San Francisco Bay University

CE305 - Computer Organization
2023 Fall Homework #5

Due day: 12/8/2023

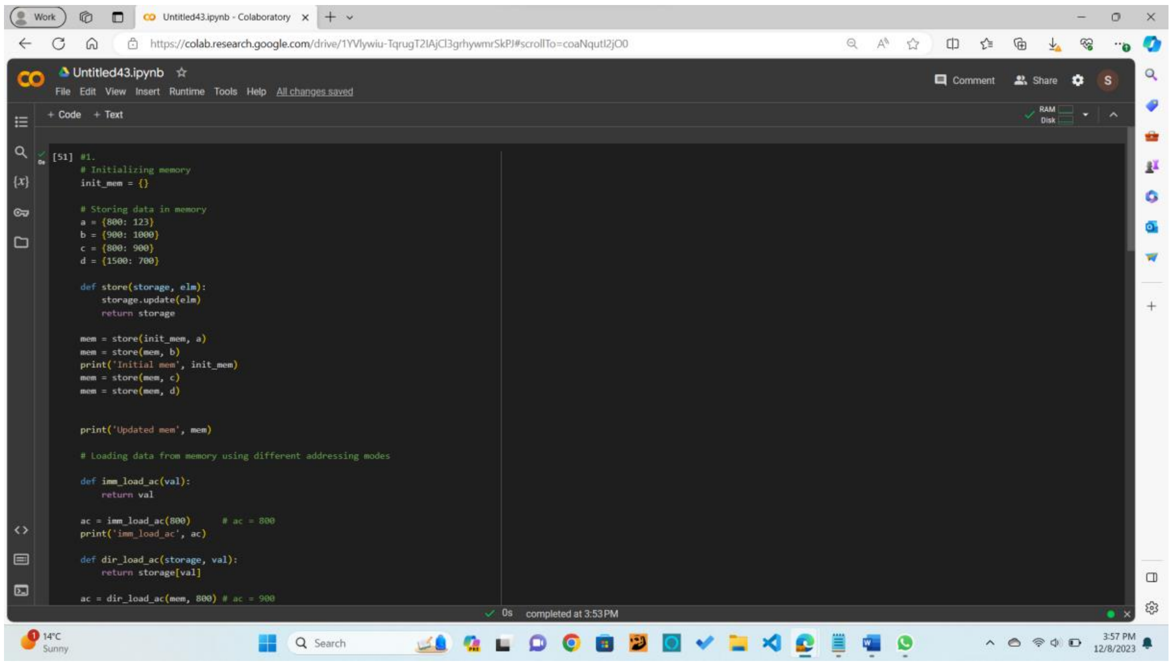
Instruction:

1. Homework answer sheet should contain the original questions and corresponding answers.
2. Answer sheet must be in PDF file format with Github links for the programming questions, but MS Word file can't be accepted. As follows is the answer sheet name format.
<course_id>_week<week_number>_StudentID_FirstName_LastName.pdf
3. The program name in Github must follow the format like *<course_id>_week<week_number>_q<question_number>_StudentID_FirstName_LastName*
4. Show screenshot of all running results, including the system date/time.
5. Only accept homework submission uploaded via Canvas.
6. Overdue homework submission can't be accepted.
3. Takes academic honesty and integrity seriously (Zero Tolerance of Cheating & Plagiarism)

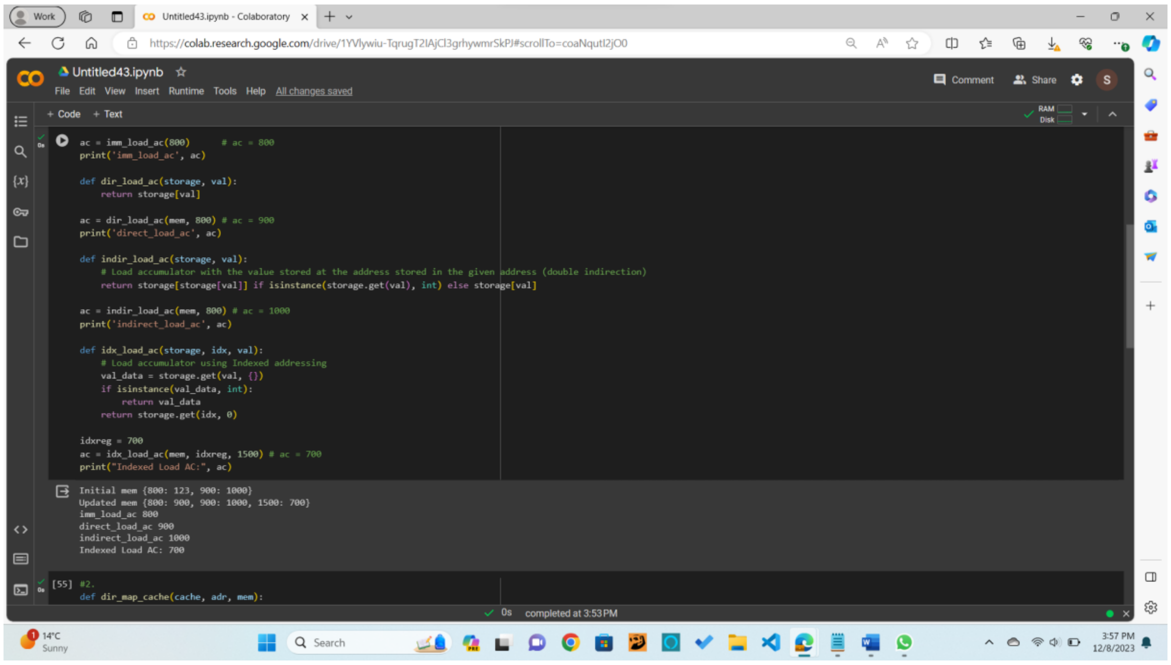
1. In MARIE architecture ISA, there are several different types of addressing modes, such as immediate addressing, direct addressing, indirect addressing and indexed addressing. Please complete the following functions in Python to simulate how the different addressing modes work.

```
init_mem={}                  # Empty memory at the very beginning
a={800: 123}                 # 1st data with address 800 and value 123
b={900: 1000}                # 2nd data with address 900 and value 1000

def store(storage,elm):      # Store an element to the memory
  storage.update(elm)
    return storage
print('Initial mem', init_mem)

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}
print('Updated mem', mem)

def imm_load_ac(val):      # Load accumulator(ac) by immediate addressing
      return val

ac = imm_load_ac(800)      # ac = 800
print('imm_load_ac', ac)

def dir_load_ac(storage, val):  # Load accumulator(ac) by direct addressing
  return storage[val]


ac = dir_load_ac(mem, 800)  # ac = 900
print('direct_load_ac', ac)

def indir_load_ac(storage, val):    # Load accumulator(ac) by indirect addressing
  return storage[storage[val]] if isinstance(storage.get(val), int) else
storage[val]



ac = indir_load_ac(mem, 800)   # ac = 1000
print('indirect_load_ac', ac)

def idx_load_ac(storage, idx, val):  # Load accumulator(ac) by Indexed addressing
   val_data = storage.get(val, {})
    if isinstance(val_data, int):
        return val_data
    return val_data.get(idx, 0)

idxreg=700
ac=idx_load_ac(mem, idxreg, 800)  # ac = 900
print("Indexed Load AC:", ac)
```
➔



```
[51] #1.
     # Initializing memory
     init_mem = {}

     # Storing data in memory
     a = {800: 123}
     b = {900: 1000}
     c = {800: 900}
     d = {1500: 700}

     def store(storage, elm):
         storage.update(elm)
         return storage

     mem = store(init_mem, a)
     mem = store(mem, b)
     print('Initial mem', init_mem)
     mem = store(mem, c)
     mem = store(mem, d)


     print('Updated mem', mem)

     # Loading data from memory using different addressing modes

     def imm_load_ac(val):
         return val

     ac = imm_load_ac(800)      # ac = 800
     print('imm_load_ac', ac)

     def dir_load_ac(storage, val):
         return storage[val]

     ac = dir_load_ac(mem, 800) # ac = 900
```

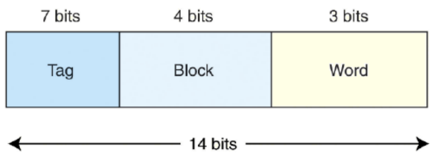


```
ac = imm_load_ac(800)      # ac = 800
print('imm_load_ac', ac)

def dir_load_ac(storage, val):
    return storage[val]

ac = dir_load_ac(mem, 800) # ac = 900
print('direct_load_ac', ac)

def indir_load_ac(storage, val):
    # Load accumulator with the value stored at the address stored in the given address (double indirection)
    return storage[storage[val]] if isinstance(storage.get(val), int) else storage[val]

ac = indir_load_ac(mem, 800) # ac = 1000
print('indirect_load_ac', ac)

def idx_load_ac(storage, idx, val):
    # Load accumulator using Indexed addressing
    val_data = storage.get(val, {})
    if isinstance(val_data, int):
        return val_data
    return storage.get(idx, 0)

idxreg = 700
ac = idx_load_ac(mem, idxreg, 1500) # ac = 700
print("Indexed Load AC:", ac)
```

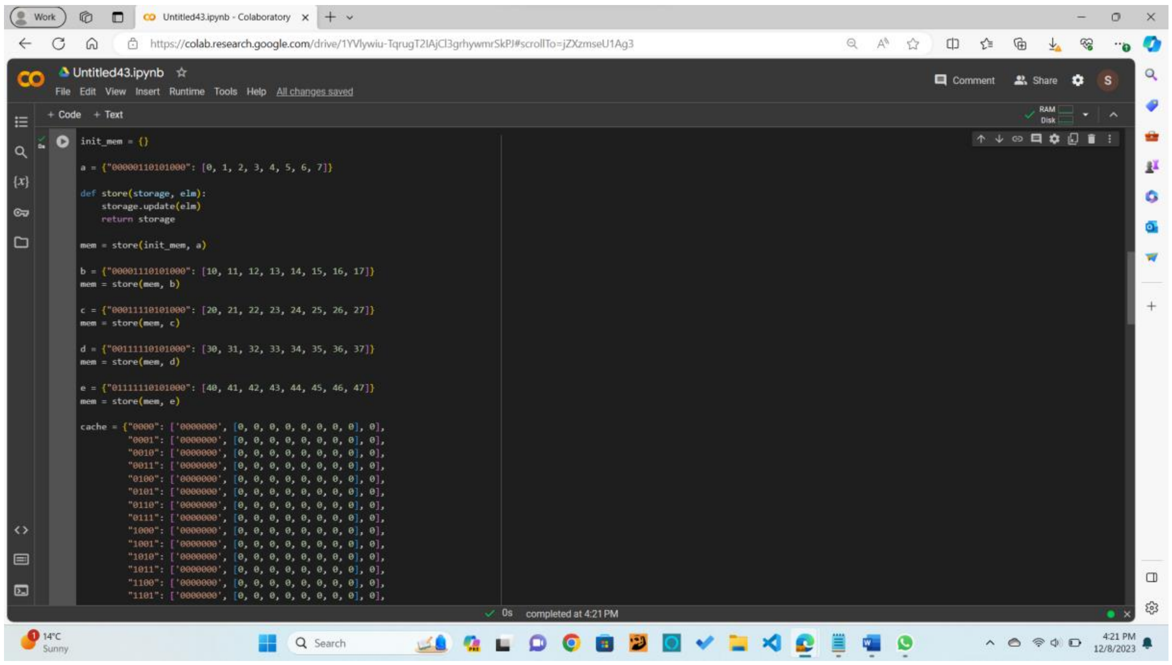
```
Initial mem {800: 123, 900: 1000}
Updated mem {800: 900, 900: 1000, 1500: 700}
imm_load_ac 800
direct_load_ac 900
indirect_load_ac 1000
Indexed Load AC: 700
```

```
[55] #2.
     def dir_map_cache(cache, adr, mem):
```

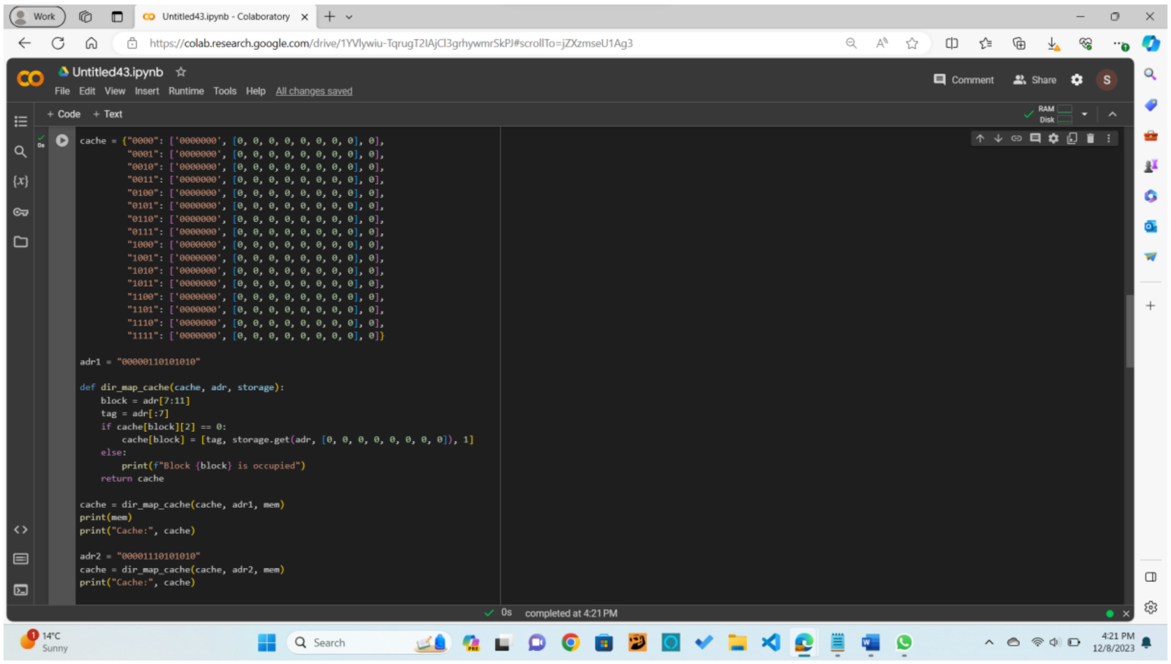
2. If the main memory consists of 2^{14} words, 2^{11} blocks will be created in it, each block holds 8 words, and cache has $16 = 2^4$ blocks, any memory address can be separate as following segments for Tag, Block and Word. In direct mapped cache, the whole block can be directly mapped to the cache line based on the values of 4-bit in Block segment. Please complete the following functions in Python program.



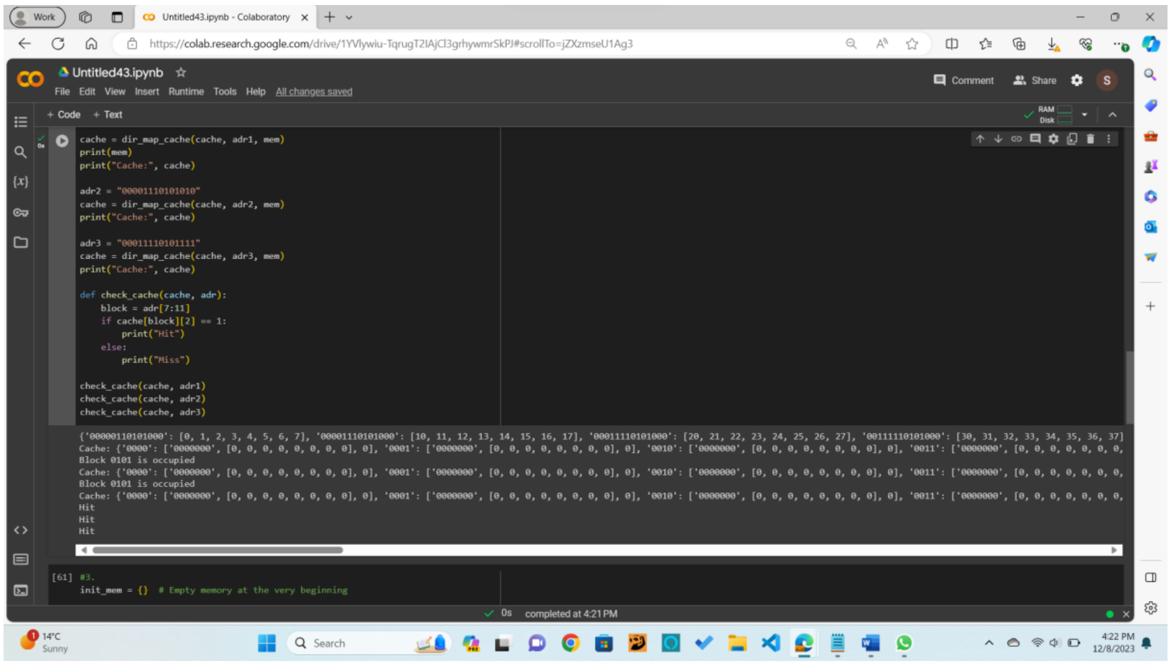
```
init_mem={}
# Take memory address as the key and all values in the block as key's value
a={"00000110101000":[0,1,2,3,4,5,6,7]}
# binary 00000110101000 is 1A8 in Hex
# "0000011": 7-bit tag;
# "0101": 4-bit block;
# "000": 1st address in the current block

# value = 0 at Address 0000011_0101_000 in main memory
# value = 1 at Address 0000011_0101_001 in main memory
# ... ...
# value = 7 at Address 0000011_0101_111 in main memory

def store(storage, elm):
  """
  Here is your program
  """

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

b={"00001110101000":[10,11,12,13,14,15,16,17]}
# bin 00001110101000 is 3A8 in Hex
mem=store(mem, b)
# {'00000110101000': [0, 1, 2, 3, 4, 5, 6, 7],
#  '00001110101000': [10, 11, 12, 13, 14, 15, 16, 17]}


# cache link format:
# 1. key -> block label;
# 2. value -> tag(7bits), values of 8 words, valid(1bit)]
cache={"0000": ["0000000", [0,0,0,0,0,0,0,0], 0],
       "0001": ["0000000", [0,0,0,0,0,0,0,0], 0],
       "0010": ["0000000", [0,0,0,0,0,0,0,0], 0],
       "0011": ["0000000", [0,0,0,0,0,0,0,0], 0],
       "0100": ["0000000", [0,0,0,0,0,0,0,0], 0],
       "0101": ["0000000", [0,0,0,0,0,0,0,0], 0],
       "0110": ["0000000", [0,0,0,0,0,0,0,0], 0],
       "0111": ["0000000", [0,0,0,0,0,0,0,0], 0],
       "1000": ["0000000", [0,0,0,0,0,0,0,0], 0],
        "1001": ["0000000", [0,0,0,0,0,0,0,0], 0],
        "1010": ["0000000", [0,0,0,0,0,0,0,0], 0],
        "1011": ["0000000", [0,0,0,0,0,0,0,0], 0],
        "1100": ["0000000", [0,0,0,0,0,0,0,0], 0],
        "1101": ["0000000", [0,0,0,0,0,0,0,0], 0],
        "1110": ["0000000", [0,0,0,0,0,0,0,0], 0],
        "1111": ["0000000", [0,0,0,0,0,0,0,0], 0]}

adr1 = "00000110101010"          # hex address: 1AA

def dir_map_cache(cache, adr, storage):
  """
  Here is your program
  """

cache=dir_map_cache(cache, adr1, mem)
# cache
# {'0000': ['0000000', [0, 0, 0, 0, 0, 0, 0, 0], 0],
#  '0001': ['0000000', [0, 0, 0, 0, 0, 0, 0, 0], 0],
#  '0010': ['0000000', [0, 0, 0, 0, 0, 0, 0, 0], 0],
#  '0011': ['0000000', [0, 0, 0, 0, 0, 0, 0, 0], 0],
#  '0100': ['0000000', [0, 0, 0, 0, 0, 0, 0, 0], 0],
#  '0101': ['0000011', [0, 1, 2, 3, 4, 5, 6, 7], 1],
#  '0110': ['0000000', [0, 0, 0, 0, 0, 0, 0, 0], 0],
#  '0111': ['0000000', [0, 0, 0, 0, 0, 0, 0, 0], 0],
#  '1000': ['0000000', [0, 0, 0, 0, 0, 0, 0, 0], 0],
#  '1001': ['0000000', [0, 0, 0, 0, 0, 0, 0, 0], 0],
#  '1010': ['0000000', [0, 0, 0, 0, 0, 0, 0, 0], 0],
#  '1011': ['0000000', [0, 0, 0, 0, 0, 0, 0, 0], 0],
#  '1100': ['0000000', [0, 0, 0, 0, 0, 0, 0, 0], 0],
#  '1101': ['0000000', [0, 0, 0, 0, 0, 0, 0, 0], 0],
#  '1110': ['0000000', [0, 0, 0, 0, 0, 0, 0, 0], 0],
#  '1111': ['0000000', [0, 0, 0, 0, 0, 0, 0, 0], 0]}

adr2 = "00001110101010"          # hex address: 3AA

cache=dir_map_cache(cache, adr2, mem)
# Block in the cache is occupied

c={"00001110111000":[20,21,22,23,24,25,26,27]}
# bin 001110101000 is 3B8 in Hex
mem=store(mem, c)
# Store block values in c to main memory
# mem
# {'00000110101000': [0, 1, 2, 3, 4, 5, 6, 7],
#  '00001110101000': [10, 11, 12, 13, 14, 15, 16, 17],
#  '00001110111000': [20, 21, 22, 23, 24, 25, 26, 27]}

adr3 = "00001110111111"         # hex address: 7BF
cache=dir_map_cache(cache, adr3, mem)
# cache
# {'0000': ['0000000', [0, 0, 0, 0, 0, 0, 0, 0], 0],
#  '0001': ['0000000', [0, 0, 0, 0, 0, 0, 0, 0], 0],
#  '0010': ['0000000', [0, 0, 0, 0, 0, 0, 0, 0], 0],
#  '0011': ['0000000', [0, 0, 0, 0, 0, 0, 0, 0], 0],
#  '0100': ['0000000', [0, 0, 0, 0, 0, 0, 0, 0], 0],
#  '0101': ['0000011', [0, 1, 2, 3, 4, 5, 6, 7], 1],
#  '0110': ['0000000', [0, 0, 0, 0, 0, 0, 0, 0], 0],
#  '0111': ['0000111', [20, 21, 22, 23, 24, 25, 26, 27], 1],
#  '1000': ['0000000', [0, 0, 0, 0, 0, 0, 0, 0], 0],
#  '1001': ['0000000', [0, 0, 0, 0, 0, 0, 0, 0], 0],
#  '1010': ['0000000', [0, 0, 0, 0, 0, 0, 0, 0], 0],
#  '1011': ['0000000', [0, 0, 0, 0, 0, 0, 0, 0], 0],
#  '1100': ['0000000', [0, 0, 0, 0, 0, 0, 0, 0], 0],
#  '1101': ['0000000', [0, 0, 0, 0, 0, 0, 0, 0], 0],
#  '1110': ['0000000', [0, 0, 0, 0, 0, 0, 0, 0], 0],
#  '1111': ['0000000', [0, 0, 0, 0, 0, 0, 0, 0], 0]}


def check_cache(cache, adr):
  """
  Here is your program
  """

check_cache(cache, adr1) # Check if adr1="00000110101010" is saved in cache or not
# Hit
check_cache(cache, adr2) # Check if adr2="00001110101010" is saved in cache or not
# Miss
check_cache(cache, adr3) # Check if adr3="00001110111111" is saved in cache or not
# Hit
```
→



```
init_mem = {}

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

def store(storage, elm):
    storage.update(elm)
    return storage

mem = store(init_mem, a)

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

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

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

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

cache = {"0000": ['0000000', [0, 0, 0, 0, 0, 0, 0, 0], 0],
         "0001": ['0000000', [0, 0, 0, 0, 0, 0, 0, 0], 0],
         "0010": ['0000000', [0, 0, 0, 0, 0, 0, 0, 0], 0],
         "0011": ['0000000', [0, 0, 0, 0, 0, 0, 0, 0], 0],
         "0100": ['0000000', [0, 0, 0, 0, 0, 0, 0, 0], 0],
         "0101": ['0000000', [0, 0, 0, 0, 0, 0, 0, 0], 0],
         "0110": ['0000000', [0, 0, 0, 0, 0, 0, 0, 0], 0],
         "0111": ['0000000', [0, 0, 0, 0, 0, 0, 0, 0], 0],
         "1000": ['0000000', [0, 0, 0, 0, 0, 0, 0, 0], 0],
         "1001": ['0000000', [0, 0, 0, 0, 0, 0, 0, 0], 0],
         "1010": ['0000000', [0, 0, 0, 0, 0, 0, 0, 0], 0],
         "1011": ['0000000', [0, 0, 0, 0, 0, 0, 0, 0], 0],
         "1100": ['0000000', [0, 0, 0, 0, 0, 0, 0, 0], 0],
         "1101": ['0000000', [0, 0, 0, 0, 0, 0, 0, 0], 0],
```



```
cache = {"0000": ['0000000', [0, 0, 0, 0, 0, 0, 0, 0], 0],
         "0001": ['0000000', [0, 0, 0, 0, 0, 0, 0, 0], 0],
         "0010": ['0000000', [0, 0, 0, 0, 0, 0, 0, 0], 0],
         "0011": ['0000000', [0, 0, 0, 0, 0, 0, 0, 0], 0],
         "0100": ['0000000', [0, 0, 0, 0, 0, 0, 0, 0], 0],
         "0101": ['0000000', [0, 0, 0, 0, 0, 0, 0, 0], 0],
         "0110": ['0000000', [0, 0, 0, 0, 0, 0, 0, 0], 0],
         "0111": ['0000000', [0, 0, 0, 0, 0, 0, 0, 0], 0],
         "1000": ['0000000', [0, 0, 0, 0, 0, 0, 0, 0], 0],
         "1001": ['0000000', [0, 0, 0, 0, 0, 0, 0, 0], 0],
         "1010": ['0000000', [0, 0, 0, 0, 0, 0, 0, 0], 0],
         "1011": ['0000000', [0, 0, 0, 0, 0, 0, 0, 0], 0],
         "1100": ['0000000', [0, 0, 0, 0, 0, 0, 0, 0], 0],
         "1101": ['0000000', [0, 0, 0, 0, 0, 0, 0, 0], 0],
         "1110": ['0000000', [0, 0, 0, 0, 0, 0, 0, 0], 0],
         "1111": ['0000000', [0, 0, 0, 0, 0, 0, 0, 0], 0]}

adr1 = "00000110101010"

def dir_map_cache(cache, adr, storage):
    block = adr[7:11]
    tag = adr[:7]
    if cache[block][2] == 0:
        cache[block] = [tag, storage.get(adr, [0, 0, 0, 0, 0, 0, 0, 0]), 1]
    else:
        print(f"Block {block} is occupied")
    return cache

cache = dir_map_cache(cache, adr1, mem)
print(mem)
print("Cache:", cache)

adr2 = "00001110101010"
cache = dir_map_cache(cache, adr2, mem)
print("Cache:", cache)
```

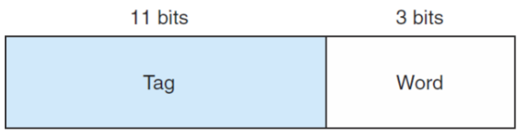


```
cache = dir_map_cache(cache, adr1, mem)
print(mem)
print("Cache:", cache)

adr2 = "00001110101010"
cache = dir_map_cache(cache, adr2, mem)
print("Cache:", cache)

adr3 = "00011110101111"
cache = dir_map_cache(cache, adr3, mem)
print("Cache:", cache)

def check_cache(cache, adr):
    block = adr[7:11]
    if cache[block][2] == 1:
        print("Hit")
    else:
        print("Miss")

check_cache(cache, adr1)
check_cache(cache, adr2)
check_cache(cache, adr3)
```

```
{'00000110101000': [0, 1, 2, 3, 4, 5, 6, 7], '00001110101000': [10, 11, 12, 13, 14, 15, 16, 17], '00011110101000': [20, 21, 22, 23, 24, 25, 26, 27], '00111110101000': [30, 31, 32, 33, 34, 35, 36, 37]
Cache: {'0000': ['0000000', [0, 0, 0, 0, 0, 0, 0, 0], 0], '0001': ['0000000', [0, 0, 0, 0, 0, 0, 0, 0], 0], '0010': ['0000000', [0, 0, 0, 0, 0, 0, 0, 0], 0], '0011': ['0000000', [0, 0, 0, 0, 0, 0, 0,
Block 0101 is occupied
Cache: {'0000': ['0000000', [0, 0, 0, 0, 0, 0, 0, 0], 0], '0001': ['0000000', [0, 0, 0, 0, 0, 0, 0, 0], 0], '0010': ['0000000', [0, 0, 0, 0, 0, 0, 0, 0], 0], '0011': ['0000000', [0, 0, 0, 0, 0, 0, 0,
Block 0101 is occupied
Cache: {'0000': ['0000000', [0, 0, 0, 0, 0, 0, 0, 0], 0], '0001': ['0000000', [0, 0, 0, 0, 0, 0, 0, 0], 0], '0010': ['0000000', [0, 0, 0, 0, 0, 0, 0, 0], 0], '0011': ['0000000', [0, 0, 0, 0, 0, 0, 0,
Hit
Hit
Hit
```

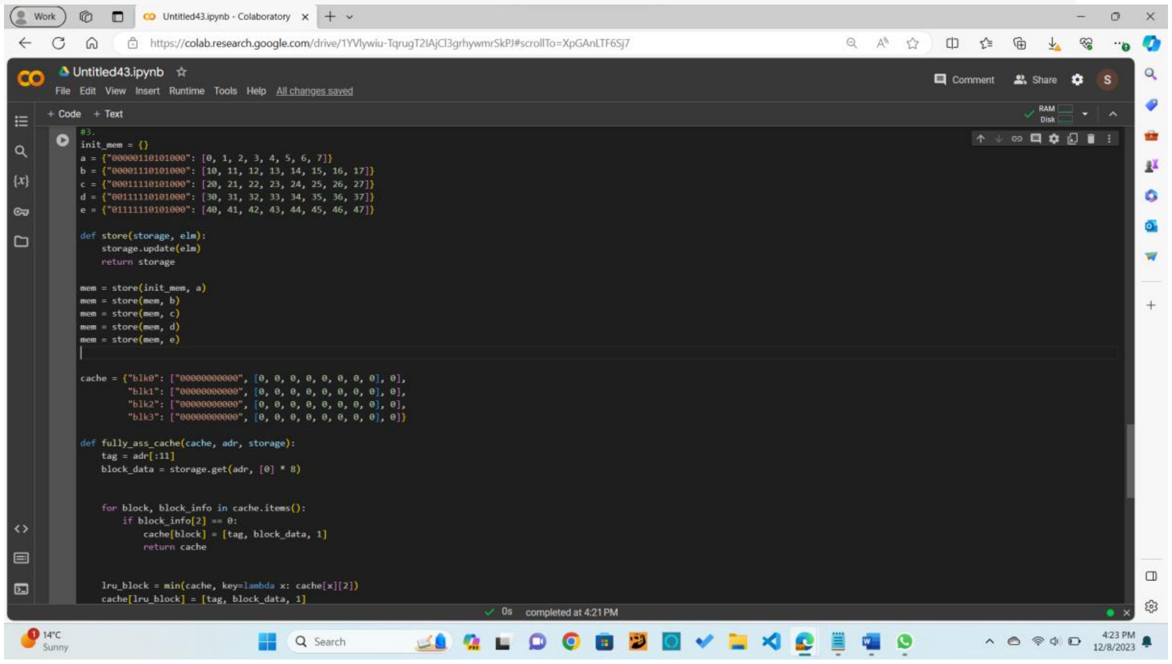
```
[61] #3.
     init_mem = {}  # Empty memory at the very beginning
```

3. To avoid thrashing issue in direct mapped cache as above, the technique of fully associative cache will be taken. The 14-bit memory address can be separated as follows for Tag and word segments. Assuming that there are only 4 cache lines in the cache, the block in the main memory can be mapped to any cache line if the valid bit is 1 showing it is available.

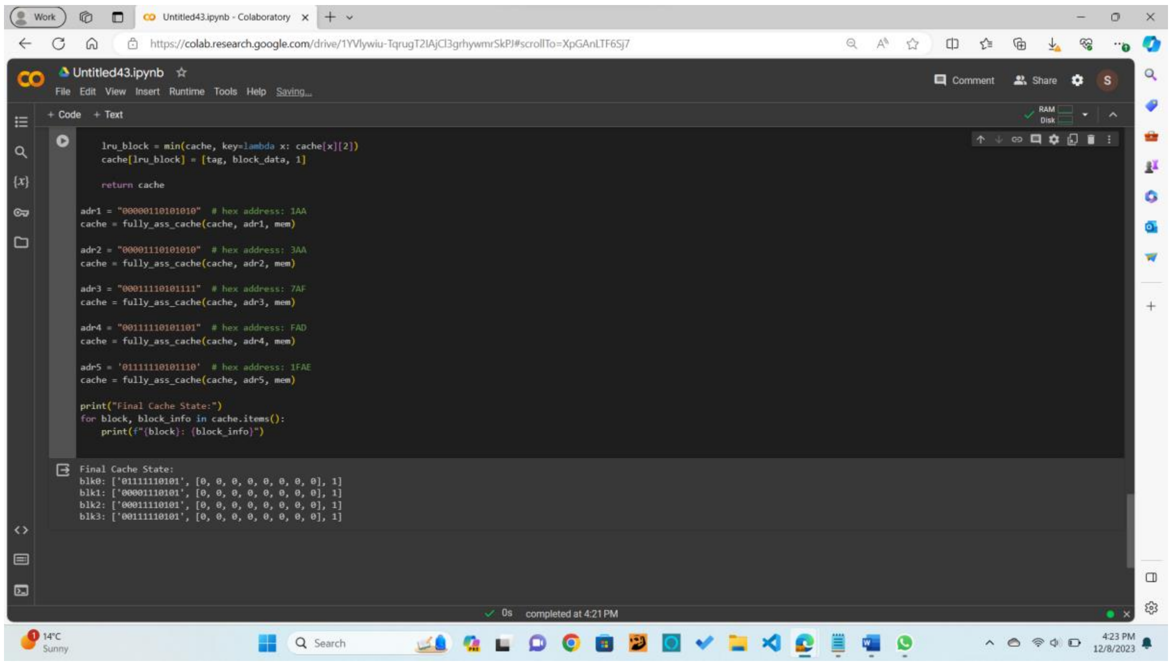


```
init_mem={}

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

def store(storage, elm):
  """
  Here is your program
  """

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

b={"00001110101000":[10,11,12,13,14,15,16,17]}
mem=store(init_mem, b)
# mem={'00000110101000': [0, 1, 2, 3, 4, 5, 6, 7],
#       '00001110101000': [10, 11, 12, 13, 14, 15, 16, 17]}

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

```
# mem={'00000110101000': [0, 1, 2, 3, 4, 5, 6, 7],
#      '00001110101000': [10, 11, 12, 13, 14, 15, 16, 17],
#      '00011110101000': [20, 21, 22, 23, 24, 25, 26, 27]}

d={"00111110101000":[30,31,32,33,34,35,36,37]}
mem=store(init_mem, d)
# mem={'00000110101000': [0, 1, 2, 3, 4, 5, 6, 7],
#      '00001110101000': [10, 11, 12, 13, 14, 15, 16, 17],
#      '00011110101000': [20, 21, 22, 23, 24, 25, 26, 27],
#      '00111110101000': [30, 31, 32, 33, 34, 35, 36, 37]}

e={"01111110101000":[40,41,42,43,44,45,46,47]}
mem=store(init_mem, e)
# mem={'00000110101000': [0, 1, 2, 3, 4, 5, 6, 7],
#      '00001110101000': [10, 11, 12, 13, 14, 15, 16, 17],
#      '00011110101000': [20, 21, 22, 23, 24, 25, 26, 27],
#      '00111110101000': [30, 31, 32, 33, 34, 35, 36, 37],
#      '01111110101000': [40, 41, 42, 43, 44, 45, 46, 47]}

# Initialize cache
# cache format: key -> block label
#               value -> tag(11bits), values of 8 words, valid(1bit)
# Assume that there are only 4 cache lines
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 fully_ass_cache(cache, adr, storage):
  """
  Here is your program
  """

adr1 = "00000110101010"         # hex address: 1AA
cache=fully_ass_cache(cache, adr1, mem)
# cache={'blk0': ['00000110101', [0, 1, 2, 3, 4, 5, 6, 7], 1],
#        '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]}

adr2 = "00001110101010"         # hex address: 3AA
cache=fully_ass_cache(cache, adr2, mem)
# cache={'blk0': ['00000110101', [0, 1, 2, 3, 4, 5, 6, 7], 1],
#        'blk1': ['00001110101', [10, 11, 12, 13, 14, 15, 16, 17], 1],
#        'blk2': ['00000000000', [0, 0, 0, 0, 0, 0, 0, 0], 0],
#         'blk3': ['00000000000', [0, 0, 0, 0, 0, 0, 0, 0], 0]}

adr3 = "00011110101111"          # hex address: 7AF
cache=fully_ass_cache(cache, adr3, mem)
# cache={'blk0': ['00000110101', [0, 1, 2, 3, 4, 5, 6, 7], 1],
#         'blk1': ['00001110101', [10, 11, 12, 13, 14, 15, 16, 17], 1],
#         'blk2': ['00011110101', [20, 21, 22, 23, 24, 25, 26, 27], 1],
#         'blk3': ['00000000000', [0, 0, 0, 0, 0, 0, 0, 0], 0]}

adr4 = "00111110101101"          # hex address: FAD
cache=fully_ass_cache(cache, adr4, mem)
# cache={'blk0': ['00000110101', [0, 1, 2, 3, 4, 5, 6, 7], 1],
#         'blk1': ['00001110101', [10, 11, 12, 13, 14, 15, 16, 17], 1],
#         'blk2': ['00011110101', [20, 21, 22, 23, 24, 25, 26, 27], 1],
#         'blk3': ['00111110101', [30, 31, 32, 33, 34, 35, 36, 37], 1]}

adr5 = '01111110101110'          # hex address: 1FAE
cache=fully_ass_cache(cache, adr5, mem)
# One cache line needs to be evicted
```


Untitled43.ipynb - Colaboratory
https://colab.research.google.com/drive/1YVlywiu-TqrugT2IAjCl3grhywmrSkPJ#scrollTo=XpGAnLTF6Sj7
Untitled43.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
+ Code + Text
0s completed at 4:21 PM
14°C Sunny
Search
4:23 PM 12/8/2023

```
#3.
init_mem = {}
a = {"00000110101000": [0, 1, 2, 3, 4, 5, 6, 7]}
b = {"00001110101000": [10, 11, 12, 13, 14, 15, 16, 17]}
c = {"00011110101000": [20, 21, 22, 23, 24, 25, 26, 27]}
d = {"00111110101000": [30, 31, 32, 33, 34, 35, 36, 37]}
e = {"01111110101000": [40, 41, 42, 43, 44, 45, 46, 47]}

def store(storage, elm):
    storage.update(elm)
    return storage

mem = store(init_mem, a)
mem = store(mem, b)
mem = store(mem, c)
mem = store(mem, d)
mem = store(mem, e)


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 fully_ass_cache(cache, adr, storage):
    tag = adr[:11]
    block_data = storage.get(adr, [0] * 8)


    for block, block_info in cache.items():
        if block_info[2] == 0:
            cache[block] = [tag, block_data, 1]
            return cache


    lru_block = min(cache, key=lambda x: cache[x][2])
    cache[lru_block] = [tag, block_data, 1]
```



```
    lru_block = min(cache, key=lambda x: cache[x][2])
    cache[lru_block] = [tag, block_data, 1]

    return cache

adr1 = "00000110101010"  # hex address: 1AA
cache = fully_ass_cache(cache, adr1, mem)

adr2 = "00001110101010"  # hex address: 3AA
cache = fully_ass_cache(cache, adr2, mem)

adr3 = "00011110101111"  # hex address: 7AF
cache = fully_ass_cache(cache, adr3, mem)

adr4 = "00111110101101"  # hex address: FAD
cache = fully_ass_cache(cache, adr4, mem)

adr5 = '01111110101110'  # hex address: 1FAE
cache = fully_ass_cache(cache, adr5, mem)

print("Final Cache State:")
for block, block_info in cache.items():
    print(f"{block}: {block_info}")
```

```
Final Cache State:
blk0: ['01111110101', [0, 0, 0, 0, 0, 0, 0, 0], 1]
blk1: ['00001110101', [0, 0, 0, 0, 0, 0, 0, 0], 1]
blk2: ['00011110101', [0, 0, 0, 0, 0, 0, 0, 0], 1]
blk3: ['00111110101', [0, 0, 0, 0, 0, 0, 0, 0], 1]
```