Python Skeleton:

```python
import argparse

# Constants for the cache configuration
NUM_SETS = ...  # Configure number of sets
ASSOC = ...  # Configure cache associativity
BLOCK_SIZE = 64  # 64 bytes as block size

# Define the arrays for the cache simulation
tag_array = [[0 for _ in range(ASSOC)] for _ in range(NUM_SETS)]
lru_position = [[0 for _ in range(ASSOC)] for _ in range(NUM_SETS)]
dirty = [[False for _ in range(ASSOC)] for _ in range(NUM_SETS)]

# Variables to maintain the simulation statistics
Hit = 0
Miss = 0

def update_lru(address):
    # Logic for updating LRU policy
    pass

def update_fifo(address):
    # Logic for updating FIFO policy
    pass

def simulate_access(op, address):
    set_idx = (address // BLOCK_SIZE) % NUM_SETS
    tag = address // BLOCK_SIZE

    for i in range(ASSOC):
        if tag == tag_array[set_idx][i]:
            global Hit
            Hit += 1
            # Choose policy (LRU or FIFO) based on the configuration
            if ...:  # LRU policy is chosen
                update_lru(address)
            else:
                update_fifo(address)
        else:
            global Miss
            Miss += 1
            # Handle the miss scenario here

if __name__ == "__main__":
    # Setting up the argument parser
    parser = argparse.ArgumentParser(description="Cache Simulation using Trace Files")

    # Adding an argument to specify the trace file
    parser.add_argument("trace_file", type=str, help="Path to the trace file")

    # Parsing the command line arguments
    args = parser.parse_args()

    # Using the provided trace file path from the parsed arguments
    with open(args.trace_file, 'r') as file:
        for line in file:
            op, address = line.split()
            address = int(address, 16)
            simulate_access(op, address)

    # Print out the statistics
    print(f"Hits: {Hit}")
    print(f"Misses: {Miss}")
```