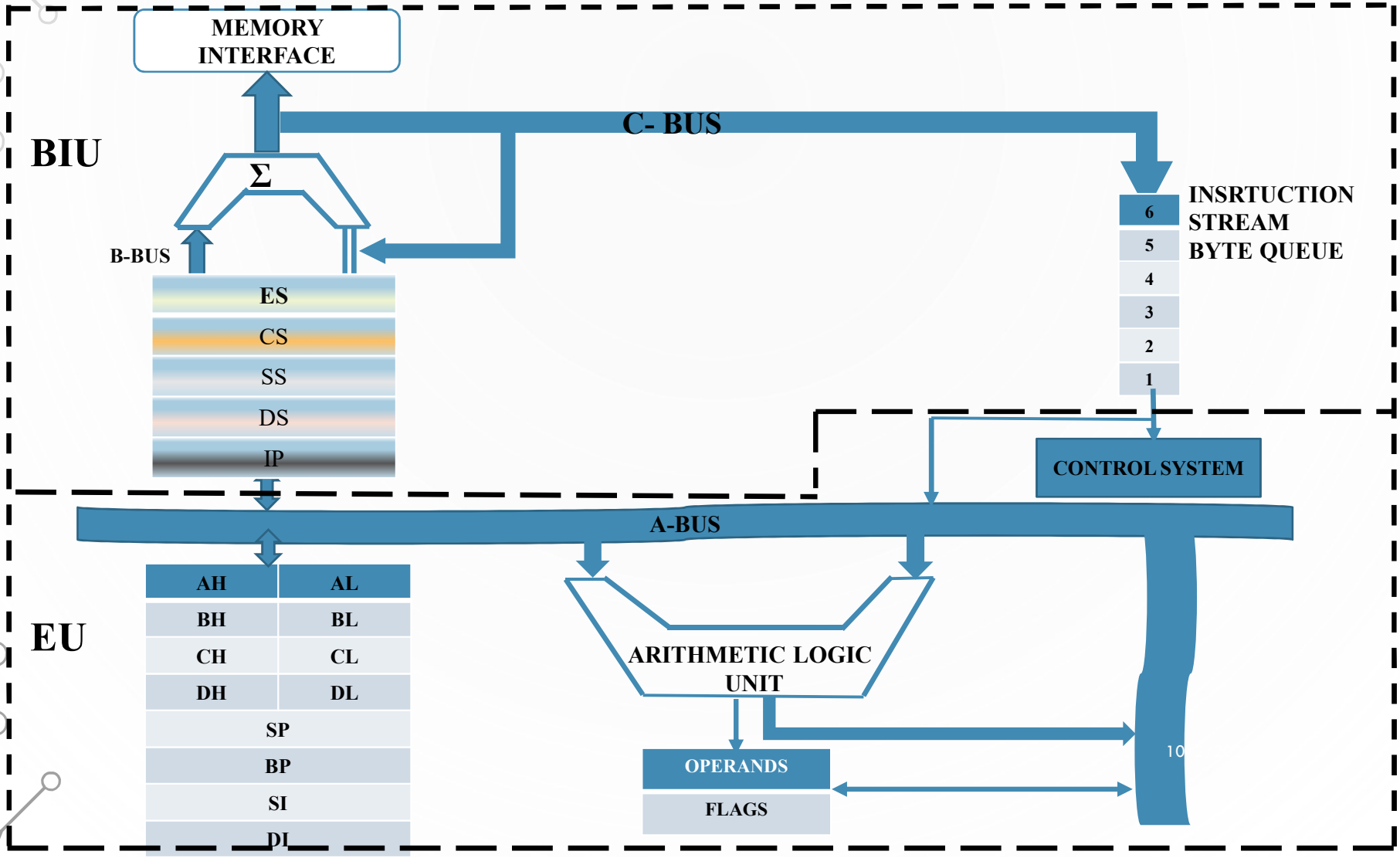
UNIT 3

PROGRAMMING WITH INTEL 8086

INTEL 8086 MICROPROCESSOR

- **8086 Microprocessor Functional Block Diagram:**
  - **16-bit microprocessor (i.e. 16 bit data processing capability).**
  - **16-bit implies that its ALU, its internal registers, and most of its instructions are intended to work with 16 bit binary data.**
  - **16 bit data bus, so it can read data from or write data to memory and ports either 16 bits or 8 bits at a time.**
  - **20 bit wide address bus (can address any one of 2^20, or 1048576 memory locations).**
  - **CPU is divided into 2 independent functional parts:**
    1. **Bus Interface Unit (BIU)**
    2. **Execution Unit (EU)**

8086 MICROPROCESSOR INTERNAL ARCHITECTURE



BIU: It handles all transfers of data and addresses on the buses for the execution unit.

- Sends out addresses
- Fetches instructions from memory
- Read / write data from/to ports and memory i.e. handles all transfers of data and addresses on the busses.

EU

- Tells BIU where to fetch instructions or data from
- Decodes instructions
- Execute instructions

Execution Unit (EU)

1. **Instruction Decoder & ALU:**

- Decoder in the EU translates instructions fetched from the memory into a series of actions which the EU carries out.
- 16-bit ALU in the EU performs actions such as addition, subtraction, AND, OR, XOR, increment, decrement etc.

2. General purpose Registers (GPRs):

- **8 GPRs AH, AL (Accumulator), BH, BL, CH, CL, DH, DL are used to store 8 bit data.**
- **Used individually for the temporary storage of data.**
- **Used together (as register pair) to store 16-bit data words.**
- **Acceptable register pairs are:**
  - **AH-AL** pair **AX** register
  - **BH-BL** pair **BX** register (to store the 16-bit data as well as the base address of the memory location)
  - **CH-CL** pair **CX** register (to store 16-bit data and can be used as counter register for some instructions like loop)
  - **DH-DL** pair **DX** register (to store 16-bit data and also used to hold the result of 16-bit data multiplication and division operation)

3. FLAG REGISTER:

- It is a 16-bit register.
- 9-bit are used as different flags, remaining bits unused

				OF	DF	IF	TF	SF	ZF		AF		PF		CF

Fig: 16-bit flag register

Out of 9-flags, 6 are conditional (status) flags and three are control flags.

Conditional flags:

- Set or reset by the EU on the basis of the results of some arithmetic or logic operation.
- 8086 instructions check these flags to determine which of two alternative actions should be done in executing the instructions.
  - **OF (Overflow flag):** is set if there is an arithmetic overflow, i.e. the size of the result exceeds the capacity of the destination location.
  - **SF (Sign flag):** is set if the MSB of the result is 1.
  - **ZF (Zero flag):** is set if the result is zero.
  - **AF (Auxiliary carry flag):** is set if there is carry from lower nibble to upper nibble or from lower byte to upper byte.
  - **PF (Parity flag):** is set if the result has even parity.
  - **CF (Carry flag):** is set if there is carry from addition or borrow from subtraction.

Control flags:

- They are set using certain instructions.
- They are used to control certain operations of the processor.
  - **TF (Trap flag):** for single stepping through the program.
  - **IF (Interrupt flag):** to allow or prohibit the interruption of a program.
  - **DF (Direction flag):** Used with string instructions.

4. POINTER REGISTERS:

- **SP (Stack Pointer)**
- **BP (Base pointer)**
  - are used to access data in the stack segment.
  - SP is used as offset from current Stack Segment during execution of instruction that involve stack.
  - SP is automatically updated.
  - BP contains offset address and is utilized in based addressing mode.
- **Overall, these are used to hold the offset address of the stack address.**

5. INDEX REGISTERS:

- **SI (Source Index)**
- **DI (Destination index)**
  - **Both 16-bit long.**
  - **used for temporary storage of data similarly as the general purpose registers.**
  - **Specially used to hold the 16-bit offset of the data *word*.**

SI and DI are used to hold the offset address of the data segment and extra segment memory respectively.

BUS INTERFACE UNIT

- MAINLY TWO PARTS:
  1. Instruction queue
  2. Segment registers

1. INSTRUCTION QUEUE

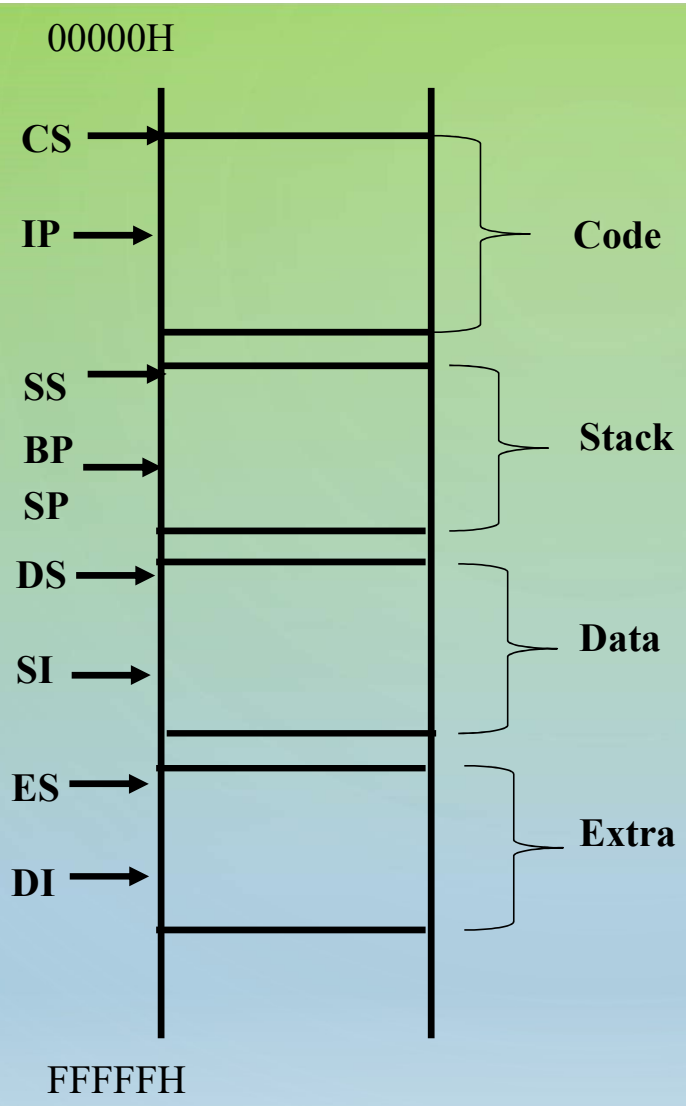
- **BIU pre-fetches up to 6- instructions bytes to be executed and places them in QUEUE.**
- **EU just picks up the fetched instruction byte from the QUEUE.**
- **This improves the overall speed of the system.**
- **The BIU stores these pre-fetched bytes in a first-in-first-out (FIFO) register set called a queue.**
- **Fetching the next instruction while the current instruction executes is called *pipelining.***

2. SEGMENT REGISTERS

- **The BIU contains a dedicated address, which is used to produce the 20 bit address.**
- **The bus control logic of the BIU generates all the bus control signals, such as the READ and WRITE signals, for memory and I/O.**
- **The BIU also has *four 16-bit* segments registers:**
  - **Code segment(CS):** holds the upper 16-bits of the starting addresses of the segment from which BIU is currently fetching instruction code bytes.
  - **Stack segment(SS):** store addresses and data while subprogram executes.
  - **Extra segment(ES):**
    - ➢ used by some string (character data) to handle memory addressing.
    - ➢ store upper 16-bits of starting addresses of two memory segments that are used for data.
  - **Data segment(DS):** store upper 16-bits of starting addresses of two memory segments that are used for data.

MEMORY SEGMENTATION

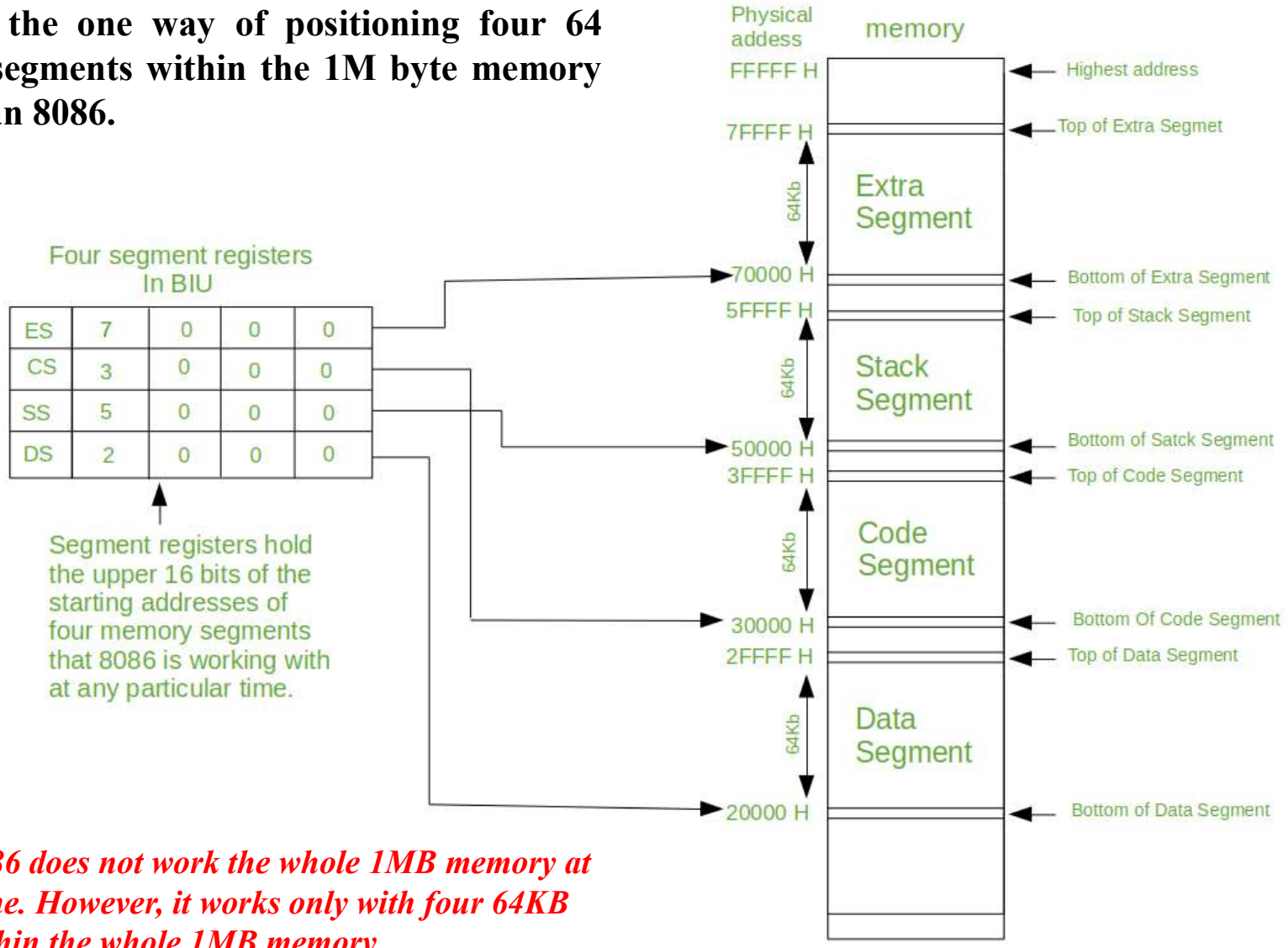
- Address lines in 8086 is 20, BIU will send 20bit address, so as to access one of the 1MB memory locations.
- A segment is a logical unit of memory that may be up to 64 kilobytes long.
- Each segment is made up of contiguous memory locations.
- It is an independent, separately addressable unit.
- Starting address will always be changing and will not be fixed.



2^20=1MB
PA=Seg. Add.*10H+Offset Add.

Assume:
CS= 154EH
IP= 4308H
Then, PA=154EH*10H+4308H
=197E8H

Below is the one way of positioning four 64 kilobyte segments within the 1M byte memory space of an 8086.



Note: the 8086 does not work the whole 1MB memory at any given time. However, it works only with four 64KB segments within the whole 1MB memory.

Types Of Segmentation:

1.Overlapping Segment – A segment starts at a particular address and its maximum size can go up to 64kilobytes. But, if another segment starts before this 64Kbytes location of the first segment, the two segments are said to be overlapping segments.

2.Non-Overlapped Segment – A segment starts at a particular address and its maximum size can go up to 64kilobytes. But if another segment starts along or after this 64kilobytes location of the first segment, then the two segments are said to be *Non-Overlapped Segment*.

Rules of Segmentation:

Segmentation process follows some rules as follows:

1. The starting address of a segment should be such that it can be evenly divided by 16.
2. Minimum size of a segment can be 16 bytes and the maximum can be 64 kB.

Segment	Offset Registers	Function
CS	IP	Address the next instruction
DS	BX, DI, SI	Address the next data
SS	SP, BP	Address the stack
ES	BX, DI, SI	Address of destination data (For string operation)

Advantages of the Segmentation:

The main advantages of segmentation are as follows:

- **Provides a powerful memory management mechanism.**
- **Data related or stack related operations can be performed in different segments.**
- **Code related operation can be done in separate code segments.**
- **Allows to processes easily and share data.**
- **Allows to extend the address ability of the processor, i.e. segmentation allows the use of 16 bit registers to give an addressing capability of 1 Megabytes. Without segmentation, it would require 20 bit registers.**
- **Possible to enhance the memory size of code data or stack segments beyond 64 KB by allotting more than one segment for each area.**

Data formats in 8086

A. ASCII:

- 7-bit code with eighth and MSB used to hold parity in some systems.

B. Binary Coded Decimal (BCD):

- Used to store packed or unpacked data/information in memory.
- Packed BCD occurs when two BCD digits are stored per memory byte, unpacked BCD occurs when one BCD is stored per byte.

C. Byte: ***Byte*** data are stored in two forms; unsigned and signed integers.

D. Word:

- 2-byte data
- Least significant byte is always stored in the lowest numbered memory location and most significant byte in the highest.

E. Double word:

- Used to store 32-bit numbers (4bytes) that are product of multiplication or division.

F. Real numbers: Also called floating point numbers, composed of *mantissa* and *exponent*.

INSTRUCTIONS IN 8086

- Instructions set found in 8086 are equivalent to 8085 but with the increase in operations variety, some few new instructions are added.
- Can be categorized as:
  - ✓Arithmetic,
  - ✓Data transfer,
  - ✓Logical and Bit manipulation,
  - ✓String,
  - ✓Program transfer and
  - ✓Process control.

1. ARITHMETIC INSTRUCTIONS

- **ADD (Addition):** Add specified byte to byte or specified word to word.
  - ADD R(8) /M(8) , R(8)/M(8)/ Immediate(8)
  - ADD R(16)/M(16), R(16)/ M(16)/ Immediate(16)
  - E.g. ADD AH, 15h; It adds contents of register AH and 15h immediately.
    - ADD AH, NUM1
    - ADD Al, [BX]
    - ADD [BX], CH/CX
    - ADD AX, [BX]

- **ADC:** Addition with Carry
  - **ADC** R/ M, R/M/Immediate data
- **SUB (Subtraction):**
  - SUB R/ M, R/M/Immediate data
- **SBB** (Subtraction with Borrow):
  - SBB R/ M, R/M/Immediate data
- **MUL**: unsigned multiplication
  - MUL R(8)/M(8) (8-bit Accumulator, AL)
  - MUL R(16)/ M(16) (16-bit Accumulator, Ax)
  - E.g. MUL R(8)(multiplier); AX (16-bit result)←R(8)×AL

    MUL R(16) (multiplier); DX:AX (32-bit result)←R(16)×AX

- **IMUL**: signed multiplication
  - *Same as MUL operation but takes sign into account.*
- **DIV (Division):**
  - DIV R(8); AX/R(8) Remainder in AH and Quotient in AL
  - DIV R(16); DX:AX/R(16) Remainder in DX and Quotient in AX
- **IDIV (Signed Division):**
  - Same operation as DIV but takes sign into account.
- **INC/DEC (Increment/Decrement by 1):**
  - Increment/decrement the Register content by unity.
  - INC/DEC R/M, (8-bit or 16-bit)
  - E.g.: INC AL, DEC BX, INC num1

- **NEG (Negate – 2's Complement):**

ASCII-BCD conversion:

- ***AAA (ASCII adjust after addition)***
  - Corrects result in AH and AL after addition when working with BCD values.
- ***DAA (decimal adjust after addition)***
  - Corrects the result of addition of two packed BCD values
- ***DAS (decimal adjust after subtraction)***
- ***AAS (ASCII adjust after subtraction)***
- ***AAM (adjust after multiplication)***
- ***AAD (adjust after division)***

LOGICAL/SHIFTING/COMPARISON INSTRUCTIONS

- **Logical**
  - ✓AND/OR/XOR R/M, R/M/Immediate
  - ✓NOT R/M; Invert each bit of byte or word.
  - ✓E. g. AND AL, AH
  - ✓XOR [BX], CL

- **Rotation**
  - ROL/ROR/RCL/RCR**,** R/M, 1/CL
  - *ROL- rotate left,*
  - *ROR-rotate right*
  - E.g. ROL AX, 1; rotated by 1
  - ROL AX, CL; if we need to rotate more than one bit
  - *RCL-rotate left through carry*
  - *RCR-rotate right through carry*
  - E.g. RCL AX, 1
  - RCL [BX], CL; Only CL can be used.

- **Shifting**
  - SHL/SHR/SAL/SAR, R/M,1/CL
  - ***SHL - logical shift left***
  - ***SHR - logical shift right***
  - *Shifts bit in true direction and fills zero in vacant place.*
  - ***SAL - arithmetic shift left***
  - ***SAR - arithmetic shift right***
  - *Shifts bit/word in true direction, in former case place zero in vacant place and in later case place previous sign in vacant place.*
  - E.g. SHL AX, 1; rotated by 1
  - SHL AX, CL; if we need to rotate more than one bit
  - SAR DX, 1
  - SAR [BX], CL; Only CL can be used.

- **Comparison**
  - CMP –compare
  - CMP R/M, R/M/Immediate
  - E.g. CMP BH, AL

Operand1		Operand 2	CF	SF	ZF
	>		0	0	**0**
	=		0	0	**1**
	<		**1**	**1**	**0**

DATA TRANSFER INSTRUCTIONS

- move bytes or words of data between memory and registers as well as between the registers and the I/O ports.
- **MOV** R/M, R/M/Immediate
  - Copy byte or word from specified source to specified destination.
- **IN** AL/AX, DX
  - Copy a byte or word from specified port number to accumulator.
  - Second operand (DX) is a port number. If required to access port number over 255 - **DX** register should be used.
  - E.g.: IN AL, DX
    OUT DX, AL/AH

- **OUT** DX, AL/AX
  - Copy a byte or word from accumulator to specified port number, first operand (DX) is a port number. If required to access port number over 255 - **DX** register should be used.
- **LEA** R, M
  - Load effective address of operand into specified register
- **LDS** BX, NUM1 (Load data segment register)
- **LES (**Load extra segment register)
- **LSS** (load stack segment register)

- **XCHG** R/M, R/M/immediate
  - E.g.: XCHG AX, BX

    XCHG AL, BL

    XCHG CL, [BX]

- **Flag Operation:**
  - CLC: Clear carry flag
  - CLD: Clear direction flag
  - CLI: Clear interrupt flag
  - STC: Set Carry flag
  - STD: Set direction flag
  - STI: Set Interrupt flag
  - CMC: Complement Carry flag
  - LAHF: Load AH from flags (lower byte)
  - SAHF: Store AH to flags
  - PUSHF: Push flags into stack
  - POPF: Pop flags off stack

- **STACK Operations:**
  - PUSH R (16-bit)
  - POP R (16-bit)
- **Looping instruction:** Register CX is automatically used as a counter.
  - LOOP: loop until complete
  - LOOPE: Loop while equal
  - LOOPZ: loop while zero
  - LOOPNE: loop while not equal
  - LOOPNZ: loop while not zero

BRANCHING INSTRUCTION:

- **Conditional**
  - JA (Jump if above)
  - JAE (Jump if above/equal)
  - JB (Jump if Below)
  - JBE (Jump if below/equal)
  - JC (Jump if carry)
  - JNC (Jump if no carry)
  - JE (Jump if equal)
  - JNE (Jump if not equal)
  - JZ (Jump if zero)
  - JNZ (Jump if no zero)
  - JG (Jump if greater)
  - JNG (Jump if no greater)
  - JL (Jump if less)
  - JNL (Jump if no less)
  - JO (Jump if overflow)
  - JS (Jump if sign)
  - JNS (Jump if no sign)
  - JP (Jump if plus)
  - JPE (Jump if even parity)
  - JNP (Jump if no parity)
  - JPO (Jump if parity odd)

UNCONDITIONAL

- CALL (Call a procedure)
- INT (Interrupt)
- JMP (Unconditional jump)
- RETN/RETF (Return near/far)
- RET (Return)
- IRET (Interrupt return)

- **Type conversion:**
  - **CBW (Convert byte to word)**
  - **CWD (Convert word to double word)**
- **String instructions:**
  - MOVS/MOVSB/MOVSW; Move string
  - *DS: SI (Source)*
  - *DS: DI (Destination)*
  - *CX (String Length)*
  - CMPS/CMPSB/CMPW; Compare string
  - LODS/LODSB/LODW; Load string
  - REP; Repeat string