1. What is a short JMP?

A **short jump** is a jump instruction that transfers control **within –128 to +127 bytes** from the current instruction pointer (IP).
It uses **8-bit displacement**.

---

2. Which type of JMP is used when jumping to any location within the current code segment?

A **near jump** is used to jump **anywhere within the current code segment**.

---

3. Which JMP instruction allows the program to continue execution at any memory location in the system?

A **far jump** allows jumping to **any memory location in the system**, because it changes both **CS (Code Segment)** and **IP (Instruction Pointer)**.

---

4. Which JMP instruction is 5 bytes long?

A **far jump** is **5 bytes long** (2 bytes for IP + 2 bytes for CS + 1 byte opcode).

---

5. What is the range of a near jump in the 80386–Core2 microprocessors?

A **near jump** uses a **16-bit or 32-bit displacement**, allowing a range of:

- **16-bit mode:** ±32,767 bytes
- **32-bit mode:** ±2,147,483,647 bytes (±2 GB)

---

6. Which type of JMP instruction assembles for the following distances:

(a)	Distance = 0210H (528) bytes	→ **Near jump**
(b)	Distance = 0020H (32) bytes	→ **Short jump**
(c)	Distance = 10000H (65,536) bytes	→ **Far jump**

---

7. What can be said about a label that is followed by a colon?

It indicates a **symbolic name for an address** (a **target label**) in assembly.
Example:

LOOP_START:

→ Defines a label for jumps or calls.

---

8. The near jump modifies the program address by changing which register(s)?

It changes **only the Instruction Pointer (IP)** (in 16-bit mode) or **EIP/RIP** (in 32/64-bit mode).

---

9. The far jump modifies the program address by changing which register(s)?

It changes **both the Code Segment (CS)** and **Instruction Pointer (IP)** registers.

---

10. Explain what the JMP AX instruction accomplishes. Also identify it as a near or a far jump instruction.

JMP AX → Copies the content of **AX** into **IP** → execution continues at that address in the same segment.
✅ It's a **near jump** (within the same code segment).

---

11. Contrast JMP DI with JMP [DI].

Instruction	Meaning	Type
JMP DI	Jumps to the address contained **directly in DI register**	Near (register indirect)
JMP [DI]	Jumps to the **memory location** pointed to by DI	Near (memory indirect)

---

12. Contrast JMP [DI] with JMP FAR PTR [DI].

Instruction	Description	Segment Change
JMP [DI]	Jump to the address stored at memory pointed by DI (IP only)	No

Instruction	Description	Segment Change
JMP FAR PTR [DI]	Jump to far address stored in memory (changes CS:IP)	Yes

---

13. List the five flag bits tested by conditional jump instructions.

1. **Carry (CF)**
2. **Zero (ZF)**
3. **Sign (SF)**
4. **Overflow (OF)**
5. **Parity (PF)**

---

14. Describe how the JA instruction operates.

JA → **Jump if Above**
Used for **unsigned comparisons**.
It jumps if:

CF = 0 and ZF = 0

(i.e., first operand > second operand in unsigned comparison).

---

15. When will the JO instruction jump?

JO → **Jump if Overflow**
Jumps if **Overflow Flag (OF) = 1**.

---

16. Which conditional jump instructions follow the comparison of signed numbers?

- JG / JNLE (Jump if Greater)
- JL / JNGE (Jump if Less)
- JGE / JNL (Jump if Greater or Equal)

- JLE / JNG (Jump if Less or Equal)
- JO, JNO, JS, JNS

---

17. Which conditional jump instructions follow the comparison of unsigned numbers?

- JA / JNBE (Jump if Above)
- JB / JNAE (Jump if Below)
- JAE / JNB (Jump if Above or Equal)
- JBE / JNA (Jump if Below or Equal)
- JC, JNC

---

18. Which conditional jump instructions test both the Z and C flag bits?

- JBE (Jump if Below or Equal) → tests CF and ZF
- JAE / JNB (Jump if Above or Equal) → tests CF
- JA / JNBE (Jump if Above) → tests CF and ZF

---

19. When does the JCXZ instruction jump?

JCXZ → **Jump if CX = 0** (in 16-bit mode).
Used for loops; checks **CX register** before jumping.

---

20. Which SET instruction is used to set AL if the flag bits indicate a zero condition?

SETZ AL → Sets AL = 1 if **Zero Flag = 1**, else AL = 0.

---

21. The 8086 LOOP instruction decrements register ____________ and tests it for a 0.

➞ CX

---

22. The Pentium 4 LOOPD instruction decrements register ____________.

➞ ECX

---

23. The Core2 operated in 64-bit mode for a LOOP instruction decrements register _____________.

➞ RCX

---

24. Short sequence to store 00H into 150H bytes beginning at DATAZ using LOOP:

```
MOV CX, 150H        ; set counter
MOV DI, OFFSET DATAZ ; point to start of DATAZ
MOV AL, 00H
NEXT: MOV [DI], AL
INC DI
LOOP NEXT
```

✅ Stores 00H into 150H bytes starting at DATAZ.

---

25. Explain how the LOOPE instruction operates.

LOOPE (Loop while Equal) →

- Decrements CX
- Jumps to label **if CX ≠ 0 and ZF = 1**

Used to repeat a block while **equal condition remains true**.

---

26. Show the assembly language generated by:

```
.IF AL == 3
 ADD AL, 2
.ENDIF
```

Assembler expands this to:

```
CMP AL, 3
JNE SKIP
ADD AL, 2
SKIP:
```

27. Counting numbers above/below 42H in 100H-byte block

```
MOV SI, OFFSET BLOCK
MOV CX, 100H
MOV BL, 42H
MOV BYTE PTR [UP], 0
MOV BYTE PTR [DOWN], 0

NEXT: MOV AL, [SI]
CMP AL, BL
JA ABOVE
JB BELOW
JMP SKIP

ABOVE: INC BYTE PTR [UP]
JMP SKIP
BELOW: INC BYTE PTR [DOWN]
SKIP: INC SI
LOOP NEXT
```

✅ Counts bytes >42H and <42H, storing results in UP and DOWN.

---

28. Copy BLOCKA → BLOCKB until 00H (REPEAT–UNTIL)

```
MOV SI, OFFSET BLOCKA
MOV DI, OFFSET BLOCKB
REPEAT:
  MOV AL, [SI]
  MOV [DI], AL
  INC SI
  INC DI
UNTIL AL == 00H
```

Assembler expands this to:

```
MOV SI, OFFSET BLOCKA
MOV DI, OFFSET BLOCKB
NEXT: MOV AL, [SI]
MOV [DI], AL
INC SI
INC DI
CMP AL, 00H
JNE NEXT
```

---

29. What happens if .WHILE 1 is placed in a program?

It creates an **infinite loop** because the condition is **always true**.

---

30. Add BLOCKA to BLOCKB while sum ≠ 12H

```
MOV SI, OFFSET BLOCKA
MOV DI, OFFSET BLOCKB
WHILE SUM != 12H
  MOV AL, [SI]
  ADD AL, [DI]
```

```
 MOV [DI], AL
 CMP AL, 12H
 INC SI
 INC DI
ENDW
```

Assembler expands into a CMP/JNE loop until the sum becomes 12H.

---

31. What is the purpose of the .BREAK directive?

.BREAK **terminates a WHILE or REPEAT loop** immediately, similar to break in high-level languages.

---

32. What is a procedure?

A **procedure** is a **block of code** that performs a specific task and can be called from other parts of the program to **reuse code**.

---

33. Explain near and far CALL instructions.

- **Near CALL**:
  Saves only **IP** (instruction pointer) on the stack and jumps **within the same code segment**.
- **Far CALL**:
  Saves both **CS and IP** and transfers control to a **different segment**.

---

34. The last executable instruction in a procedure must be a(n) ____________.

➞ **RET (Return)** instruction.

---

35. How does the near RET instruction function?

Pops **IP** from the stack and continues execution at that address in the **same segment**.

---

36. How is a procedure identified as near or far?

By its **declaration**:

PROC NEAR

PROC FAR

---

37. Which directive identifies the start of a procedure?

➞ **PROC** directive.
Example:
MYPROC PROC NEAR

---

38. Write a near procedure that cubes CX (only modifies CX).

```
CUBE PROC NEAR

   PUSH AX

   MOV AX, CX

   IMUL CX

   IMUL CX

   MOV CX, AX

   POP AX

   RET

CUBE ENDP
```

---

39. Explain what RET 6 accomplishes.

- Pops the **return address** from the stack.
- Then **adds 6** to **SP** (discarding 6 bytes of parameters passed to the procedure).

---

40. Procedure: Multiply DI × SI, divide by 100H, result in AX

```
MUL_DIV PROC NEAR
   PUSH DX
   MOV AX, DI
   MUL SI
   MOV BX, 100H
   DIV BX
   POP DX
   RET
MUL_DIV ENDP
```

---

41. Procedure: Sum EAX, EBX, ECX, EDX → EAX; set EDI = 1 if carry

```
SUM_PROC PROC NEAR
   XOR EDI, EDI
   ADD EAX, EBX
   ADC EAX, ECX
   ADC EAX, EDX
   JC CARRY
   JMP DONE
CARRY: MOV EDI, 1
DONE:  RET
SUM_PROC ENDP
```

---

42. What is an interrupt?

An **interrupt** is a **signal that temporarily halts CPU execution**, saving its state and jumping to a predefined **interrupt service routine (ISR)**.

43. Which software instructions call an interrupt service procedure?

➞ **INT n** instructions.
Example: INT 21H

44. How many different interrupt types are available in the microprocessor?

➞ **256 interrupt types** (00H–FFH).

45. Interrupt vector contents and purpose

Content	Description
IP (2 bytes)	Offset address of ISR
CS (2 bytes)	Segment address of ISR

Stored at address = 4 × Type Number

46. Purpose of interrupt vector type number 0

➞ **Divide-by-zero error interrupt**.

47. How does IRET differ from RET?

- **RET**: Pops only **IP (and CS for far)**.
- **IRET**: Pops **IP, CS, and Flags** — restoring CPU state after ISR.

48. What is the IRETD instruction?

Used in **32-bit mode**, pops **EIP, CS, and EFLAGS** from the stack.

49. What is the IRETQ instruction?

Used in **64-bit mode**, pops **RIP, CS, and RFLAGS** from the stack.

50. The INTO instruction interrupts only for what condition?

➞ When the **Overflow Flag (OF) = 1**.

51. Interrupt vector for INT 40H stored at which memory?

Address = 40H × 4 = 0100H
So stored at:

000100H–000103H

52. Instructions controlling INTR pin

- STI → Set Interrupt Flag (enable INTR)
- CLI → Clear Interrupt Flag (disable INTR)

53. Instruction testing the BUSY pin

➞ WAIT instruction (pauses until BUSY pin = 0).

54. When will the BOUND instruction interrupt a program?

If the **array index** lies **outside the specified bounds**.

55. ENTER 16,0 creates a stack frame that contains ____________ bytes.

➞ **16 bytes**

56. Which register moves to the stack when ENTER executes?

➞ **BP (Base Pointer)** register (used to set up stack frame).

57. Which instruction passes opcodes to the numeric coprocessor?

➞ **ESC (Escape)** instruction.