ECE9047/ECE9407
Final Examination Sample Questions

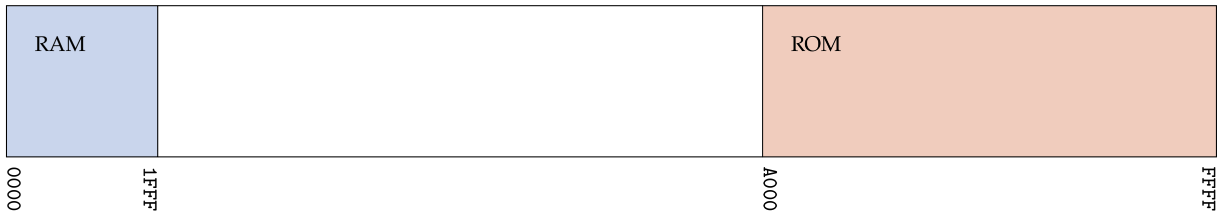
John McLeod

2020 04 10

Memory Mapping

For all problems, the system has an address space of $64\,\mathrm{kB}$ with $8\,\mathrm{bit}$ memory cells. (Note that the ARM Cortex®A9 has a much larger 32-bit address space, but there is no need to write out such long address numbers to demonstrate understanding of the concept of memory mapping, so we will use a smaller address space.)

1. Design the memory decoding scheme (minterms) for the memory map shown. All RAM and ROM chips are $8\,\mathrm{k}{\times}8$.



Answer: Note that `1000` in hexadecimal is $16^3 = 2^{12} = 2^2 \times 2^{10} = 4\,\mathrm{k}$, so this means any $8\,\mathrm{k}{\times}8$ chip covers `2000` memory addresses. The addresses `0000` to `1FFF` (inclusive) cover a range of `2000`, so we need one RAM chip. The addresses `A000` to `FFFF` (inclusive) cover a range of `6000`, so we need 3 ROM chips for the ranges `A000` to `BFFF`, `C000` to `DFFF`, and finally `E000` to `FFFF`.

The address bus for a $64\,\mathrm{k}$ address space is $16\,\mathrm{bits}$ ($64\,\mathrm{k}{=}\ 2^6 \times 2^{10} = 2^{16}$). These address bits are written as $a_{15}, \ldots a_0$, where a_{15} is the most significant bit (first) and a_0 is the least significant bit (last). For the RAM chip, the address range in binary is:

RAM start	0000 0000 0000 0000
stop	0001 1111 1111 1111

The first three bits uniquely identify the RAM chip, so the decode logic to select the RAM chip is $\bar{a}_{15}\bar{a}_{14}\bar{a}_{13}$ (here $\bar{a}_n$ means the value is true when $a_n = 0$).

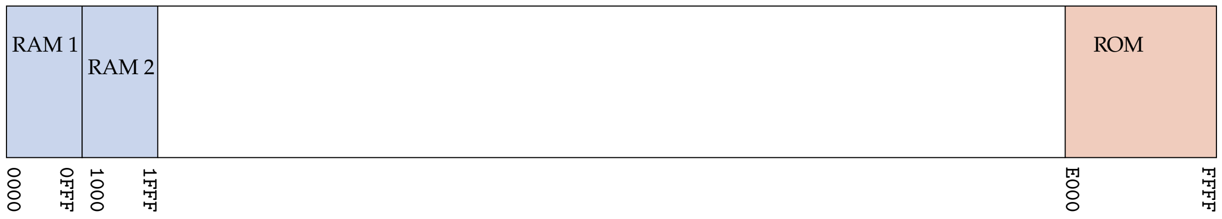
For the ROM chips, we have:

ROM 1 start	1010 0000 0000 0000
stop	1011 1111 1111 1111
ROM 2 start	1100 0000 0000 0000
stop	1101 1111 1111 1111
ROM 3 start	1110 0000 0000 0000
stop	1111 1111 1111 1111

The decode logic for ROM chips 1, 2, and 3 is: $a_{15}\bar{a}_{14}a_{13}$, $a_{15}a_{14}\bar{a}_{13}$, and $a_{15}a_{14}a_{13}$, respectively.

2. Design a memory map using two $4\,\mathrm{k}{\times}8$ RAM chips, occupying a contiguous block starting at address `0000`, and one $8\,\mathrm{k}{\times}8$ ROM chip ending at address `FFFF`. Draw the memory map, showing the boundaries of each individual memory chip, and design the memory decoding scheme.

   Answer: A $4\,\mathrm{k}$ chip takes up `1000` memory addresses, so the first RAM chip spans addresses `0000` to `0FFF`, while the second spans the addresses `1000` to `1FFF`. The ROM chip is twice as large, and spans addresses `E000` to `FFFF`.



   In binary, these chips span the memory addresses: For the ROM chips, we have:

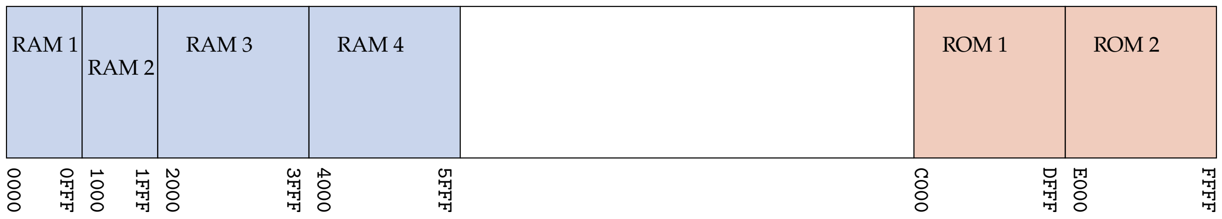
RAM 1 start	0000 0000 0000 0000
stop	0000 1111 1111 1111
RAM 2 start	0001 0000 0000 0000
stop	0001 1111 1111 1111
ROM start	1110 0000 0000 0000
stop	1111 1111 1111 1111

   The minterms are $\bar{a}_{15}, \bar{a}_{14}, \bar{a}_{13}, \bar{a}_{12}$, $\bar{a}_{15}, \bar{a}_{14}, \bar{a}_{13}, a_{12}$, and a_{15}, a_{14}, a_{13} for RAM 1, RAM 2, and ROM, respectively. Notice how bit a_{12} is part of the decoding logic for the smaller RAM chips, but part of the address bus (fed into the chip) for the larger ROM chip.

3. For this memory map, we want the RAM to occupy the memory space `0000` to `5FFF`, and the ROM to occupy the memory space `C000` to `FFFF`. We have only two $8\,\mathrm{k}{\times}8$ ROM chips, two $8\,\mathrm{k}{\times}8$ RAM chips, and two $4\,\mathrm{k}{\times}8$ RAM chips. Draw the memory map, showing the boundaries of each individual memory chip, and design the memory decoding scheme.

   Answer: The RAM space is `6000` addresses, meaning $6 \times 4\,\mathrm{k} = 24\,\mathrm{k}$. This is exactly the amount available in RAM chips, so we must use all of them. It doesn't matter how we arrange them in memory space, but it is arguably simplest to keep the smaller chips together. I will place the $4\,\mathrm{k}{\times}8$ RAM chips starting at `0000` and `1000`, and the $8\,\mathrm{k}{\times}8$ RAM chips starting at `2000` and `4000`. Other arrangements are also valid.

   The ROM space is `10000`−`C000`=`4000`, which again is exactly the amount available in ROM chips. There is only one possible arrangement, with one chip starting at `C000` and one starting at `E000`.



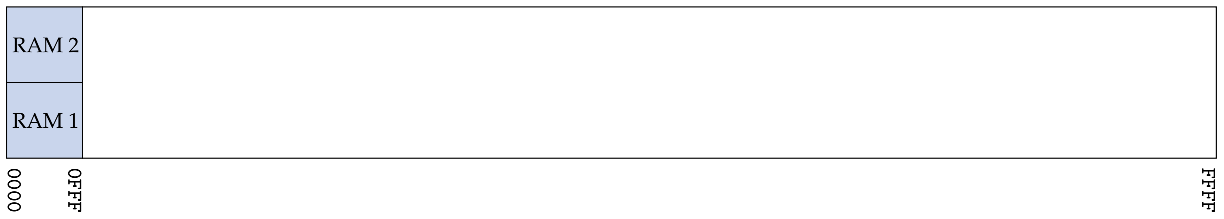
In binary, these chips span the memory addresses: For the ROM chips, we have:

RAM 1 start	0000 0000 0000 0000
stop	0000 1111 1111 1111
RAM 2 start	0001 0000 0000 0000
stop	0001 1111 1111 1111
RAM 3 start	0010 0000 0000 0000
stop	0011 1111 1111 1111
RAM 2 start	0100 0000 0000 0000
stop	0101 1111 1111 1111
ROM 1 start	1100 0000 0000 0000
stop	1101 1111 1111 1111
ROM 2 start	1110 0000 0000 0000
stop	1111 1111 1111 1111

The minterms are $\bar{a}_{15}, \bar{a}_{14}, \bar{a}_{13}, \bar{a}_{12}$, $\bar{a}_{15}, \bar{a}_{14}, \bar{a}_{13}, a_{12}$, $\bar{a}_{15}, \bar{a}_{14}, a_{13}$, and $\bar{a}_{15}, a_{14}, \bar{a}_{13}$ for RAM chips 1 through 4, respectively, and $a_{15}, a_{14}, \bar{a}_{13}$ and a_{15}, a_{14}, a_{13} for ROM 1 and ROM 2, respectively. Again, notice how bit a_{12} is part of the decoding logic for the smaller RAM chips, but part of the address bus (fed into the chip) for the larger ROM chip.

4. Design $4\,\text{kB}$ of memory starting at address `0000`. Due to suburban families hording supplies during the pandemic, we only have access to two $4\,\text{k}{\times}4$. Draw the memory map and shown the boundary of each memory chip. Remember, these memory chips are only half as wide $(4\,\text{bit})$ as needed.

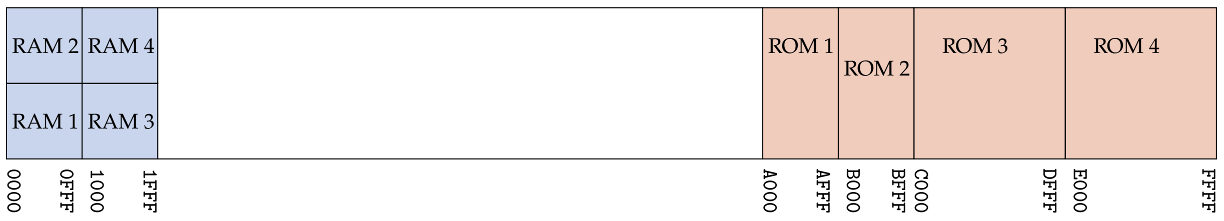
   Answer: The RAM space is `1000` addresses. Since the chips have only half the data width necessary, they must be placed "side by side" in memory.



   Both chips will be selected by the minterm $\bar{a}_{15}, \bar{a}_{14}, \bar{a}_{13}, \bar{a}_{12}$, however only half of the data lines (i.e., d_7, d_6, d_5, d_4 will connect to RAM 1 and the other half (d_3, d_2, d_1, d_0) to RAM 2. This way, both chips will be active at the same time and each will produce the lower and upper $4\,\text{bits}$ of memory needed. Notice how this is completely invisible to the programmer.

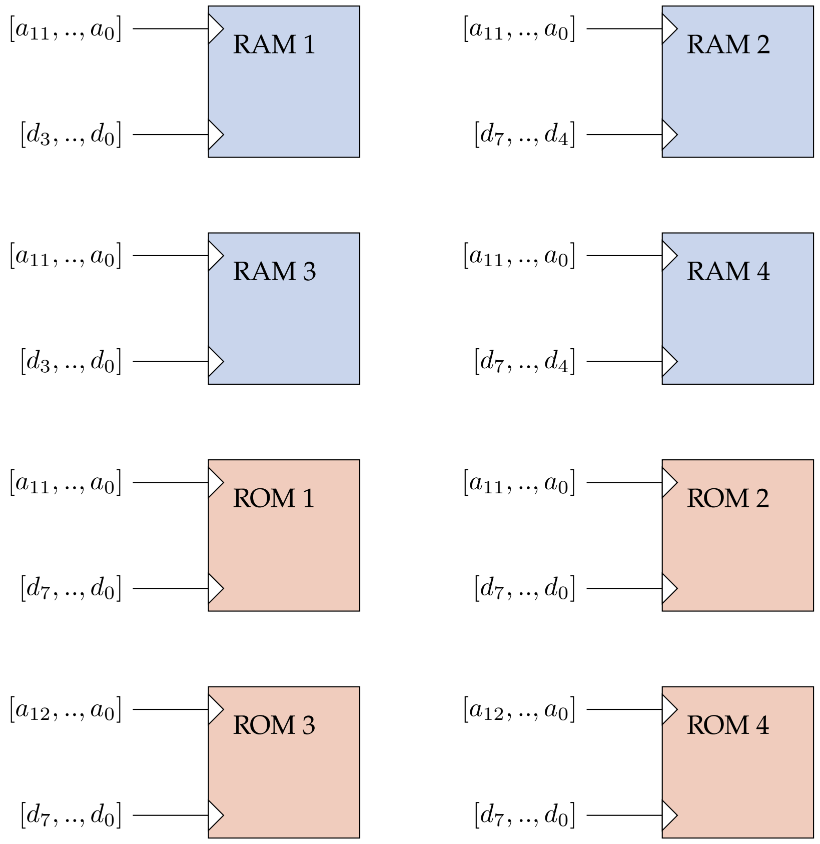
5. Use two $8\,\text{k}{\times}8$ ROM chips in a contiguous block ending at address `FFFF`, two $4\,\text{k}{\times}8$ ROM chips occupying a contiguous block starting at address `A000`, and four $4\,\text{k}{\times}4$ RAM chips in a contiguous block starting at address `0000`. Draw the memory map and shown the boundary of each memory chip, and design the memory decoding scheme. Show how the CPU lines (16-bit address bus and 8-bit data bus) are connected to each chip.

   Answer: See the memory map below. Based on the question and the previous answers, you should be able to understand how it is drawn.



Chip selection uses the minterms $\bar{a}_{15}, \bar{a}_{14}, \bar{a}_{13}, \bar{a}_{12}$ for RAM 1 and RAM 2; $\bar{a}_{15}, \bar{a}_{14}, \bar{a}_{13}, a_{12}$ for RAM 3 and RAM 4; $a_{15}, \bar{a}_{14}, a_{13}, \bar{a}_{12}$ for ROM 1; $a_{15}, \bar{a}_{14}, a_{13}, a_{12}$ for ROM 2; and $a_{15}, a_{14}, \bar{a}_{13}$ for ROM 3, and a_{15}, a_{14}, a_{13} for ROM 4.

We can sketch the connection of address and data lines to the chips as shown below.



The point of this diagram is to show which physical wires are actually attached to the chips. The remaining address wires (a_{15}, a_{14}, a_{13}, and for the smaller chips a_{12} also) are used in a logic circuit to select the chip.

Note: On the actual exam it is **not** necessary to draw memory maps to scale, as I have here. Drawing each chip as a cell in a table or spreadsheet (with a border) is probably fine, as is sketching on paper and taking a photograph or scanning it.
It is also not necessary to write the "logical not" of minterm components as $\bar{a}_{13}$ — which may be difficult if you are typing the exam in some word processors. You could instead write something like "![A13]", $-a_{13}$, or anything you like as long as you make a note explaining your notation.

Assembly Language Programming

Implement the following programs in ARMv7 assembly language. Where necessary, use the memory mapped peripherals on a DE1 SoC, as given by the online simulator we used in lab 1.

1. Generate the first 100 Fibonacci numbers and store them in memory in one sequential array.

   Answer: There is more than one way to solve this problem. An example is given below. This example has lots more comments than I expect you to write on an examination.

```
.global _start
_start:

   /* arbitrary address in memory to store Fibonnaci series*/
   ldr r0, =0x0000a000
   /* counter to keep track of Fibonnaci numbers */
   mov r1, #1
   /* first Fibonnaci number */
   mov r2, #0
   /* second Fibonnaci number */
   mov r3, #1

   /* store these in memory, increment memory address*/
   str r2, [r0], #4
   str r3, [r0], #4

/* start loop */
_fib_loop:
   /* calculate Nth Fibonnaci number, store in r4 temporarily*/
   add r4, r3, r2
   /* move (N-1)th Fibonnaci number from r3 to r2*/
   mov r2, r3
   /* move Nth Fibonnaci number from r4 to r3*/
   mov r3, r4
   /* write Nth Fibonnaci number to memory, increment memory address*/
   str r3, [r0], #4
   /* increment counter */
   add r1, #1

   /* check if counter reaches 100*/
   cmp r1, #100
   /* loop back if counter less than 100*/
   ble _fib_loop
```

---

2. Write a subroutine to find the largest number in a continuous array of numbers stored in memory.

   Answer: For a problem like this, you can assume that an address to the start of the array, and the number of elements in the array, are already provided. In this answer, I will write the code as if it were a subroutine in a larger program (you do not have to do it this way). Again, this code has considerably more comments than I expect your from your answers on the exam.

```
/* this is the subroutine to find the largest number in an array
   we assume the address to the start of the array is in r0
   we assume the number of elements in the array is in r1
   we assume all elements in the array are words
   we assume the array is unsigned
   the largest element is returned in r2
   registers r3, r4, r5 are used in this subroutine */
_find_max:
   /* initialize largest element in array */
   mov r2, #0
   /* initialize counter */
   mov r3, #0
   /* recopy the memory address of the array to r4 because
      this subroutine will modify it */
   mov r4, r0

/* loop through the array */
_array_loop:
   /* read the next value from the array and increment the memory address
   ldr r5, [r4], #4
   /* check this value with the previous maximum */
   cmp r5, r2
   /* move this value into r2 if it is the largest */
   movgt r2, r5
   /* increment counter */
   add r3, #1
   /* check if the array is done */
   cmp r3, r1
   /* loop back if we are still going through the array */
   blt _array_loop

   /* after finding the largest value, return */
   /* assume the subroutine was called with the link register */
   bx lr
```

3. Write a program to that allows the user to flip the switches (address `FF200040`) and light up a corresponding LED (address `FF200000`).

   Answer: This problem is just an infinite loop reading from the switches and writing to the LEDs.

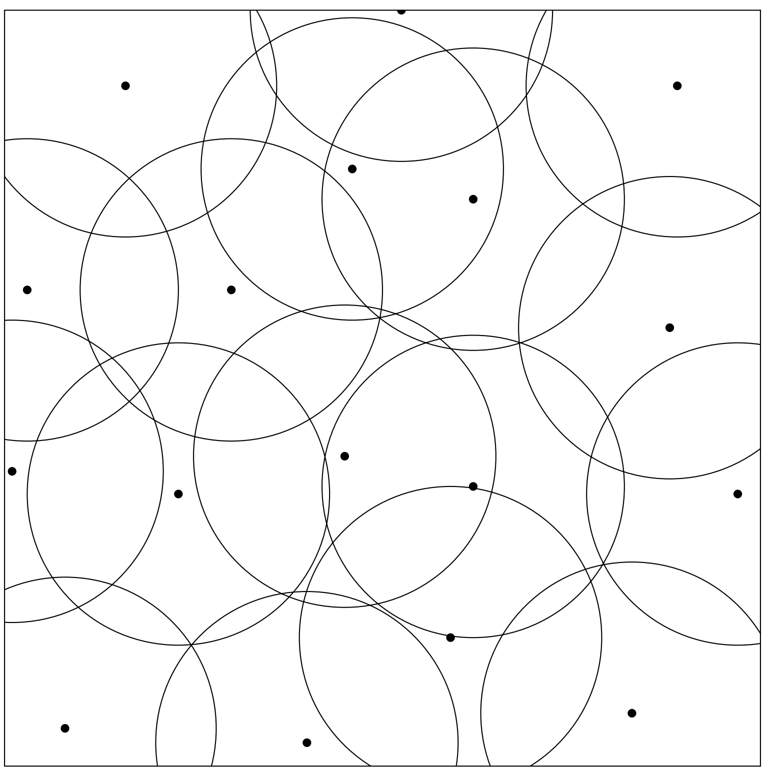
```
.global _start
_start:
   /* get switch address */
   ldr r0, =0xff200040
   /* get LED address */
   ldr r1, =0xff200000

_loop:
   /* read switches*/
   ldr r2, [r0]
   /* write to LEDs */
   str r2, [r1]
   /* that's it! */
   b _loop
```

You can test this code with the online simulator, use F3 to continuously run the program and while it is running click on various switches and watch the corresponding LEDs light up.

Wireless Sensor Design

Consider a square area of size $L \times L$ with a semi-randomly distributed set of 17 sensor nodes, as shown. The detection area of each sensor node is circular, with a radius of $0.2L$. The communication area of each sensor node is also circular, with a radius of $0.4L$.



1. What is the k-coverage of the WSN?

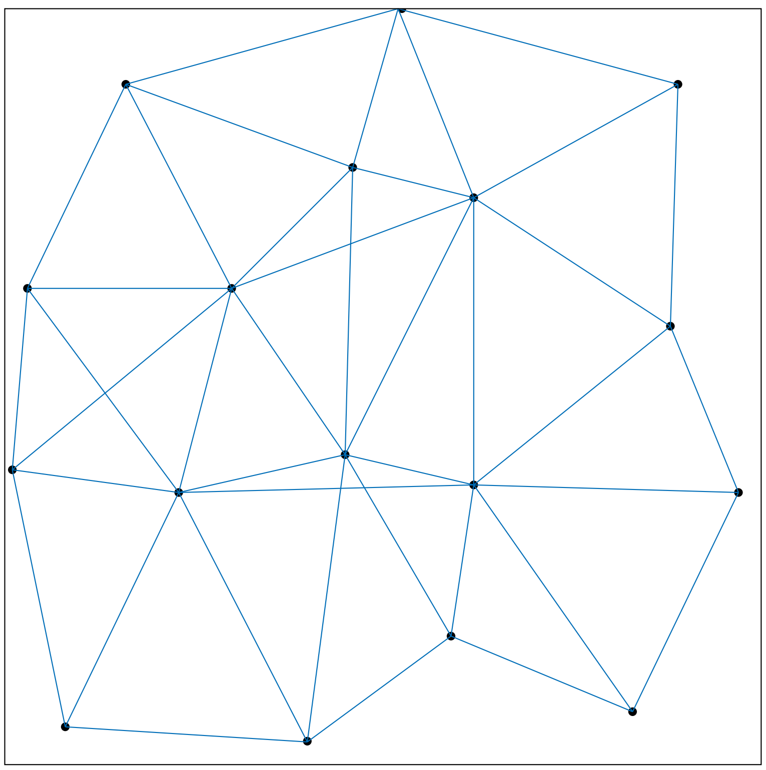
   Answer: All points in the area are covered by at least one sensor, so $k > 1$. However there are clearly areas around the edge that are only covered by one sensor, so $k = 1$ for this coverage area.

2. What is the k-connectivity of the WSN?

   Answer: The communication radius is twice the sensing radius. To find the k-connectivity we can select the nodes that look the most isolated, and measure how many other nodes they are connected with.

   In this case, several of the boundary nodes have the same low connectivity of $k = 3$, as shown.

   Note: The circles on the original figure indicate the sensor radius, not the communication radius. I apologize for any confusion.



3. In your opinion, which sensor node is the most critical — i.e., which node, if it fails, will cause the WSN to degrade the most?

   Answer: This is a subjective question, since there is no clear statement of purpose for the WSN the most important node is a matter of opinion.

   - If all areas are equally important for sensing, then arguably one of boundary nodes (one of the ones with only 3-connectivity) are most important because if they go down, they leave the most area without sensor coverage.
   - If areas in the center are more important, then one of the central nodes may be most important because some of them can also leave areas uncovered by sensors if the nodes goes down, and they are also responsible for most of the direct cross-network communication paths.
   - Since this network has 3-connectivity, there are no areas that depend on only a single node for communication. This network cannot be disconnected by any single failed node.

   Any sort of answer which discusses coverage and connectivity (and possibly other concepts) is valid.

4. Sketch your best guess for the path that traces the maximal breach distance of the WSN. Justify your answer.

   Answer: A Voronoi diagram can be constructed by drawing bisectors to the lines that connect two neighbouring points. Extend the bisectors until they intersect another bisector from a different set of points.

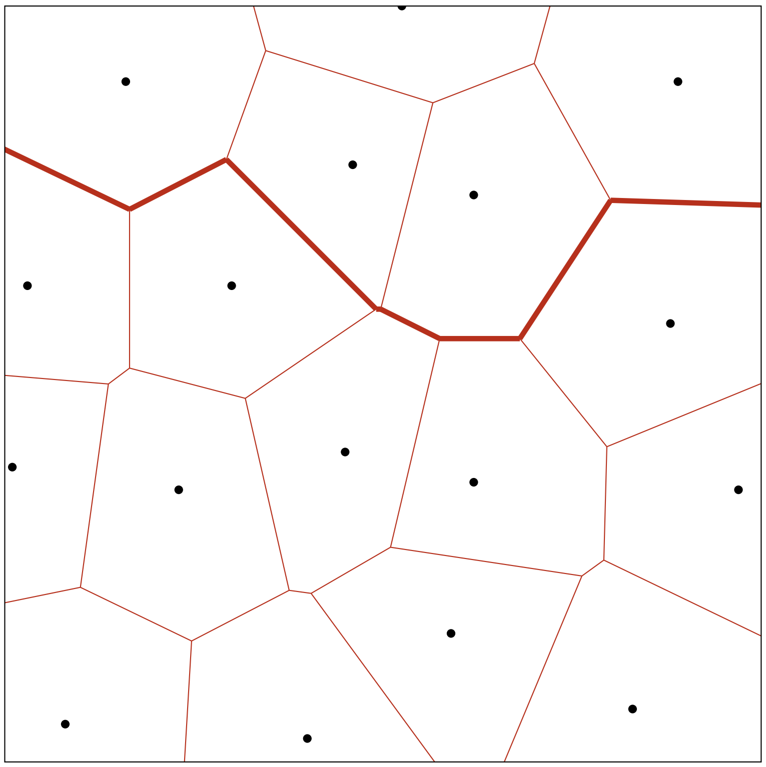
   A Voronoi diagram can also be constructed by expanding circles around each node, circles which flatten out when they push into another circle.

   There are also online Voronoi diagram generators (see here, for example: http://www.raymondhill.net/voronoi/rhill-voronoi.html), and plenty of open-source implementations in various languages (C, Python, etc.).

   Finally, it is **not** necessary to draw an accurate Voronoi diagram for a question which asks you to "sketch your best guess". Just use your judgment, and follow these rules.

- The breach line must follow straight line segments.
- These line segments need to be equally distant from the two closest nodes.
- Choose the nodes that are furthest apart for the maximal breach.

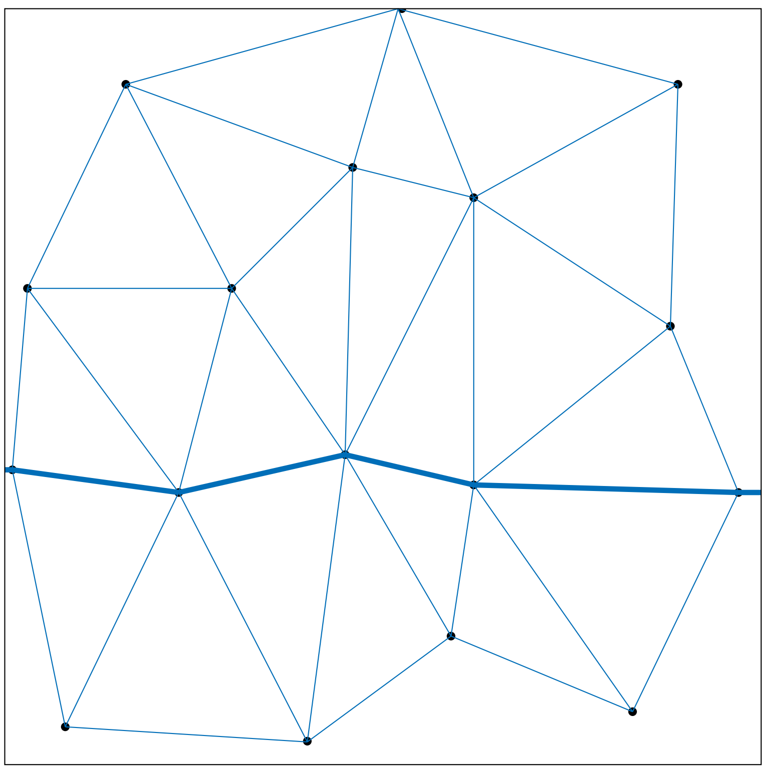
The actual maximal breach path depends on the cost function associated with each line on the Voronoi diagram. Since a cost function isn't provided, this is also subjective. A reasonable breach path is shown below. This path is fairly short and maintains a large distance from all sensor nodes relative to other paths.



5. Sketch your best guess for the path that traces the maximal support path of the WSN. Justify your answer.

   Answer: The Delauney triangulation is easier to draw than a Voronoi diagram, just connect nodes in triangles while avoiding overlapping triangles and triangles with very small internal angles. For a small network like this, it is reasonable to draw the Delauney triangulation, but it is not necessary. You can sketch the path by just connecting nodes in the network.

   Again, the support path requires a cost function which is not stated, so this question is subjected. A reasonable support path is shown below. Notice that the path has extra lines coming in and out of the network area, these lines are not part of the Delauney triangulation diagram.



6. Would a regular grid of sensor nodes produce a better or worse WSN? Explain your answer by referring to k-coverage, k-connectivity, number of nodes, reliability of the network, and any other concepts discussed in class that you think are relevant.

   Answer: A regular grid WSN could be constructed with 16 nodes, one fewer than the given WSN. We can place the corner nodes so the sensor area just touches the corner of the WSN region. This puts the lower left node at coordinates $(x, y) = 2^{-\frac{1}{2}} r_S$, where $r_S = 0.2L$ is the sensor radius. This regular spacing puts nodes approximately $0.179L$ apart. The k-connectivity is still 3, and the k-coverage is still 1, so apart from using 1 fewer sensor node, a regular grid WSN does not offer that much in performance enhancements. However since it is easier to deploy and easier to locate nodes for replacement, unless there is good reason (i.e. key areas requiring extra sensor coverage) for the given WSN, a regular grid WSN is probably better.

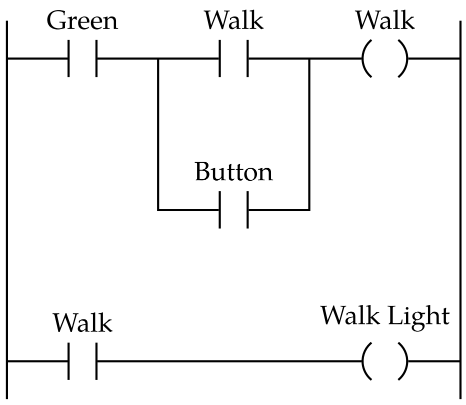
If you wish to redraw the map of the WSN precisely using your favourite software, the coordinates of the nodes, expressed as fractions of the side length L, are (0.01,0.39); (0.03,0.63); (0.08,0.05); (0.16,0.9); (0.23,0.36); (0.3,0.63); (0.4,0.031); (0.45,0.41); (0.46,0.79); (0.525,1.0); (0.59,0.17); (0.62,0.37); (0.62,0.75); (0.83,0.07); (0.88,0.58); (0.89,0.9); (0.97,0.36).

Relay Ladder Logic

Implement the following control mechanisms using relay ladder logic.

1. Walk signals at an intersection with traffic lights. The walk signals should turn on, and remain on, when a pedestrian presses the button and the main traffic light is green. The walk signal should turn off as soon as the light turns red. If no pedestrian presses a button, the walk signal should remain off.

   Answer: There is more than one way to answer this question, depending on how inputs and outputs are defined. The logic for this problem is simple: the walk signal should be on if the green light is on (or, alternatively, if the red light is off) and if the walk signal was already on, or the button is pressed. Using "Green" to represent the green light begin on, "Button" to represent the button being pushed, "Walk light" to represent the walk light being on, and "Walk" to represent an internal parameter (used as both input and output) indicating that the walk light should be on, we have the following ladder diagram.



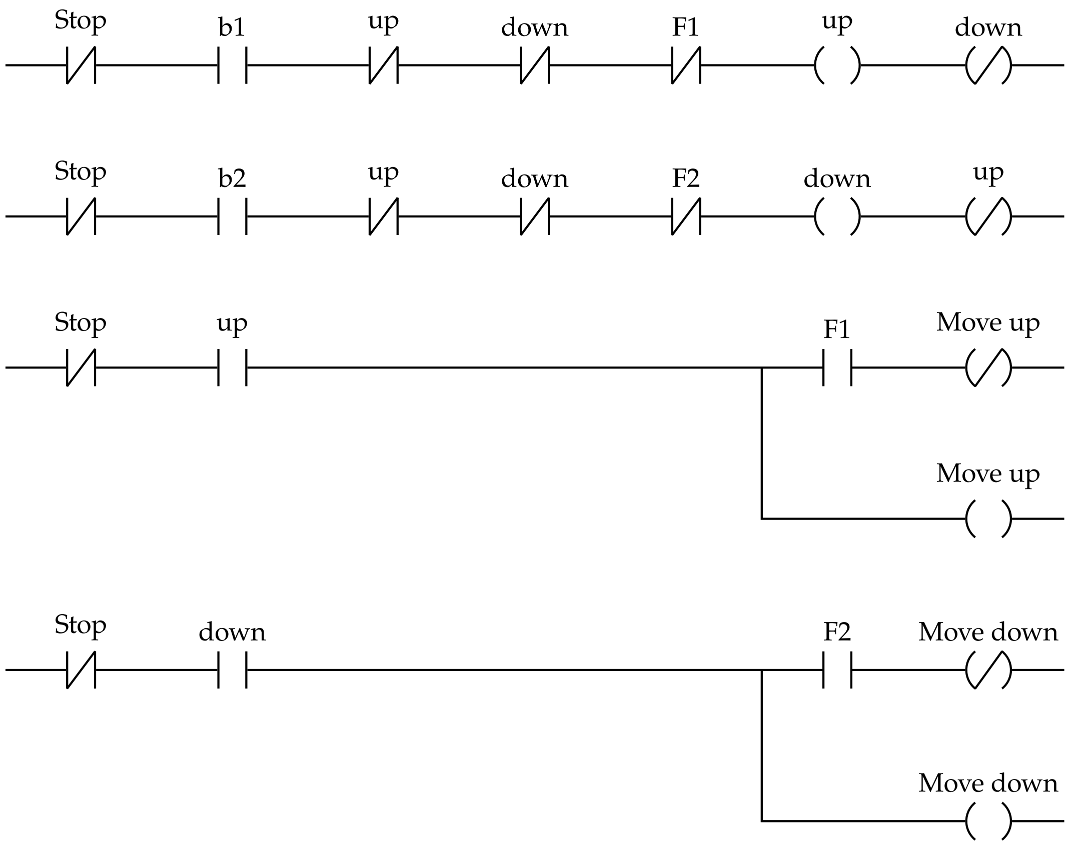
2. A 2-floor elevator. User controls must include the floor buttons and door open or close buttons. An emergency stop can optionally be implemented. Mechanical signals include whether the doors are in motion opening or closing, or stationary open or closed, and whether the elevator is moving up or down, or stationary at the first or second floor. Appropriate feedback controls need to be used so that the doors don't open while the elevator is moving, and so that the elevator doesn't start moving while the doors are open. (*This example is more challenging than I would put on an exam.*)

   Answer: There is more than one way of solving this problem. Be careful with looking up examples of elevator control on the internet — there are many ways of implementing the controls, some of which are more robust but much more complicated than the one shown here.

   The code below is only partial, it only controls the motion of the elevator, not the doors. I used multiple partially redundant lines of code to avoid complicated logic statements — this code could be compacted but on an exam there is no need to do so.

   Here "Stop" is the emergency stop button; "b1" and "b2" are the buttons for floors 1 and 2, respectively; "F1" and "F2" are the electromechanical indicators that the elevator is at floors 1 and 2, respectively; "Move up" and "Move down" control the elevator motors, and "up" and "down' are internal parameters representing that the motors should be running.

   The door controls can be implemented in the same multiple line, sequential manner. If I get a chance I will update this code to include door controls.


Stop
b1
up
down
F1
up
down
Stop
b2
up
down
F2
down
up
Stop
up
F1
Move up
Move up
Stop
down
F2
Move down
Move down