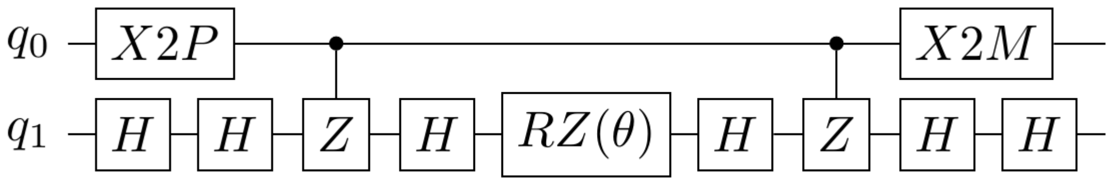
以单激发子 $\theta a_1^\dagger a_0 - \theta a_0^\dagger a_1$ 中部分线路为例。



```python
from typing import Any
import re

def get_gate_terms(string:str) -> list[str]:
    """Extract lines containing gate."""
    terms = string.split("\n")
    terms = [_.strip() for _ in terms if _.strip()]
    return terms

def get_gate_name(term:str) -> str:
    """Get the gate name from the term."""
    name = re.findall(r"^([\w]+)(?=\s)", term)[0]
    return name

def get_gate_qubits(term:str) -> list[int]:
    """Get the qubits acted upon by the gate from the term."""
    qubits = re.findall(r"(?<=Q)(\d+)", term)
    return [int(i) for i in qubits]

def get_gate_info(term:str) -> tuple[str, list[int], str]:
    """Get the gate information (name, params and qubits) according to the
term."""
    term = term.strip()
    gate_name = get_gate_name(term)
    qubits = get_gate_qubits(term)
    param = ""
    if gate_name.startswith("R"):
        param = term.split(" ")[-1]
    return gate_name, qubits, param

def get_n_qubits(gate_terms:list[str]):
    """Get the number of qubits used in circuit."""
    high_qubit = -1
    for gate_str in gate_terms:
        qubits = get_gate_qubits(gate_str)
        high_qubit = max(high_qubit, max(qubits))
    return high_qubit + 1

def adjacent_cancelable(gate0:str, gate1:str):
    """whether two adjacent gates can be canceled."""
    gate0_name, gate0_qubits, _ = get_gate_info(gate0)
    gate1_name, gate1_qubits, _ = get_gate_info(gate1)

    if gate0_name not in ["X", "Y", "Z", "H"]:
        return False
    if gate0_name != gate1_name:
        return False
    if gate0_qubits != gate1_qubits:
        return False
    return True

def simplify_circuit(circuit:str):
    """Buffering simplification."""
    gate_terms = get_gate_terms(circuit)
    circ_dict = dict(zip(range(len(gate_terms)), gate_terms))

    n_qubits = get_n_qubits(gate_terms)
    buffer0:dict[int, Any] = {i:0 for i in range(n_qubits)}
    buffer1:dict[int, Any] = {i:0 for i in range(n_qubits)}

    for i, gate_str in enumerate(gate_terms):
        qubits = get_gate_qubits(gate_str)
        related_qubits = set(qubits)
        for q in qubits:
            if not isinstance(buffer1[q], int):
                related_qubits.update(get_gate_qubits(buffer1[q][1]))
        for q in related_qubits:
            buffer0[q] = buffer1[q]

        for q in qubits:
            buffer1[q] = (i, gate_str)

        other_qubits = list(related_qubits.difference(set(qubits)))
        for q in other_qubits:
            buffer1[q] = 0

        for q in qubits:
            # Do nothing if the appointed elements are 0s in buffer0,
            # which implies no gates are there.
            if isinstance(buffer0[q], int):
                continue
            # Cancel them if two adjacent gates in buffer0 and buffer1
            # can be canceled simultaneously.
            if adjacent_cancelable(buffer0[q][1], buffer1[q][1]):
                if buffer0[q][0] in circ_dict:
                    del circ_dict[buffer0[q][0]]
                    buffer0[q] = 0
                if buffer1[q][0] in circ_dict:
                    del circ_dict[buffer1[q][0]]
                    buffer1[q] = 0
    return "\n".join(circ_dict.values())

string =  "X2P Q0\nH Q1\nH Q1\nCZ Q0 Q1\nH Q1\nRZ Q1 theta\nH Q1\nCZ Q0 Q1\nH
Q1\nH Q1\nX2M Q0"
print("before:")
print(string)

res = simplify_circuit(string)
print("\nafter:")
print(res)
```

```
before:
```

```
X2P Q0
H Q1
H Q1
CZ Q0 Q1
H Q1
RZ Q1 theta
H Q1
CZ Q0 Q1
H Q1
H Q1
X2M Q0

after:
X2P Q0
CZ Q0 Q1
H Q1
RZ Q1 theta
H Q1
CZ Q0 Q1
X2M Q0
```