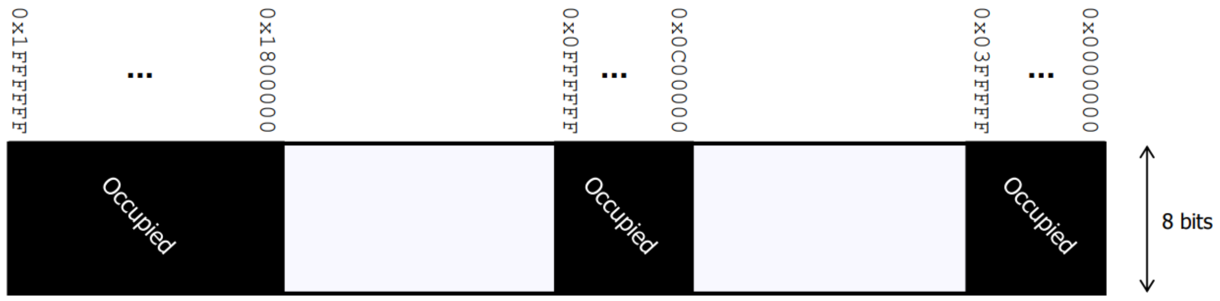
Practice Sheet 3
CSE 112 Computer Organization

Q1.
The figure below depicts the entire memory space of a microprocessor. Each memory address occupies one byte.



a. What is the size (in bytes, KB, or MB) of the memory space? What is the address bus size of the microprocessor?
b. If we have a memory chip of 4MB, how many bits do we require to address 4MB of memory?
c. We want to connect the 4MB memory chip to the microprocessor. For optimal implementation, we must place those 4MB in an address range where every single address shares some MSBs (e.g.: 0x1C00000 to 0x1FFFFFF). In other words, the starting address where this chip is mapped must be a multiple of 4MB. Find the list of all the possible address ranges that the 4MB memory chip can occupy. You can only use the non-occupied portions of the memory space.

Solution:

The figure below depicts the entire memory space of a microprocessor. Each memory address occupies one byte. (12 pts)

- What is the size (in bytes, KB, or MB) of the memory space? What is the address bus size of the microprocessor?

  Address space: `0x0000000` to `0x1FFFFFF`. To represent all these addresses, we require 25 bits. So, the address bus size of the microprocessor is 25 bits. The size of the memory space is then 2^{25}=32 MB.

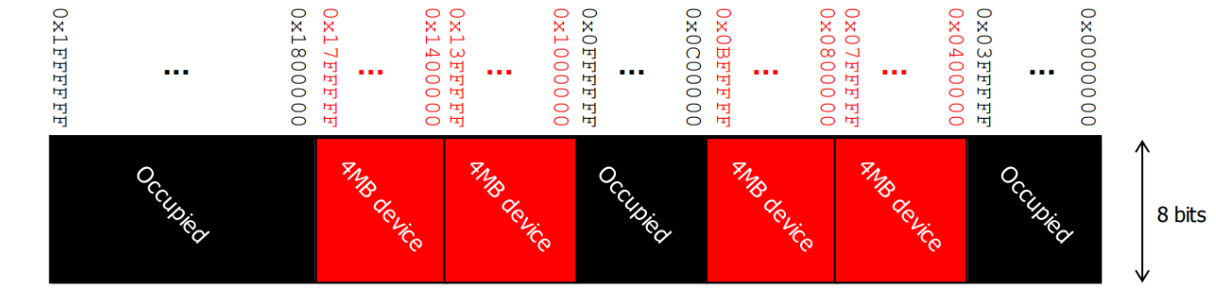
- If we have a memory chip of 4MB, how many bits do we require to address 4MB of memory? (2 pts)

  4MB = 2^{22} bytes. Thus, we require 22 bits to address only the memory device.

- We want to connect the 4MB memory chip to the microprocessor. For optimal implementation, we must place those 4MB in an address range where every single address shares some MSBs (e.g.: `0x1C00000` to `0x1FFFFFF`). Provide a list of all the possible address ranges that the 4MB memory chip can occupy. You can only use any of the non-occupied portions of the memory space as shown below. (8 pts)

  The 22-bit address range for an 4MB memory would be: `0x0000000` to `0x03FFFFF`. To place this range within the 25-bit memory space in the figure, we have four options:

`0x0400000` to `0x07FFFFF`	`0x0800000` to `0x0BFFFFF`	`0x1000000` to `0x13FFFFF`	`0x1400000` to `0x17FFFFF`



Reference

https://www.secs.oakland.edu/~llamocca/Courses/ECE2700/W21/Homework%202.pdf

Q2.

Suppose we have a memory module with the following interfaces:

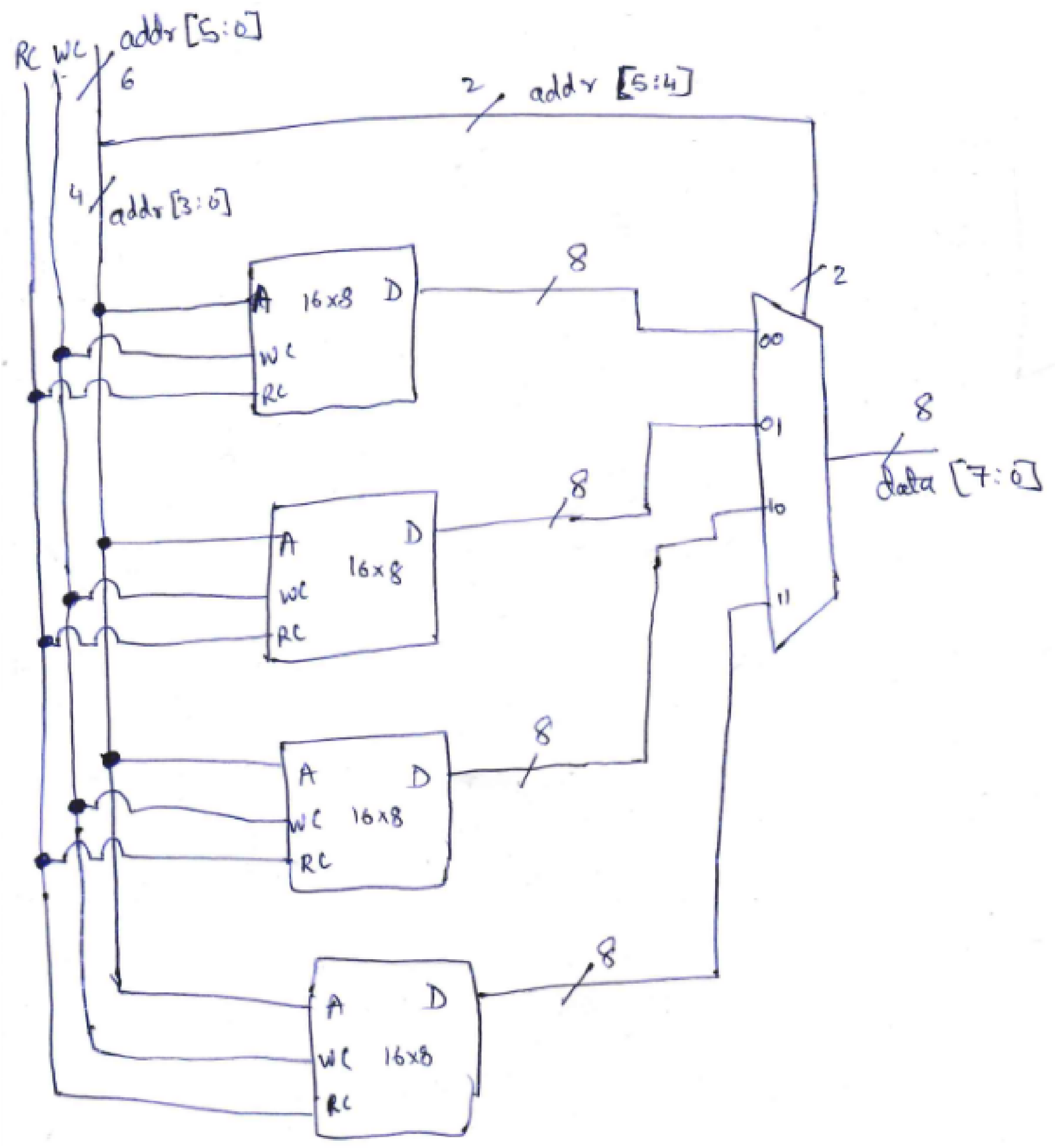
- Address bus: This specifies which selected address
- Data bus: This specifies the data which is read/written from/to the memory
- Read control:- Tells memory to read the data from the selected address
- Write control:- Tells memory to write the data to the selected address

a. Create a 64x8 memory using 16x8 memory by using multiplexers and demultiplexers.
b. Create a 16x16 memory using 16x8 memory by using multiplexers and demultiplexers.

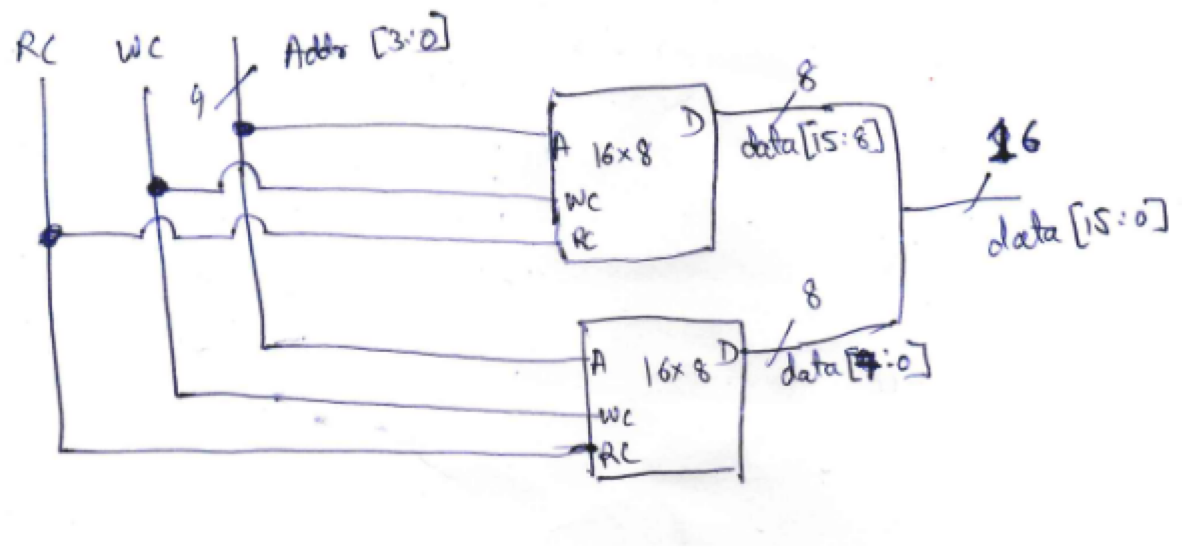
Report the size of the address and data bus sizes as well.
Note that AxB notation means A addresses, each holding B bits of data.

Solution:

a) Address bus size = 6 bits
Data bus size = 8 bits



b) Address bus size = 4 bits
Data bus size = 16 bits



Q3.
Which of the following snippets of code has better temporal locality. Explain.
a.

```
for(i = 0; i < 10; i++)
      for(j = 0; j < 10; j++)
            a[j] = b[j];
```

b.

```
for(j = 0; j < 10; j++)
      for(i = 0; i < 10; i++)
            a[j] = b[j];
```

Solution: In part A, the array is indexed by the fastest moving variable. The access pattern looks like
A[0], A[1], A[2], …, A[9], A[0], A[1], …, A[9], …
In part B, the array is indexed by the slowest moving variable. The access pattern looks like
A[0], A[0], … A[0], A[1], A[1], ..., A[1], …
Therefore pattern B has better temporal locality as the same address is accessed again and again.

Q4.

Suppose you have a 32 bit processor that has a byte addressable memory. This processor has a 512 byte fully-associative cache, with 16 byte cache lines. The cache uses LRU (least recently used) replacement policy.

a) What is the total number of cache lines?
b) How many sets does this cache have?
c) How many cache-line in each set?
d) How many bits are required for tag, index and cache-line offset?
e) Mark which segments of the address denote tag, index and cache-line offset.

Solution:

a) Total number of cache lines = Cache_size/size_of_cache_line = 512/16 = 32
b) Since this is a fully associative cache, it only has a single set.
c) All the cache lines belong to the one and only set, Therefore 32 lines in the set.
d) Cache_offset_bits = d
   2^d= (size_of_cache_line/size_of_single_memory_address)
   2^d= 16/1
   $2^d = 2^4$
   d = 4 bits

   Index_bits = i
   2^i= number of sets in cache
   Number of sets = (cache_size/(cache_line_in_each_set * size_of_cache_line))
   (number of sets in fully associative cache = 1)
   2^d= 1
   $2^d = 2^0$
   d = 0 bits

   Tag = address_size - Index_bits - Cache_offset_bits
   Tag = 32- 0-4
   Tag = 28 bits
e) Address will be broken down in the following manner:

Tag (28 bits)	Offset (4 bits)

32 4 0