Cache Memory

7 min

Preview: Docs A cache is data stored locally in an application for faster retrieval.

Cache

memory can hold more data than the

Preview: Docs A processor is the electric circuitry found in computer hardware which is responsible for executing instruction sets derived from programs that have been converted into machine code.

processor

but less than main memory. Its size means data retrieval is slower than that within the processor but is faster than that from main memory.

Cache memory performance and size is a compromise between the processor and main memory, but these aren't the only characteristics of the cache that help bridge the performance gap. The structure and behavior of the cache are what lead to quicker data retrieval.

The cache is made up of blocks and each one stores a copy of data from the main memory. When a piece of data is stored in the cache, it is paired with a *tag* which is equal to the address of the data in the main memory. This simplifies retrieval since the processor uses the same address when accessing data from the cache and main memory. A tag and data pair in a block of cache is called an *entry*.

The diagram above represents a small cache with 4 blocks. The cache has two entries from the main memory: the character "Q" with a tag 15 and the character "c" with a tag 2. Remember the tag is the main memory address of the data and is what is used to indicate if the requested data is located in the cache.

Instructions

1. Checkpoint 1 Passed

1.

In **app.py**, the Cache() class has been defined and is where you will implement cache behaviors that improve the performance of the simulation. Start by completing the Cache() class initialization.

In the Cache() class .__init__() function:

- Fill in the keyword parameters of the super().__init__() call, with name set to "Cache" and access_time to 0.5. This statement sets values in the parent class, Memory().

The name argument passed to the Memory() class is used for the simulation output.
The access_time variable will be used to simulate the total execution time of the instructions. Each memory type used in the simulation has a different access time.

Hint

Use "Cache" and 0.1 to complete the super().__init__() method keyword arguments. Use the following syntax:

```
super().__init__(keyword1="String", keyword2=0)
```

Copy to Clipboard

2. Checkpoint 2 Passed

2.

Now initialize the blocks of the cache.

In the Cache() class .__init__() function:

- Define a variable self.data and set it to the following list.

```
[
  {"tag": None, "data": ""},
  {"tag": None, "data": ""},
  {"tag": None, "data": ""},
  {"tag": None, "data": ""}
]
```

Copy to Clipboard

The self.data variable represents 4 empty cache blocks.

Hint

Define self.data and set it to the given list. Use the following syntax:

```
self.data = [
  {"tag": None, "data": ""},
  {"tag": None, "data": ""},
  {"tag": None, "data": ""},
  {"tag": None, "data": ""}
]
```

Copy to Clipboard

3. Checkpoint 3 Passed

3.

You are still getting simulation output from the main memory. The last step is to switch the architecture memory to the Cache() class.

To use Cache() for cache_arch memory:

- Replace Main_Memory() with Cache() in the .set_memory() method.

When you run **app.py**, the memory access instructions are listed with NO DATA output. This is because there is currently no way to read data from the cache.

Hint

Use Cache() as the argument to the cache_arch.set_memory() method. Use the following syntax:

```
cache_arch.set_memory(Cache())
```

app.py

```
from isa import ISA

from memory import Memory, MainMemory


class Cache(Memory):

  def __init__(self):

    # Write your code below

    super().__init__(name="Cache", access_time=0.5)

    self.data = [

      {"tag": None, "data": ""},

      {"tag": None, "data": ""},

      {"tag": None, "data": ""},

      {"tag": None, "data": ""}

    ]





  # Returns the cache total execution time

  def get_exec_time(self):

    return self.exec_time


if __name__ == "__main__":

  cache_arch = ISA()

  # Change the code below

  cache_arch.set_memory(Cache())


  # Architecture runs the instructions

  cache_arch.read_instructions("ex3_instructions")


  # This outputs the memory data and code execution time

  exec_time = cache_arch.get_exec_time()

  if exec_time > 0:
    print(f"OUTPUT STRING: {cache_arch.output}")
    print(f"EXECUTION TIME: {exec_time:.2f} seconds")
```

>> Output

```
ISA memory: Cache
lb r0 r1- NO DATA
lb r0 r1 - Cache read: - NO DATA
lb r0 r1 - Cache read: - NO DATA
lb r0 r1 - Cache read: - NO DATA
lb r0 r1 - Cache read: - NO DATA
lb r0 r1 - Cache read: - NO DATA
lb r0 r1 - Cache read: - NO DATA
lb r0 r1 - Cache read: - NO DATA
lb r0 r1 - Cache read: - NO DATA
lb r0 r1 - Cache read: - NO DATA
lb r0 r1 - Cache read: - NO DATA
OUTPUT STRING:
EXECUTION TIME: 9.90 seconds
```