DO NOT TURN OVER

until instructed to

Practical Exam 0

On Vula you will find a template which you must use.
Along with your assembly code you must submit a Makefile. When ‘make’ is run in your submission directory a .elf file should be produced. Do not launch GDB in a recipe.
The later parts have dependencies on the earlier parts. Hence, the earlier parts need to be correctly solved before the later parts will be marked.

Part 1: (2)
Iterate through each individual byte in the DATA block, adding 1 to it and writing it to an equivalent block at the start of RAM.
In other words:
- the first byte in DATA block must be read, have 1 added to it, and stored to 0x20000000
- the second byte in DATA block must be read, have 1 added to it, and stored to 0x20000001
- the third byte in DATA block must be read, have 1 added to it, and stored to 0x20000002
… etc ...
When the automarker hits the label *copy_to_RAM_complete* it will verify the data which you have just written to RAM and then modify the block to different values.
(*Hint: See Resources->toolchain_guide.pdf, section 7.1.7 for a reminder on how to examine memory)*

Part 2: (3)
Iterate through each byte in the block of data in RAM and find the MAXIMUM value when treating each byte as an UNSIGNED value. Display it on the LEDs.
(Note: we’re not looking for a pair here, we’re just looking for a single byte.)
You should find 0xEC. The marker will verify that this value is on the LEDs when it hits the instruction labeled *display_maximum_done*

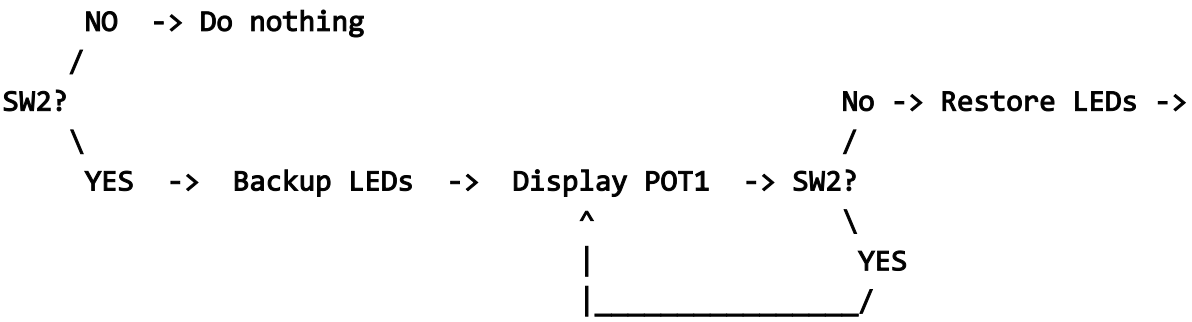
Part 3: (1)
TIM6 should be made to generate an interrupt every 1.5 seconds.
The interrupt handler should increment the value displayed on the LEDs by 1.
Naturally, it should start counting from the value found and displayed in Part 2.

Part 4: (2)
After the displayed value gets to 0xF8 (1111 1000), it should wrap back to the value found in part 2.
Ie: 0xEC, 0xED, 0xEF, …, 0xF7, 0x0xF8, 0xEC, 0xED, … etc
(Hint: to implement this, you should save the result of part 2 somewhere and then recall it when wrapping. Do not hard-code it because the maximum value will change when the marker modifies RAM. Remember, it would be dangerous to ‘save’ the part 2 value to the stack because the stack frame is automatically appended to the top of the stack when an interrupt happens. Rather save the part 2 result to a fixed memory address and recall if from there)

Part 4: (1)
When SW1 is held down, the LEDs should immediately reset back to the value found in Part 2.
They should remain at that value until SW1 is released at which point the behaviour should return to the timer incrementing whatever value happens to be on the LEDs by 1.

Part 5: (1)
If SW2 is held down, the LEDs should immediately sample POT1 and display a value on the LEDs proportional to what the pot is outputting.
Display 0 if the pot output is 0 V. Display 0xFF if 3.3 V. Linear in between.
When SW2 is released, restore whatever value was on the LEDs before SW2 was held down and go back to incrementing that value.
(Note: you may find your ADC reading only goes up to ~0xFC; that's okay.)

Suggestion:



Bonus: (1)
The brightness of the red LED at the bottom right of the board (D9) should vary smoothly between fully on and fully off as POT0 is rotated from clockwise to anticlockwise.

Marked out of: 10
Available marks: 11