**Notes Sheet**

**Table of Contents**

**Chapter 1** – Complex Numbers and Magnitude  
**Chapter 2** – Matrix Multiplication of Pauli Operators and Finding the Adjoint  
**Chapter 3** – Matrix Powers and Identity Relationships  
**Chapter 4** – Composite Matrix Operations with Pauli Matrices  
**Chapter 5** – Tensor Product of Pauli Operators  
**Chapter 6** – Pauli Operator Acting on a Single Qubit  
**Chapter 7** – Tensor Product State Representation  
**Chapter 8** – Exercise 3.1.21 — Bloch Sphere & Pauli Operator Eigenvectors  
**Chapter 9** – Exercise 3.3.6 — Proving CNOT Is Hermitian and Unitary  
**Chapter 10** – Exercise 3.3.9 — Multi-Qubit Circuit Composition  
**Chapter 11** – Exercise 5.1.5 — Phase Kickback & Controlled-V Operator  
**Summary Sheet** – Quick Reference: Formulas, Steps, and Recognition Cues

**Chapter 1 — Complex Numbers and Magnitude**

**Problem Recognition**

When a problem asks you to *simplify an expression with complex numbers* or *find its magnitude*, look for:

* The imaginary unit i, where i² = -1
* Expressions like (a + bi) or (c + di)
* Operations involving multiplication, conjugation, or absolute value symbols like |z|

**Why This Method Applies**

Complex numbers combine *real* and *imaginary* parts.  
When we multiply or take magnitude, we use algebra plus the rule i² = -1.

**Step-by-Step Method**

**Step 1 – Identify the real and imaginary parts**

z = a + bi

where a is the real part and b is the imaginary part.

**Step 2 – To find the conjugate**

z\* = a - bi

**Step 3 – Multiply two complex numbers**

(a + bi)(c + di)

= (ac - bd) + (ad + bc)i

**Step 4 – To find magnitude**

|z| = sqrt(a² + b²)

**Example 1**

Simplify and find the magnitude of

(2 + 3i)(1 - 4i)

**Solution:**

= (2)(1) - (3)(-4) + [(2)(-4) + (3)(1)]i

= (2 + 12) + (-8 + 3)i

= 14 - 5i

**Magnitude:**

|z| = sqrt(14² + (-5)²)

|z| = sqrt(196 + 25)

|z| = sqrt(221)

**Example 2**

Find the conjugate and magnitude of z = 4 - 3i.

z\* = 4 + 3i

|z| = sqrt(4² + (-3)²)

= sqrt(16 + 9)

= 5

**Example 3**

Multiply (1 + 2i) and (3 + 4i) and express the result in standard form.

(1 + 2i)(3 + 4i)

= (1)(3) - (2)(4) + [(1)(4) + (2)(3)]i

= (3 - 8) + (4 + 6)i

= -5 + 10i

**Formulas and Symbol Meanings**

z = a + bi → complex number

z\* = a - bi → conjugate

|z| = sqrt(a² + b²) → magnitude (length)

i² = -1 → defining property of imaginary unit

**Key Takeaways**

* Always use i² = -1 to simplify powers of i.
* Magnitude is always **non-negative**.
* The conjugate flips the sign of the imaginary part.

**Chapter 2 — Matrix Multiplication of Pauli Operators and Finding the Adjoint**

**Problem Recognition**

Look for matrices labeled as:

X, Y, Z

and for questions that ask to:

* Multiply them together (XY, XZ, etc.)
* Find if they’re *unitary* or *Hermitian*
* Compute the *adjoint* (conjugate transpose)

**Why This Method Applies**

Pauli matrices are fundamental 2×2 matrices used to describe qubit transformations.  
Their special properties (Hermitian and unitary) make them reusable in quantum gates.

**Pauli Matrices**

X = [ 0 1 ]

[ 1 0 ]

Y = [ 0 -i ]

[ i 0 ]

Z = [ 1 0 ]

[ 0 -1 ]

**Step-by-Step Method**

**Step 1 – To multiply matrices**  
Multiply row-by-column:

(XY)₍ij₎ = Σₖ X₍ik₎ \* Y₍kj₎

**Step 2 – To find the adjoint**  
Take the **conjugate transpose**:

A\* = (Ā)ᵀ

where each complex number is conjugated (change i → -i).

**Step 3 – To check if unitary**

A\* A = I

**Step 4 – To check if Hermitian**

A\* = A

**Example 1**

Find XY and check if it is Hermitian.

X = [0 1]

[1 0]

Y = [0 -i]

[i 0]

XY = [ (0)(0)+(1)(i) (0)(-i)+(1)(0) ]

[ (1)(0)+(0)(i) (1)(-i)+(0)(0) ]

XY = [ i 0 ]

[ 0 -i ]

Compute conjugate transpose:

(XY)\* = [ -i 0 ]

[ 0 i ]

Since (XY)\* ≠ XY → not Hermitian.

**Example 2**

Find ZX.

Z = [1 0]

[0 -1]

X = [0 1]

[1 0]

ZX = [ (1)(0)+(0)(1) (1)(1)+(0)(0) ]

[ (0)(0)+(-1)(1) (0)(1)+(-1)(0) ]

ZX = [ 0 1 ]

[ -1 0 ]

**Example 3**

Find the adjoint of Y.

Y = [ 0 -i ]

[ i 0 ]

Conjugate all i’s:

Ȳ = [ 0 i ]

[ -i 0 ]

Transpose:

Y\* = [ 0 -i ]

[ i 0 ]

So Y\* = Y → Hermitian.

**Formulas and Symbol Meanings**

A\* = conjugate transpose

I = identity matrix

X² = Y² = Z² = I

**Key Takeaways**

* Every Pauli matrix is **Hermitian and unitary**.
* To find the adjoint: **flip across diagonal + conjugate**.
* Multiplying two Pauli matrices usually produces **i × (the third Pauli)**.

Absolutely — continuing directly from Chapter 3 onward.  
Same formatting, same style.

**Chapter 3 — Matrix Powers and Identity Relationships**

**Problem Recognition**

You’ll know you’re in this category if the problem asks:

* “Compute X², Y², or Z²”
* “Show that a Pauli matrix squared equals the identity”
* “Prove this operator is unitary”

**Why This Method Applies**

Squaring a matrix tests whether it represents a reflection, rotation, or identity transformation in qubit space.  
For Pauli matrices, applying the same operator twice returns the qubit to its original state, forming the **identity**.

**Step-by-Step Method**

**Step 1 – Write the matrix**  
Use the known Pauli matrix forms:

X = [ 0 1 ]

[ 1 0 ]

Y = [ 0 -i ]

[ i 0 ]

Z = [ 1 0 ]

[ 0 -1 ]

**Step 2 – Multiply the matrix by itself**

X² = X \* X

**Step 3 – Simplify each entry (row by column)**  
Apply normal matrix multiplication rules.

**Step 4 – Check if result = I**

If

A² = I

then the matrix is **unitary and involutory**.

**Example 1**

Compute ( X^2 ).

X = [0 1]

[1 0]

X² = [ (0)(0)+(1)(1) (0)(1)+(1)(0) ]

[ (1)(0)+(0)(1) (1)(1)+(0)(0) ]

X² = [ 1 0 ]

[ 0 1 ] = I

**Example 2**

Compute ( Y^2 ).

Y = [0 -i]

[i 0]

Y² = [ (0)(0)+(-i)(i) (0)(-i)+(-i)(0) ]

[ (i)(0)+(0)(i) (i)(-i)+(0)(0) ]

Y² = [ (-i\*i) 0 ]

[ 0 (-i\*i) ]

Since ( i\*i = -1 ):

Y² = [1 0]

[0 1] = I

**Example 3**

Compute ( Z^2 ).

Z = [1 0]

[0 -1]

Z² = [ (1)(1)+(0)(0) (1)(0)+(0)(-1) ]

[ (0)(1)+(-1)(0) (0)(0)+(-1)(-1) ]

Z² = [1 0]

[0 1] = I

**Formulas and Symbol Meanings**

A² = I → means A is its own inverse

Unitary → A\* A = I

Hermitian → A\* = A

**Key Takeaways**

* For Pauli matrices, squaring always returns the identity.
* ( X² = Y² = Z² = I ).
* This proves each Pauli is a reflection operator on the Bloch sphere.

**Chapter 4 — Composite Matrix Operations with Pauli Matrices**

**Problem Recognition**

You’ll see problems that ask to:

* Multiply two or more Pauli matrices in sequence (XZ, YX, ZY)
* Determine commutation or anticommutation
* Identify resulting phase factors (±i)

**Why This Method Applies**

Pauli matrices follow **non-commutative multiplication**, meaning the order of multiplication matters.  
Their relationships are cyclic:

X Y = iZ

Y Z = iX

Z X = iY

and reversing order flips the sign:

Y X = -iZ

Z Y = -iX

X Z = -iY

**Step-by-Step Method**

**Step 1 – Recall Pauli multiplication rules**  
Use the cyclic pattern or derive directly by matrix multiplication.

**Step 2 – Write the result as i times the third matrix**

XY = iZ

**Step 3 – If order reversed, add negative**

YX = -iZ

**Step 4 – Verify using actual multiplication if uncertain**

**Example 1**

Find ( XY ).

XY = iZ

To verify:

X = [0 1]

[1 0]

Y = [0 -i]

[i 0]

XY = [ i 0 ]

[ 0 -i ] = iZ

**Example 2**

Find ( ZX ).

Z = [1 0]

[0 -1]

X = [0 1]

[1 0]

ZX = [ 0 1 ]

[ -1 0 ]

Compare with ( iY = [0 1; -1 0] ).  
✅ Therefore ( ZX = iY ).

**Example 3**

Find ( YZ ).

Y = [0 -i]

[i 0]

Z = [1 0]

[0 -1]

YZ = [0 i]

[i 0] = iX

**Formulas**

XY = iZ

YZ = iX

ZX = iY

YX = -iZ

ZY = -iX

XZ = -iY

**Key Takeaways**

* Order matters for Pauli matrix products.
* The pattern cycles through X → Y → Z.
* Reversing order changes sign (anticommutation).

**Chapter 5 — Tensor Product of Pauli Operators**

**Problem Recognition**

Look for ⊗ or problems mentioning:

* “2-qubit space”
* “Compute X ⊗ I”, “Y ⊗ Z”, etc.
* “Represent the combined operator for two qubits”

**Why This Method Applies**

Tensor products build multi-qubit systems.  
If one qubit uses operator X and the other uses Z, then the system operator is X ⊗ Z.

**Step-by-Step Method**

**Step 1 – Recall definition of tensor product**

For

A = [a b]

[c d]

B = [w x]

[y z]

Then

A ⊗ B = [ aB bB ]

[ cB dB ]

Each element of A multiplies the entire matrix B.

**Step 2 – Compute for example X ⊗ Z**

X = [0 1]

[1 0]

Z = [1 0]

[0 -1]

X ⊗ Z = [ (0)Z (1)Z ]

[ (1)Z (0)Z ]

= [ 0 0 1 0

0 0 0 -1

1 0 0 0

0 -1 0 0 ]

**Example 1**

Compute ( X ⊗ I ).

I = [1 0]

[0 1]

X ⊗ I = [ 0 0 1 0

0 0 0 1

1 0 0 0

0 1 0 0 ]

**Example 2**

Compute ( Z ⊗ X ).

Z = [1 0]

[0 -1]

X = [0 1]

[1 0]

Z ⊗ X = [ (1)X (0)X

(0)X (-1)X ]

= [ 0 1 0 0

1 0 0 0

0 0 0 -1

0 0 -1 0 ]

**Example 3**

Compute ( Y ⊗ Y ).

Y = [0 -i]

[i 0]

Y ⊗ Y = [ (0)Y (-i)Y

(i)Y (0)Y ]

After simplification, ( Y ⊗ Y = diag(-1, 1, 1, -1) ).

**Formulas**

A ⊗ B = expand each element of A times full B

dim(A ⊗ B) = (dim A × dim B)

**Key Takeaways**

* Tensor product combines systems: 2 qubits → 4D space.
* Operator for both qubits is **A ⊗ B**.
* Order matters: first qubit first operator.

Excellent — continuing with Chapters **6–8** in the same clean textbook format.

**Chapter 6 — Pauli Operator Acting on a Single Qubit**

**Problem Recognition**

Look for questions like:

* “Find the result of applying X to |0⟩”
* “Compute Z|1⟩”
* “Describe the new state after a gate acts on a qubit”

These questions involve **operator–state multiplication**, where a **matrix** acts on a **column vector**.

**Why This Method Applies**

In quantum computing, every qubit is a 2D vector and every gate is a 2×2 matrix.  
Applying a gate means multiplying that matrix by the vector representing the qubit.

**Step-by-Step Method**

**Step 1 – Write the computational basis states**

|0⟩ = [1]

[0]

|1⟩ = [0]

[1]

**Step 2 – Write the operator matrix**  
Use any of:

X = [0 1]

[1 0]

Y = [0 -i]

[i 0]

Z = [1 0]

[0 -1]

**Step 3 – Multiply the matrix by the qubit vector**  
Normal matrix multiplication (row × column).

**Example 1**

Apply **X** to |0⟩.

X|0⟩ = [0 1] [1]

[1 0] [0]

= [0]

[1]

= |1⟩

So X flips the qubit.

**Example 2**

Apply **Z** to |1⟩.

Z|1⟩ = [1 0] [0]

[0 -1][1]

= [0]

[-1]

= -|1⟩

Adds a phase of −1 to |1⟩.

**Example 3**

Apply **Y** to |0⟩.

Y|0⟩ = [0 -i][1]

[i 0][0]

= [0]

[i]

= i|1⟩

**ASCII Gate Diagram**

|0⟩ → X → |1⟩

|1⟩ → X → |0⟩

|0⟩ → Z → |0⟩

|1⟩ → Z → -|1⟩

|0⟩ → Y → i|1⟩

|1⟩ → Y → -i|0⟩

**Formulas**

X|0⟩ = |1⟩

X|1⟩ = |0⟩

Z|0⟩ = |0⟩

Z|1⟩ = -|1⟩

Y|0⟩ = i|1⟩

Y|1⟩ = -i|0⟩

**Key Takeaways**

* X flips the qubit (like a NOT gate).
* Z flips the **phase** of |1⟩.
* Y does both: flip + phase.

**Chapter 7 — Tensor Product State Representation**

**Problem Recognition**

Look for:

* Combined states like |0⟩|1⟩ or |01⟩
* Phrases such as “two-qubit system” or “tensor product of states”
* Operations like |ψ⟩ = |0⟩ ⊗ |1⟩

**Why This Method Applies**

Tensor products describe **multi-qubit states**.  
If each qubit is 2D, two qubits combine into a 4D vector.

**Step-by-Step Method**

**Step 1 – Write basis states**

|0⟩ = [1 0]ᵀ

|1⟩ = [0 1]ᵀ

**Step 2 – Use tensor product definition**  
For two vectors:

|a⟩ ⊗ |b⟩ =

[ a₁b₁ ]

[ a₁b₂ ]

[ a₂b₁ ]

[ a₂b₂ ]

**Step 3 – Substitute and compute**

**Example 1**

Find |0⟩ ⊗ |1⟩.

|0⟩ = [1 0]

|1⟩ = [0 1]

|0⟩⊗|1⟩ = [1×0, 1×1, 0×0, 0×1]ᵀ

= [0, 1, 0, 0]ᵀ

So |0⟩|1⟩ = [0,1,0,0]ᵀ.

**Example 2**

Find |1⟩ ⊗ |0⟩.

|1⟩ = [0 1]

|0⟩ = [1 0]

|1⟩⊗|0⟩ = [0×1, 0×0, 1×1, 1×0]ᵀ

= [0,0,1,0]ᵀ

**Example 3**

Find |ψ⟩ = (|0⟩ + |1⟩)/√2 ⊗ |1⟩.

(|0⟩ + |1⟩)/√2 = (1/√2)[1 1]ᵀ

|1⟩ = [0 1]ᵀ

|ψ⟩ = (1/√2)[|0⟩⊗|1⟩ + |1⟩⊗|1⟩]

= (1/√2)([0,1,0,0] + [0,0,0,1])

= (1/√2)[0,1,0,1]

**Formulas**

|00⟩ = |0⟩⊗|0⟩ = [1 0 0 0]ᵀ

|01⟩ = |0⟩⊗|1⟩ = [0 1 0 0]ᵀ

|10⟩ = |1⟩⊗|0⟩ = [0 0 1 0]ᵀ

|11⟩ = |1⟩⊗|1⟩ = [0 0 0 1]ᵀ

**Key Takeaways**

* Tensor product combines qubits into larger systems.
* Basis states expand to powers of 2 in size.
* Order of qubits matters: first qubit leftmost.

**Chapter 8 — Exercise 3.1.21: Bloch Sphere & Pauli Operator Eigenvectors**

**Problem Recognition**

This type asks for:

* Eigenstates of X, Y, Z
* Relationships on the Bloch sphere
* States like |X+⟩, |X−⟩, etc.

**Why This Method Applies**

Every Pauli matrix has two eigenvectors that correspond to opposite points on the **Bloch sphere**.  
These represent directions ± along x̂, ŷ, or ẑ.

**Step-by-Step Method**

**Step 1 – Recall eigenvalue definition**

A|ψ⟩ = λ|ψ⟩

where λ is the eigenvalue and |ψ⟩ is the eigenvector.

**Step 2 – Solve for |ψ⟩**  
Find non-zero vectors satisfying (A - λI)|ψ⟩ = 0.

**Step 3 – Normalize the vector**  
So that ⟨ψ|ψ⟩ = 1.

**Example 1**

Find eigenstates of **X**.

X = [0 1]

[1 0]

Solve:

X|ψ⟩ = λ|ψ⟩

→ [0 1; 1 0][a b]ᵀ = λ[a b]ᵀ

→ [b a]ᵀ = λ[a b]ᵀ

So:

b = λa

a = λb

From first equation: b = λa → a = λb = λ²a → λ² = 1 → λ = ±1.

For λ = +1:

b = a → |ψ₁⟩ = (1/√2)[1 1]ᵀ = |X+⟩

For λ = −1:

b = -a → |ψ₂⟩ = (1/√2)[1 -1]ᵀ = |X−⟩

**Example 2**

Find eigenstates of **Z**.

Z = [1 0]

[0 -1]

Immediate:

Z|0⟩ = |0⟩ → λ=+1

Z|1⟩ = -|1⟩ → λ=-1

So eigenstates are |0⟩ and |1⟩.

**Example 3**

Find eigenstates of **Y**.

Y = [0 -i]

[i 0]

Solve:

[0 -i; i 0][a b]ᵀ = λ[a b]ᵀ

→ [-ib, ia]ᵀ = λ[a b]ᵀ

Gives:

-ib = λa → b = iλa

Try λ=+1:

b = i(1)a = ia → |ψ₁⟩ = (1/√2)[1 i]ᵀ = |Y+⟩

Try λ=−1:

b = i(-1)a = -ia → |ψ₂⟩ = (1/√2)[1 -i]ᵀ = |Y−⟩

**ASCII Bloch Diagram (simplified)**

|0⟩ (north)

|

|Y−⟩ -x | |Y+⟩ +x

|

|1⟩ (south)

**Formulas**

|X±⟩ = (1/√2)(|0⟩ ± |1⟩)

|Y±⟩ = (1/√2)(|0⟩ ± i|1⟩)

|Z±⟩ = |0⟩, |1⟩

**Key Takeaways**

* Each Pauli matrix has two eigenstates with eigenvalues ±1.
* These eigenstates form the six poles of the Bloch sphere.
* Eigenvectors show how gates rotate or reflect qubit states.

Continuing with **Chapters 9–11** — same structured, textbook format with clear procedural explanations, stacked equations, and ASCII-style clarity.

**Chapter 9 — Exercise 3.3.6: Proving CNOT Is Hermitian and Unitary**

**Problem Recognition**

You’ll know it’s this type of problem when you see:

* The **Controlled-NOT (CNOT)** operator or matrix
* Questions asking if it’s *Hermitian* or *unitary*
* Statements like “show that CNOT = CNOT\*” or “CNOT² = I”

**Why This Method Applies**

The CNOT gate flips the **target** qubit if and only if the **control** qubit is 1.  
Like Pauli matrices, it’s both Hermitian (its conjugate transpose equals itself) and unitary (its inverse equals its conjugate transpose).

**CNOT Definition**

The **CNOT** acts on two qubits:

|x⟩|y⟩ → |x⟩|y ⊕ x⟩

where ⊕ means XOR (addition modulo 2).

Matrix form (computational basis |00⟩, |01⟩, |10⟩, |11⟩):

CNOT = [1 0 0 0

0 1 0 0

0 0 0 1

0 0 1 0]

**Step-by-Step Method**

**Step 1 – Check Hermitian**

Take the conjugate transpose:

CNOT\* = (CNOT̄)ᵀ

Since CNOT has only 0s and 1s, it’s real, so:

CNOT\* = CNOTᵀ

Transpose:

CNOTᵀ = [1 0 0 0

0 1 0 0

0 0 0 1

0 0 1 0]

✅ Same as original → **Hermitian**.

**Step 2 – Check Unitary**

Multiply by its adjoint:

CNOT\* CNOT = I

Perform the multiplication:

CNOT² =

[1 0 0 0

0 1 0 0

0 0 0 1

0 0 1 0]

\*

[1 0 0 0

0 1 0 0

0 0 0 1

0 0 1 0]

=

[1 0 0 0

0 1 0 0

0 0 1 0

0 0 0 1]

= I

✅ **Unitary.**

**Example 1**

Prove CNOT² = I directly.

Using the XOR rule:

|x⟩|y⟩ → |x⟩|y ⊕ x⟩ → |x⟩|(y ⊕ x) ⊕ x⟩ = |x⟩|y⟩

→ Applying twice restores original state.

**Example 2**

Check Hermiticity numerically.

Because all entries are real, conjugation doesn’t change them.  
Transpose doesn’t change the structure either → CNOT = CNOT\*.

**Example 3**

Show CNOT† = CNOT⁻¹.

Since CNOT² = I:

CNOT⁻¹ = CNOT

and CNOT† = CNOT

→ CNOT is its own inverse.

**ASCII Gate Diagram**

Control: ─■────

Target: ─⊕────

Effect:

|00⟩ → |00⟩

|01⟩ → |01⟩

|10⟩ → |11⟩

|11⟩ → |10⟩

**Formulas**

Hermitian → U\* = U

Unitary → U\*U = I

CNOT² = I

**Key Takeaways**

* CNOT is both **Hermitian and Unitary**.
* It is **its own inverse**.
* Real matrices with 0s and 1s are automatically self-conjugate.

**Chapter 10 — Exercise 3.3.9: Multi-Qubit Circuit Composition**

**Problem Recognition**

This problem involves:

* Circuits with **multiple gates and qubits**
* States represented as |ψ₀⟩, |ψ₁⟩, etc.
* Sequential application of gates (e.g., H then CNOT)

**Why This Method Applies**

Quantum circuits are represented as **ordered matrix multiplications** on states.  
Each gate applies a linear transformation to the current state.

**Step-by-Step Method**

**Step 1 – Start with initial state**

|ψ₀⟩ = |00⟩

**Step 2 – Apply first gate (often Hadamard)**

H|0⟩ = (|0⟩ + |1⟩)/√2

So:

|ψ₁⟩ = (1/√2)(|00⟩ + |10⟩)

**Step 3 – Apply CNOT**  
CNOT flips target qubit if control is 1:

|00⟩ → |00⟩

|10⟩ → |11⟩

So:

|ψ₂⟩ = (1/√2)(|00⟩ + |11⟩)

This is a **Bell state** (entangled).

**Example 1**

Circuit: |0⟩ —H—■—  
        |0⟩ —⊕—

|ψ₀⟩ = |00⟩

H on first → (|0⟩+|1⟩)/√2 ⊗ |0⟩ = (1/√2)(|00⟩+|10⟩)

CNOT → (1/√2)(|00⟩+|11⟩)

**Example 2**

Circuit: |1⟩ —H—■—  
       |0⟩ —⊕—

|ψ₀⟩ = |10⟩

H|1⟩ = (|0⟩ - |1⟩)/√2

→ |ψ₁⟩ = (1/√2)(|00⟩ - |10⟩)

CNOT → (1/√2)(|00⟩ - |11⟩)

→ Second Bell state variant.

**Example 3**

If the first gate acts on the second qubit instead:

|ψ₀⟩ = |00⟩

(I⊗H)|00⟩ = (1/√2)(|00⟩+|01⟩)

Then CNOT(control=1st,target=2nd) does nothing (since control=0)

Final |ψ⟩ = (1/√2)(|00⟩+|01⟩)

**ASCII Circuit Diagram**

|0⟩──H──■──

|0⟩────⊕──

Output:

|ψ⟩ = (|00⟩ + |11⟩)/√2

**Formulas**

H = (1/√2)[[1,1],[1,-1]]

CNOT = [[1,0,0,0],[0,1,0,0],[0,0,0,1],[0,0,1,0]]

**Key Takeaways**

* Gate order matters: rightmost gate acts first.
* Entanglement arises naturally from H + CNOT.
* Use tensor products for multi-qubit states.

**Chapter 11 — Exercise 5.1.5: Phase Kickback & Controlled-V Operator**

**Problem Recognition**

This problem involves:

* Controlled-V or phase operators (C(V))
* Phase estimation or eigenstate relationships
* Using a control qubit to modify another operation’s phase

**Why This Method Applies**

“Phase kickback” occurs when a **controlled operation** transfers phase information from the target qubit back to the control qubit.  
This principle underlies algorithms like **Deutsch–Jozsa** and **Phase Estimation**.

**Step-by-Step Method**

**Step 1 – Recall eigenvalue form**

V|ψ⟩ = λ|ψ⟩, λ = e^{iθ}

**Step 2 – Controlled-V definition**

C(V) acts on |x⟩|ψ⟩ as:

|0⟩|ψ⟩ → |0⟩|ψ⟩

|1⟩|ψ⟩ → |1⟩V|ψ⟩

**Step 3 – Apply to eigenstate |ψ⟩**

C(V)|x⟩|ψ⟩ = |0⟩|ψ⟩ + e^{iθ}|1⟩|ψ⟩

→ The **phase e^{iθ}** moves onto the control qubit.

**Example 1**

If V|ψ⟩ = e^{iπ/2}|ψ⟩,  
then applying C(V) to (1/√2)(|0⟩+|1⟩)|ψ⟩ gives:

(1/√2)(|0⟩|ψ⟩ + e^{iπ/2}|1⟩|ψ⟩)

= (1/√2)(|0⟩ + i|1⟩)|ψ⟩

→ Control qubit picks up a phase = **kickback**.

**Example 2**

If the eigenvalue is e^{iθ}, the general rule:

(1/√2)(|0⟩+|1⟩)|ψ⟩ → (1/√2)(|0⟩ + e^{iθ}|1⟩)|ψ⟩

The control qubit now encodes θ as a phase difference.

**Example 3**

If θ = π:

(1/√2)(|0⟩+|1⟩)|ψ⟩ → (1/√2)(|0⟩ - |1⟩)|ψ⟩

→ Becomes |X−⟩ on the control qubit.

**ASCII Gate Diagram**

Control ─■────

│

Target ───V────

Result:

|ψ\_total⟩ = (1/√2)(|0⟩|ψ⟩ + e^{iθ}|1⟩|ψ⟩)

**Formulas**

C(V) = |0⟩⟨0| ⊗ I + |1⟩⟨1| ⊗ V

Phase kickback: eigenphase transfers to control qubit

**Key Takeaways**

* Controlled operators conditionally apply a gate.
* If the target is an eigenstate, the **control** gains the phase.
* This concept powers algorithms like **Phase Estimation**.

**Summary Sheet — Quick Reference**

**Complex Numbers**

(a + bi)(c + di) = (ac - bd) + (ad + bc)i

|a + bi| = sqrt(a² + b²)

i² = -1

**Pauli Matrices**

X = [0 1]

[1 0]

Y = [0 -i]

[i 0]

Z = [1 0]

[0 -1]

**Key Relationships**

X² = Y² = Z² = I

XY = iZ, YZ = iX, ZX = iY

YX = -iZ, ZY = -iX, XZ = -iY

**Tensor Products**

A⊗B = [aB bB; cB dB]

|00⟩=[1 0 0 0]ᵀ, |01⟩=[0 1 0 0]ᵀ, |10⟩=[0 0 1 0]ᵀ, |11⟩=[0 0 0 1]ᵀ

**Gate Actions**

X|0⟩=|1⟩, X|1⟩=|0⟩

Z|0⟩=|0⟩, Z|1⟩=-|1⟩

Y|0⟩=i|1⟩, Y|1⟩=-i|0⟩

**CNOT**

CNOT = [1 0 0 0

0 1 0 0

0 0 0 1

0 0 1 0]

CNOT² = I, Hermitian and Unitary

**Eigenstates**

|X±⟩ = (1/√2)(|0⟩ ± |1⟩)

|Y±⟩ = (1/√2)(|0⟩ ± i|1⟩)

|Z±⟩ = |0⟩, |1⟩

**Controlled Gates**

C(V) = |0⟩⟨0|⊗I + |1⟩⟨1|⊗V

Phase kickback: (1/√2)(|0⟩+|1⟩)|ψ⟩ → (1/√2)(|0⟩+e^{iθ}|1⟩)|ψ⟩

**Final Key Takeaways**

* Always write matrices and states explicitly before multiplying.
* The **adjoint** = conjugate transpose.
* **Unitary** means reversible.
* **Hermitian** means observable (real eigenvalues).
* Keep tensor order consistent.
* Phases can move between qubits — this is quantum interference.

✅ **End of Notes Sheet**