Homework #5a: Virtual Memory (100 points)
Submit a compressed folder containing **VM.h**, **VM.cpp**, & **hw5.ino** to Canvas

For this assignment, you must implement demand page virtual memory using the Arduino Uno and SRAM chip (Microchip 23LC1024). Specifically, your program must meet the following requirements:

- Implements a **virtual** address space of 131,072 bytes (size of SRAM)
  - 4096 × 32-byte pages
- Implements a **physical** address space of *at least* 512 bytes
  - E.g., **16** × 32-byte pages
- Implements a **VM** class:
  - Overloads the **[ ]** (subscript) operator for byte-level virtual memory access
  - Uses **First Come First Serve** page replacement algorithm
  - Private members:
    - Page table (at least **16** entries)
    - Physical memory (array of *at least 512* bytes)
    - Microchip_23LC1024 object (SRAM chip)
    - etc.
  - Tracks page fault statistics:
    - Number of *memory references*
    - Number of *page faults*
  - Member functions:
    - VM() : Default constructor
    - getFaultRate() : returns *current* fault rate (# faults / # refs)
    - resetFaultRate() : resets page fault variables (faults = refs = 0)
  - Declared in **VM.h**
  - Implemented in **VM.cpp**
  - **VM.h** must have the following **"# defines"**
    - **#define VERBOSE 0**
      - For testing purposes…
        - If verbose == 0, DO NOT print virtual memory progress
        - If verbose == 1, print virtual memory progress to Serial
    - **#define TABLE_SIZE 16**
      - Easily adjust size of page table AND physical memory, e.g.,
        - physical memory size = TABLE_SIZE * PAGE_SIZE
        - physical memory size = 16 * 32
        - physical memory size = 512 bytes
    - **#define SRAM 5**
    - **#define HOLD 7**
      - Easily adjust digital control pins for **SRAM's *CS* and *HOLD***

- Submit an *Arduino sketch* (**hw5.ino**) that *validates* your VM class:
  - Serial baud rate of *115,200*
  - Sums two **large** vectors (*at least* 50,000 bytes each)
    - Prints *result* of vector summation to Serial
    - Prints *page fault rate* to Serial
  - Simulates **page thrashing**
    - Prints *page fault rate* to Serial

HINTS:

- inverted page table
- unsigned long
- constructor initializer list (to instantiate SRAM object)
- Arduino **Streaming** library (for easy printing)
- if (VERBOSE) //print something…
- next_page_victim = (next_page_victim + 1) % TABLE_SIZE;
- double, triple check your circuit
- **start early,** work incrementally, print often

EXAMPLES:

Example 1: Store and use multiple values from virtual addresses (on different pages)
//*VERBOSE* = 0; *TABLE_SIZE* = 24; **vm** is instance of VM object

CODE:

```
vm[7978]    = 100;
vm[127937]  = 100;
vm[39847]   = vm[7978] + vm[127937];

//print using Streaming library:
Serial << vm[7978] << " + " << vm[127937] << " = " << vm[39847] << endl;
```

OUTPUT:

```
100 + 100 = 200
```

Example 2: Save a character string to various pages in virtual memory
//VERBOSE = 0; TABLE_SIZE = 24; **vm** is instance of VM object

CODE:

```
char *msg = "Hello from Virtual Memory!!!";
for(int i = 0; i < strlen(msg); i++){
  vm[i*32] = msg[i];
}

for(int i = 0; i < strlen(msg); i++){
  Serial << (char) vm[i*32];
}
Serial << endl;
```

OUTPUT:

```
Hello from Virtual Memory!!!
```

Example 3: access virtual memory at page boundaries of arbitrary address (32,000)
//*VERBOSE* = 1; *TABLE_SIZE* = 1; **vm** is instance of VM object; page table initially empty

CODE:

```
vm[32000] = 0xAB; //page 1000, offset 0
vm[32031] = 0xCD; //page 1000, offset 31
vm[32032] = 0xEF; //page 1001, offset 0
```

OUTPUT:

```
============================
requesting virtual address: 32000
    page = 1000, offset = 0
PAGE FAULT!
    empty slot in page table: 0
    reading SRAM memory page: 1000
    physical address: 0
============================
requesting virtual address: 32031
    page = 1000, offset = 31
PAGE FOUND!
    page table entry: 0
    physical address: 31
============================
requesting virtual address: 32032
    page = 1001, offset = 0
PAGE FAULT!
    NO SPACE in page table
    page OUT: 1000
    page IN:  1001
    physical address: 0
```

Example 4: Access four arbitrary virtual memory addresses (on different pages)
//*VERBOSE* = 1; *TABLE_SIZE* = **2**; **vm** is instance of VM object; page table initially empty

CODE:

```
vm[2749]    = 0xAB;
vm[963]     = 0xCD;
vm[129775]  = 0xEF;
vm[71066]   = 0x01;
```

OUTPUT:

```
============================
requesting virtual address: 2749
    page = 85, offset = 29
PAGE FAULT!
    empty slot in page table: 0
    reading SRAM memory page: 85
    physical address: 29
============================
requesting virtual address: 963
    page = 30, offset = 3
PAGE FAULT!
    empty slot in page table: 1
    reading SRAM memory page: 30
    physical address: 35
============================
requesting virtual address: 129775
    page = 4055, offset = 15
PAGE FAULT!
    NO SPACE in page table
    page OUT: 85
    page IN:  4055
    physical address: 15
============================
requesting virtual address: 71066
    page = 2220, offset = 26
PAGE FAULT!
    NO SPACE in page table
    page OUT: 30
    page IN:  2220
    physical address: 58
```