**Chapter 04**

**1. The first byte of an instruction is the \_\_\_\_\_\_\_\_\_\_\_\_, unless it contains one of the override prefixes.**

✅ **Answer:** The **opcode**.  
🧠 *Explanation:* The opcode specifies the operation the processor will perform.

**2. Describe the purpose of the D- and W-bits found in some machine language instructions.**

✅ **Answer:**

* **D-bit (Direction bit):** Determines the direction of data transfer — whether data moves **from register to memory** or **from memory to register**.
* **W-bit (Width bit):** Indicates the size of data — **0 for 8-bit** and **1 for 16-bit (or 32-bit)** operations.

**3. In a machine language instruction, what information is specified by the MOD field?**

✅ **Answer:**  
The **MOD field** specifies the **addressing mode** — whether the operand is a **register**, a **memory address**, or a **memory address with displacement**.

**4. If the register field (REG) of an instruction contains 010 and W = 0, what register is selected, assuming that the instruction is a 16-bit mode instruction?**

✅ **Answer:**  
Register selected → **DL** (8-bit register).  
🧠 *Because when W = 0, it means 8-bit, and REG = 010 corresponds to DL.*

**5. How are the 32-bit registers selected for the Pentium 4 microprocessor?**

✅ **Answer:**  
By using the **operand-size prefix (66H)** or operating in **32-bit mode**, which selects **32-bit registers (EAX, EBX, ECX, etc.)** instead of 16-bit registers.

**6. What memory-addressing mode is specified by R/M = 001 with MOD = 00 for a 16-bit instruction?**

✅ **Answer:**  
Addressing mode → **[BX + DI]**  
🧠 *Because MOD = 00 and R/M = 001 corresponds to the BX + DI effective address in 16-bit mode.*

**7. Identify the default segment registers assigned to the following:**

| **Register** | **Default Segment Register** |
| --- | --- |
| (a) SP | SS (Stack Segment) |
| (b) EBX | DS (Data Segment) |
| (c) DI | DS (Data Segment) |
| (d) EBP | SS (Stack Segment) |
| (e) SI | DS (Data Segment) |

**8. Convert 8B07H from machine language to assembly language.**

✅ **Answer:**  
MOV AL, [BX]  
🧠 *Opcode 8B = MOV, with addressing mode using BX as the base register.*

**9. Convert 8B9E004CH from machine language to assembly language.**

✅ **Answer:**  
MOV BX, [SI+4C00H]  
🧠 *Opcode 8B = MOV, and displacement 4C00H is added to SI.*

**10. If a MOV SI,[BX+2] instruction appears in a program, what is its machine language equivalent?**

✅ **Answer:**  
8B 70 02  
🧠 *Opcode 8B (MOV r16, r/m16), ModR/M byte encodes [BX+2], and 02 is the displacement.*

**11. If a MOV ESI,[EAX] instruction appears in a program for the Core2 microprocessor operating in the 16-bit instruction mode, what is its machine language equivalent?**

✅ **Answer:**  
67 8B 30  
🧠 *Prefix 67H switches addressing mode, 8B = MOV, and ModR/M encodes ESI ← [EAX].*

**12. What is the purpose of REX?**

✅ **Answer:**  
The **REX prefix** is used in **64-bit mode** to enable access to **extended registers (R8–R15)** and specify **64-bit operand size**.

**13. What is wrong with a MOV CS,AX instruction?**

✅ **Answer:**  
You **cannot move data directly into the Code Segment (CS)** using MOV.  
🧠 *CS can only be changed using FAR JMP, FAR CALL, or RET.*

**14. Form a short sequence of instructions that load the data segment register with 1000H.**

✅ **Answer:**

MOV AX, 1000H

MOV DS, AX

🧠 *You must first load the value into AX, then move AX into DS.*

**15. The PUSH and POP instructions always transfer a(n) \_\_\_\_\_\_\_\_\_\_\_\_-bit number between the stack and a register or memory location in the 80386–Core2 microprocessors when operated in the 32-bit mode.**

✅ **Answer:** **32-bit** number.

**16. Create an instruction that places RAX onto the stack in the 64-bit mode for the Pentium 4.**

✅ **Answer:**

PUSH RAX

**17. What segment register may not be popped from the stack?**

✅ **Answer:** **CS (Code Segment)**

**18. Which registers move onto the stack with the PUSHA instruction?**

✅ **Answer:**  
AX, CX, DX, BX, SP, BP, SI, DI (in that order).

**19. Which registers move onto the stack for a PUSHAD instruction?**

✅ **Answer:**  
EAX, ECX, EDX, EBX, ESP, EBP, ESI, EDI

**20. Describe the operation of each of the following instructions:**

| **Instruction** | **Description** |
| --- | --- |
| (a) **PUSH AX** | Pushes the contents of AX onto the stack. |
| (b) **POP ESI** | Pops the top of the stack into ESI. |
| (c) **PUSH [BX]** | Pushes the data from the memory location pointed to by BX onto the stack. |
| (d) **PUSHFD** | Pushes the EFLAGS register onto the stack. |
| (e) **POP DS** | Pops the top of the stack into DS (data segment). |
| (f) **PUSHD 4** | Pushes the 32-bit immediate value **4** onto the stack. |

**21. Explain what happens when the PUSH BX instruction executes. Make sure to show where BH and BL are stored. (Assume SP = 0100H and SS = 0200H.) (Exam)**

✅ **Answer:**

* The **PUSH BX** instruction stores the contents of BX onto the stack.
* SP is **decreased by 2** (because BX is 16-bit).
* BH and BL are stored as follows:
  + Then, **BH** is stored at **[SS:00FFH]** and **BL** at **[SS:00FEH]**  
    🧠 Physical address = SS × 10H + SP = 0200H × 10H + 0100H = **02000H + 0100H = 02100H**.

**22. Repeat question 21 for the PUSH EAX instruction.**

✅ **Answer:**

* **PUSH EAX** pushes the 32-bit contents of EAX onto the stack.
* SP (or ESP in 32-bit mode) is **decreased by 4**.
* The bytes are stored in order:
  + **[SS:SP] ← AL**
  + **[SS:SP+1] ← AH**
  + **[SS:SP+2] ← upper bytes of EAX**
  + **[SS:SP+3] ← highest byte of EAX**

**23. The 16-bit POP instruction (except for POPA) increments SP by \_\_\_\_\_\_\_\_\_\_\_\_.**

✅ **Answer:** **2 bytes**

**24. What values appear in SP and SS if the stack is addressed at memory location 02200H?**

✅ **Answer:**  
If stack base = 02200H,  
then **SS = 0220H** and **SP = 0000H**  
🧠 So the stack begins at physical address 02200H × 10H + 0000H = 22000H.

**25. Compare the operation of a MOV DI,NUMB instruction with an LEA DI,NUMB instruction.**

✅ **Answer:**

* **MOV DI, NUMB** loads the **contents (value)** stored at label NUMB into DI.
* **LEA DI, NUMB** loads the **address (offset)** of NUMB into DI.

🧠 MOV → gets the data.  
LEA → gets the address.

**26. What is the difference between an LEA SI,NUMB instruction and a MOV SI,OFFSET NUMB instruction?**

✅ **Answer:**  
There is **no operational difference** — both load the **offset address** of NUMB into SI.  
However, **LEA** can also calculate addresses with registers (e.g. [BX+DI+5]), while **OFFSET** works only with labels.

**27. Which is more efficient, a MOV with an OFFSET or an LEA instruction?**

✅ **Answer:**  
**MOV with OFFSET** is more efficient — it uses **fewer CPU cycles** because the assembler directly substitutes the address value.

**28. Describe how the LDS BX,NUMB instruction operates.**

✅ **Answer:**  
**LDS (Load Pointer using DS)** loads:

* The **offset** part of NUMB into **BX**
* The **segment** part (next word) into **DS**

🧠 So LDS loads both offset and data segment address.

**29. What is the difference between the LDS and LSS instructions?**

✅ **Answer:**

* **LDS** loads destination register and **DS (Data Segment)**.
* **LSS** loads destination register and **SS (Stack Segment)**.

**30. Develop a sequence of instructions that moves the contents of data segment memory locations NUMB and NUMB+1 into BX, DX, and SI.**

✅ **Answer:**

MOV BX, [NUMB]

MOV DX, [NUMB+1]

MOV SI, [NUMB]

**31. What is the purpose of the direction flag?**

✅ **Answer:**  
The **Direction Flag (DF)** controls the direction of string operations:

* **DF = 0 →** Process strings **forward** (increment SI/DI).
* **DF = 1 →** Process strings **backward** (decrement SI/DI).

**32. Which instructions set and clear the direction flag?**

✅ **Answer:**

* **STD** → Set Direction Flag (DF = 1)
* **CLD** → Clear Direction Flag (DF = 0)

**33. Which string instruction(s) use both DI and SI to address memory data?**

✅ **Answer:**  
**MOVS**, **CMPS**, and **SCAS** use both **DI (Destination Index)** and **SI (Source Index)**.

**34. Explain the operation of the LODSB instruction.**

✅ **Answer:**

* **LODSB** loads a byte from memory location **[DS:SI]** into **AL**.
* After loading:
  + If **DF = 0**, SI = SI + 1
  + If **DF = 1**, SI = SI − 1

**35. Explain the operation of the LODSQ instruction for the 64-bit mode of the Pentium 4 or Core2.**

✅ **Answer:**

* **LODSQ** loads a 64-bit value from memory **[RSI]** into **RAX**.
* Then increments or decrements **RSI** by 8 bytes depending on the **Direction Flag**.

**36. Explain the operation of the OUTSB instruction.**

✅ **Answer:**

* **OUTSB** outputs a byte from memory location **[DS:SI]** to the port addressed by **DX**.
* SI is then incremented or decremented depending on DF.

**37. Explain the operation of the STOSW instruction.**

✅ **Answer:**

* **STOSW** stores the **word in AX** into the memory location **[ES:DI]**.
* Then **DI** is incremented or decremented by 2 depending on the **Direction Flag**.

**38. Develop a sequence of instructions that copy 12 bytes of data from an area of memory addressed by SOURCE into an area of memory addressed by DEST.**

✅ **Answer:**

MOV CX, 12 ; number of bytes to copy

LEA SI, SOURCE ; source address

LEA DI, DEST ; destination address

CLD ; clear direction flag

REP MOVSB ; repeat move 12 times

**39. What does the REP prefix accomplish and what type of instruction is it used with?**

✅ **Answer:**

* **REP** repeats a string instruction until **CX (or ECX/RCX)** becomes zero.
* Used with **string instructions** like **MOVS**, **STOS**, **CMPS**, etc.

**40. Select an assembly language instruction that exchanges the contents of the EBX register with the ESI register.**

✅ **Answer:**

XCHG EBX, ESI.

**41. Where is the I/O address (port number) stored for an INSB instruction?**

**Answer:** In register **DX**.  
INSB reads a byte from the I/O port whose port number is in **DX** and stores it at **[ES:DI]** (then updates DI according to DF).

**42. Would the LAHF and SAHF instructions normally appear in software?**

**Answer:** **Rarely** in modern high-level application code.  
They are legacy instructions (load/store AH with low byte of FLAGS) used mainly in low-level, legacy, or compatibility code and some device/interrupt handlers. Modern compilers usually avoid them.

**43. Short program (using XLAT) to convert BCD 0–9 to ASCII (30H–39H) and store ASCII in a TABLE (data segment).**

**Answer (example 16-bit MASM-style):**

; Assume: input BCD digits are in array DIGITS (each 0..9), count in CX

; TABLE contains ASCII '0'..'9' (30H..39H)

MOV BX, OFFSET TABLE ; BX points to table base (DS:BX)

CONVERT\_LOOP:

MOV AL, [SI] ; get BCD digit (0..9) from source at DS:SI

XLAT ; AL := byte ptr [BX + AL] → ASCII code

MOV [DI], AL ; store ASCII at DS:DI (destination)

INC SI

INC DI

LOOP CONVERT\_LOOP

**Data (example):**

TABLE DB '0','1','2','3','4','5','6','7','8','9' ; ASCII 30h..39h

**44. Explain how the XLAT instruction transforms the contents of the AL register.**

**Answer:** XLAT replaces AL with the byte at memory address DS:BX + AL.  
Effectively: AL ← [DS : (BX + AL)]. (BX is the table base; AL is used as index.)

**45. Explain what the IN AL,12H instruction accomplishes.**

**Answer:** Reads a byte from I/O port **12H** (immediate 8-bit port) into **AL**.

**46. Explain how the OUT DX,AX instruction operates.**

**Answer:** Writes the word (16 bits) in **AX** to the I/O port whose port number is in **DX**. (If in 32/64-bit modes, still uses AX as the 16-bit value to send.)

**47. What is a segment-override prefix?**

**Answer:** A machine-code prefix that tells the processor to use a **non-default segment register** for the following memory operand (e.g., ES:, CS:, SS:, DS:). Encoded as specific prefix byte(s) before the instruction.

**48. Select an instruction that moves a byte from memory addressed by BX in the extra segment into AH.**

**Answer:**

MOV AH, ES:[BX]

(or equivalently MOV AH, [ES:BX])

**49. Develop a sequence that exchanges AX↔BX, ECX↔EDX, SI↔DI.**

**Answer:**

XCHG AX, BX

XCHG ECX, EDX

XCHG SI, DI

**50. What is an assembly-language directive?**

**Answer:** A statement for the **assembler** (not executed by CPU) that controls assembly: defines data, reserves storage, sets segments, defines constants, procedure boundaries, etc. (Examples: DB, DW, SEGMENT, PROC, ENDP, EQU.)

**51. What is accomplished by CMOVNE CX,DX on Pentium 4?**

**Answer:** Conditional move: if **ZF = 0** (not equal), move **DX → CX**; otherwise leave **CX** unchanged. (No flags are modified.)

**52. Purpose of directives DB, DW, and DD.**

**Answer:**

* DB — **Define Byte** (allocate/initialize 8-bit values).
* DW — **Define Word** (allocate/initialize 16-bit values).
* DD — **Define Doubleword** (allocate/initialize 32-bit values).

**53. Directive that reserves 30 bytes for array LIST1.**

**Answer (MASM):**

LIST1 DB 30 DUP (?)

This reserves 30 uninitialized bytes.

**54. Describe the purpose of the EQU directive.**

**Answer:** name EQU value creates a constant — the assembler substitutes value wherever name appears. (It's a compile-time symbolic constant.)

**55. What is the purpose of the .686 directive?**

**Answer:** Tells the assembler to enable instructions/features from the **686 (Pentium Pro/Pentium II/III-era)** instruction set (e.g., conditional moves). It enables assembler acceptance/encoding of those extended opcodes.

**56. What is the purpose of the .MODEL directive?**

**Answer:** In MASM it specifies the **memory model** (tiny, small, medium, compact, large, huge) which affects segment usage and default code/data sizes and how far procedures/data are referenced.

**57. If the start of a segment is identified with .DATA, what type of memory organization is in effect?(EXAM)**

**Answer:** **Data segment** — area reserved for program data (variables, arrays, constants).

**58. If the SEGMENT directive identifies the start of a segment, what type of memory organization is in effect?**

**Answer:** Depends on the **segment name** used (e.g., CODE SEGMENT → code segment, DATA SEGMENT → data segment, STACK SEGMENT → stack). SEGMENT itself starts whatever named segment follows.

**59. What does INT 21H accomplish if AH = 4CH?**

**Answer:** DOS terminate program / process: **Terminate with return code** in **AL**. Control returns to DOS with AL as exit code.

**60. What directives indicate the start and end of a procedure?(EXAM)**

**Answer:** PROC (or PROC NEAR / PROC FAR) begins a procedure; ENDP ends it.

**61. Explain purpose of the USES statement (MASM 6.x) for a procedure.**

**Answer:** USES reglist causes the assembler to emit **PUSH** instructions for the listed registers at procedure entry and corresponding **POP** instructions before return — automatically saving/restoring registers for the procedure.

**62. Develop a near procedure that stores AL in four consecutive data-segment locations addressed by DI.**

**Answer (MASM 16-bit example):**

Store4 PROC NEAR

MOV [DI], AL

MOV [DI+1], AL

MOV [DI+2], AL

MOV [DI+3], AL

RET

Store4 ENDP

**63. How is the Pentium 4 instructed to use the 16-bit instruction mode?**

**Answer:** Two ways:

* Run the CPU in **real mode** (native 16-bit instruction decoding).
* In protected/compatibility modes, use the **operand-size (66h)** and **address-size (67h)** prefixes to select 16-bit instruction/address encoding for the following instruction(s). (OS also controls the CPU mode.)

**64. Develop a far procedure that copies the contents of the word-sized memory location CS:DATA4 into AX, BX, CX, DX, SI.**

**Answer (MASM-style):**

CopyFromCS PROC FAR

MOV AX, WORD PTR CS:DATA4 ; load word at CS:DATA4 into AX

MOV BX, AX

MOV CX, AX

MOV DX, AX

MOV SI, AX

RET FAR

CopyFromCS ENDP

If you want, I can:

* Convert these to a **compact printable cheat-sheet**, or
* Expand any answer with **machine-code bytes** or a **memory diagram** for visualization. Which would help you most?