#Solution-1:

```python
#Solution-1:
def store(storage, element):
    new_storage = storage.copy()
    new_storage.update(element)
    return new_storage
 #Initialize memory
init_mem = {}
a = {800: 123}
b = {900: 1000}
# Store data in memory
mem = store(init_mem, a)  # mem = {800: 123}
mem = store(mem, b)  # mem = {800: 123, 900: 1000}

c = {800: 900}
mem = store(mem, c)  # mem = {800: 900, 900: 1000}

d = {1500: 700}
mem = store(mem, d)  # mem = {800: 900, 900: 1000, 1500: 700}

def idx_load_ac(storage, idx):
    """
    Load accumulator (ac) by indexed addressing.
    """
    return storage.get(idx) if idx != 700 else 700

index_register = 700
ac = idx_load_ac(mem, index_register)  # Now ac should correctly be 123

print(f"Memory: {mem}")
print(f"Accumulator (ac): {ac}")
```

```
Memory: {800: 900, 900: 1000, 1500: 700}
Accumulator (ac): 700
```

#Solution-2:

```python
#Solution-2:
def store(storage, elm):
    storage.update(elm)
    return storage
def dir_map_cache(cache, adr, storage):
    block_label = adr[4:8]  # Extract the 4-bit block label from the address
    tag = adr[:7]  # Extract the 7-bit tag from the address
    if block_label not in cache or cache[block_label][0] != tag:
        cache[block_label] = [tag, storage.get(adr, [0, 0, 0, 0, 0, 0, 0, 0]), 1]  # Update cache entry
    else:
        cache[block_label][2] = 1
    return cache[block_label][1]
init_mem = {}
mem = store(init_mem, {"00000110101000": [0, 1, 2, 3, 4, 5, 6, 7]})
print(mem)
cache = {}
adr1 = "00000110101000"
cache_result = dir_map_cache(cache, adr1, mem)
print(cache_result)
adr2 = "00001110101000"
cache_result = dir_map_cache(cache, adr2, mem)
print(cache_result)
adr3 = "00001110111111"
cache_result = dir_map_cache(cache, adr3, mem)
print(cache_result)
def check_cache(cache, adr):
    block_label = adr[4:8]
    if block_label in cache and cache[block_label][2] == 1:
        print("Hit")
    else:
        print("Miss")
check_cache(cache, adr1)
check_cache(cache, adr2)
check_cache(cache, adr3)
```

#Solution-3:

Individual Assignment: HW As | hw_5.ipynb - Colaboratory | Untitled document - Google D

colab.research.google.com/github/Swagaat/HW_5CE305/blob/main/hw_5.ipynb#scrollTo=kbxOjDAwUtAE

Relaunch to update

Big Picture Questi... | Statement of Purp... | Printable SAT Pra...

hw_5.ipynb

File Edit View Insert Runtime Tools Help Cannot save changes

Share

+ Code + Text Copy to Drive

Connect

```python
#Solution-3:
init_mem = {}
cache = {
    "blk0": ["00000000000", [0, 0, 0, 0, 0, 0, 0, 0], 0],
    "blk1": ["00000000000", [0, 0, 0, 0, 0, 0, 0, 0], 0],
    "blk2": ["00000000000", [0, 0, 0, 0, 0, 0, 0, 0], 0],
    "blk3": ["00000000000", [0, 0, 0, 0, 0, 0, 0, 0], 0],
}

def store(storage, elm):
    for key, value in elm.items():
        tag = key[:-3]
        for k, v in storage.items():
            if v[0] == tag and v[2] == 1:
                continue
            else:
                for k_cache, v_cache in cache.items():
                    if v_cache[2] == 0:
                        cache[k_cache] = [tag, value, 1]
                        break
                    else:
                        continue
    return storage

a = {"00000110101000": [0, 1, 2, 3, 4, 5, 6, 7]}
init_mem = store(init_mem, a)

b = {"00001110101000": [10, 11, 12, 13, 14, 15, 16, 17]}
init_mem = store(init_mem, b)

c = {"00011110101000": [20, 21, 22, 23, 24, 25, 26, 27]}
init_mem = store(init_mem, c)

d = {"00111110101000": [30, 31, 32, 33, 34, 35, 36, 37]}
init_mem = store(init_mem, d)
```

Individual Assignment: HW As | hw_5.ipynb - Colaboratory | Untitled document - Google D

colab.research.google.com/github/Swagaat/HW_5CE305/blob/main/hw_5.ipynb#scrollTo=kbxOjDAwUtAE

Relaunch to update

Big Picture Questi... | Statement of Purp... | Printable SAT Pra...

hw_5.ipynb

File Edit View Insert Runtime Tools Help Cannot save changes

Share

+ Code + Text Copy to Drive

Connect

```python
d = {"00111110101000": [30, 31, 32, 33, 34, 35, 36, 37]}
init_mem = store(init_mem, d)

e = {"01111110101000": [40, 41, 42, 43, 44, 45, 46, 47]}
init_mem = store(init_mem, e)

print("Memory:", init_mem)
print("Cache:", cache)

def imm_load_ac(val):
    return val

ac = imm_load_ac(800)
print(f"Accumulator after immediate load: {ac}")

def dir_load_ac(storage, val):
    block_address = val[:11]  # Extract the block address from the memory address
    if block_address in storage:
        return storage[block_address][int(val[11:], 2)]  # Return the specific value from the block
    else:
        print(f"Block not found for address {val}")
        return None

mem = {
    "00000110101000": [0, 1, 2, 3, 4, 5, 6, 7],
    "00001110101000": [10, 11, 12, 13, 14, 15, 16, 17],
}
ac = dir_load_ac(mem, "00000110101000")
print(f"Accumulator after direct load: {ac}")
```

```
Memory: {}
Cache: {'blk0': ['00000000000', [0, 0, 0, 0, 0, 0, 0, 0], 0], 'blk1': ['00000000000', [0, 0, 0, 0, 0, 0, 0, 0], 0], 'blk2': ['00000000000', [0, 0, 0, 0, 0,
Accumulator after immediate load: 800
Block not found for address 00000110101000
```